Safe End-to-end Learning-based Robot Autonomy via Integrated Perception, Planning, and Control

by

Glen Chou

A dissertation submitted in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
(Electrical and Computer Engineering)
University of Michigan
2022

Doctoral Committee:

Associate Professor Dmitry Berenson, Co-chair
Associate Professor Necmiye Ozay, Co-chair
Professor Jessy Grizzle
Professor Russ Tedrake, Massachusetts Institute of Technology
Associate Professor Ram Vasudevan

Glen Chou

gchou@umich.edu

ORCID iD: 0000-0003-4444-3631

ACKNOWLEDGEMENTS

First and foremost, I want to thank my advisors Dmitry Berenson and Necmiye Ozay. It's difficult to express how fortunate I am to have benefited from the incredible breadth of your combined perspectives — working with you two has been a transformative experience. I am so grateful for all of your help during the early stages of the PhD, where you patiently taught me how to transform my messy ideas into concrete research contributions. In more recent years, I'm just as grateful for your confidence in me as a researcher, by giving me the independence to pursue the projects that I was passionate about. Some individual comments: Necmiye, your enthusiasm, commitment, and work ethic are inspiring, and you've truly led the lab by example – I hope that I can be as dedicated and as dynamic of an advisor in the future. I'm also thankful to have been able to probe the depth of your technical knowledge over the years, and that you've given me the ability to apply tools traditionally from the purview of formal methods and optimization to solve difficult problems in robotics. Dmitry, thank you for always encouraging me to think about the big picture, and for reminding me to always be trying to solve real problems - your insight into the most important open challenges in robotics have been a timeless source of inspiration. I'd also like to thank committee members Jessy Grizzle, Ram Vasudevan, and Russ Tedrake for their insightful feedback on the work presented in this dissertation. Special thanks go to Nima Fazeli for generous advice and insights on the academic job search. And I'd also like to thank Claire Tomlin and Anca Dragan for introducing me to research as an undergrad – I still take inspiration from many of the tools and techniques that you introduced me to.

I'm so lucky to have been able to work with such a varied group of PhD students and postdocs. Dale, Yu-Chi, Brad, Andrew, Tom, Peter, Johnson, Mark, it has been rewarding to see everything that you've been able to accomplish on incredibly challenging problems in manipulation, planning under uncertainty, active perception, etc. – it inspires me to dream bigger in my own work. Yunus, Liren, Kwesi, Zhe, Zexiang, Andrew, Daphna, Sunho, Ruya, Mohamad, Dan, and Xiong: your work has opened my eyes to a wide variety of applications outside of robotics, and always inspires me to learn more.

I'd like to thank my co-authors Yunus, Liren, Kwesi, Petter, Craig, and Hao for helping to make each project the best it could be. I also want to thank my advisees Craig, Adarsh, Hao, Yating, and Jiayi: thank you for giving me the opportunity to work with you, and for enabling me to learn how to advise students. Special thanks go to Craig for working with me on the trusted domain project – I'm looking forward to continuing our collaboration in the future. I'm so proud of the work that all of

you have done, and I'm excited to see where all of you go next.

To Connie, Victor, Michelle, Romulo, Dennis, Jenny, Kwesi, Yunus, Daphna, Derek, Eric, Jess, Miles, Sam, and Andrew: thanks for supporting me and for creating so many wonderful memories over the years – I'm so lucky to be able to call you my friends. And last but not least, I must thank my parents for their unyielding and unconditional support throughout the PhD. Thank you for imbuing in me the importance of a hard work ethic, and for always telling me that I can accomplish anything that I put my mind to.

TABLE OF CONTENTS

LIST OF FIGURES

Figure

LIST OF TABLES

Table

LIST OF APPENDICES

Appendix

ABSTRACT

Trustworthy robots must be able to complete tasks reliably while obeying safety constraints. While traditional methods for constrained motion planning and optimal control can achieve this if the environment is accurately modeled and the task is unambiguous, future robots will be deployed in unstructured settings with poorly-understood or inaccurate dynamics, observation models, and task specifications. Thus, to plan and perform control, robots will invariably need data to learn and refine their understanding of their environments and tasks. Though machine learning provides a means to obtain perception and dynamics models from data, blindly trusting these potentially-unreliable models when planning can cause unsafe and unpredictable behavior at runtime. To this end, this dissertation is motivated by the following questions: (1) To refine their understanding of the desired task, how can robots learn components of a constrained motion planner (e.g., constraints, task specifications) in a data-efficient manner? and (2) How can robots quantify and remain robust to the inevitable uncertainty and error in learned components within the broader perception-planning-control autonomy loop in order to provide system-level guarantees on safety and task completion at runtime?

To address the first question, we propose methods that use successful human demonstrations to learn unknown constraints and task specifications. The crux of this problem relies on learning what not to do (i.e., behavior violating the unknown constraints or specifications) from only successful examples. We make the insight that the demonstrations' approximate optimality implicitly defines what the robot should not do, and that this information can be extracted by simulating lower-cost trajectories and by using the Karush-Kuhn-Tucker (KKT) optimality conditions. These strong optimality priors make our method highly data-efficient. We use these methods to learn a broad class of constraints, including nonconvex obstacle constraints, and linear temporal logic formulas, which can describe complex temporally-extended robotic tasks. We demonstrate that our constraint-learning methods scale to high-dimensional systems, e.g., learning to complete novel constrained navigation tasks for a simulated 12D quadrotor and multi-stage manipulation tasks on a 7DOF arm (both simulated and in the real world).

To address the second question, we develop methods addressing uncertainty in A) constraints learned from demonstrations and B) dynamics models and perception modules learned from data. To quantify constraint uncertainty, we extract the set of constraints that are consistent with the demonstrations' KKT conditions (i.e., a belief over constraints), which is done by solving a sequence of robust mixed integer programs. We show that the robot can plan probabilistically-safe trajectories using this constraint belief, which can be updated using constraint data gathered in execution. To address uncertainty when planning with learned dynamics models of underac-

tuated systems controlled with high-dimensional (image) observations, we estimate bounds on the error of the learned models inside a domain around their training data. Using tools from contraction theory, we propagate this model error bound into a trajectory tracking error bound. This tracking bound is used to constrain the planner to only return plans that can be safely tracked, with high probability, in spite of the errors in the perception and dynamics. We demonstrate that these theoretical guarantees translate to success in simulation, enabling safe task completion at runtime on a variety of challenging high-dimensional, underactuated systems using rich sensor observations (e.g., RGB-D images) in the feedback control loop.

CHAPTER I

Introduction

The development of autonomous robots has the potential to revolutionize society, with applications as diverse as assistive home robots, self-driving cars, manufacturing, construction, disaster response, last-mile drone delivery, and agriculture. Like with any technology, such robots should be able to reliably and safely perform their intended tasks. Many applications for robot autonomy, including several settings mentioned above, are safety-critical – often because the cost of failure in such scenarios can be high, and because humans are in the loop. In such environments, it is of paramount importance for robots to be able to provide some (probabilistic) guarantee on safety and capability when performing their tasks. However, deploying robots outside of a controlled lab setting brings about a variety of roadblocks in achieving this goal of safe and reliable autonomy. To better understand these challenges, we will first discuss a common operating loop for autonomous robots: the perception, planning, and control pipeline.

The goal of perception is to take in raw information about the world from sensors (e.g., in the form of range measurements from a LiDAR, vision from an onboard camera, force measurements from tactile sensors, etc.) and process this data in order to obtain a concise representation of the world which is useful for planning (e.g., a state estimate). In recent years, deep learning-based approaches have enabled major advances in perception, especially through computer vision, where large improvements in object classification and detection have given robots an improved understanding of their surroundings (e.g., *Levine et al.* (2016)). However, the reliability of deep learning-based models is notoriously difficult to verify because of their complexity (*Liu et al.*, 2021). Moreover, the quality of these models' predictions is sensitive to the training data; in particular, if the robot senses images which are far from the training data (i.e., out-of-distribution), the output of the machine learning-based model is not guaranteed to generalize and can be entirely inaccurate (*Shen et al.*, 2021). Adding to these challenges is the quality of the sensor measurements that robots can obtain when deployed in real environments: measurements are often inaccurate and incomplete, e.g., due to noise and occlusions; thus, the robot's surroundings may be inaccurately estimated. These combined issues can make it quite difficult to guarantee the correctness of the perception module. In this thesis, we develop a method for using perception error to guide reliable downstream planning and control from

images (Chapter X).

Next, the aim of the planner is to determine what actions the robot should take in order to complete its task. Specifically, it takes as input 1) the estimated state of the robot and the environment from the perception module, 2) a dynamics model which predicts how the robot and environment will evolve over time, and 3) a specification of the task. It then outputs a trajectory which satisfies the task specifications (while possibly also optimizing some task-dependent preferences) while being consistent with the dynamics and the perception information. Assuming that the state estimate is correct, the dynamics are entirely accurate, and the task specification is unambiguous, there are many tools for synthesizing a plan that can be safely executed with formal guarantees (e.g., constrained motion planning (*Berenson*, 2011), constrained model predictive control (*Mayne et al.*, 2000), and linear temporal logic planning (*Kress-Gazit et al.*, 2009)). However, these assumptions almost never hold, and designing algorithms that can retain safety and task completion guarantees in the presence of these imperfect assumptions is the main focus of the work presented in this thesis. One primary issue is that the robot's task is often ambiguous – one popular way for humans to specify tasks to a robot is with demonstrations (*Argall et al.*, 2009); however, these demonstrations can be imperfect and may not translate to a unique planning problem (e.g., if we assume that the planner solves an optimization problem to find a plan, there can be many possible cost functions and constraints which are consistent with the demonstrations (*Ng and Russell*, 2000)). As these true task constraints, which encode the conditions for safe and reliable task execution, are not known unambiguously, it can be difficult to find a plan that safely completes the true task. We develop methods that tackle these issues in Chapters III - VII. Moreover, the dynamics model used to make predictions is rarely correct – modeling assumptions inevitably break down in the face of factors like friction, slip, and bending, etc. Even dynamics models which are learned directly from data (e.g., via deep learning) can often be inaccurate, and can only be expected to be reliable near their training data. We propose a method for planning safely with learned dynamics in Chapters VIII-IX.

Finally, the planned trajectory is passed to a controller, which aims to track the planned trajectory. However, in the presence of model uncertainty, the planned trajectory may not be safely trackable for the true dynamics. As a result, we argue that for safe execution of a plan, it is crucial to generate plans which consider the capabilities of the downstream tracking controller – this is the approach taken by Chapters VIII-X. Further challenges arise when the system to be controlled is underactuated – in this case, it is crucial to think about how this underactuation can exacerbate and propagate tracking error when generating a plan (Chapter X).

1.1 Thesis Overview

In the light of these open problems, we will design methods which make strides towards solving the following challenges in this thesis. After discussing some related work in Chapter II, we move to the contributions of this thesis:

- Learning safety constraints and temporally-extended task requirements from

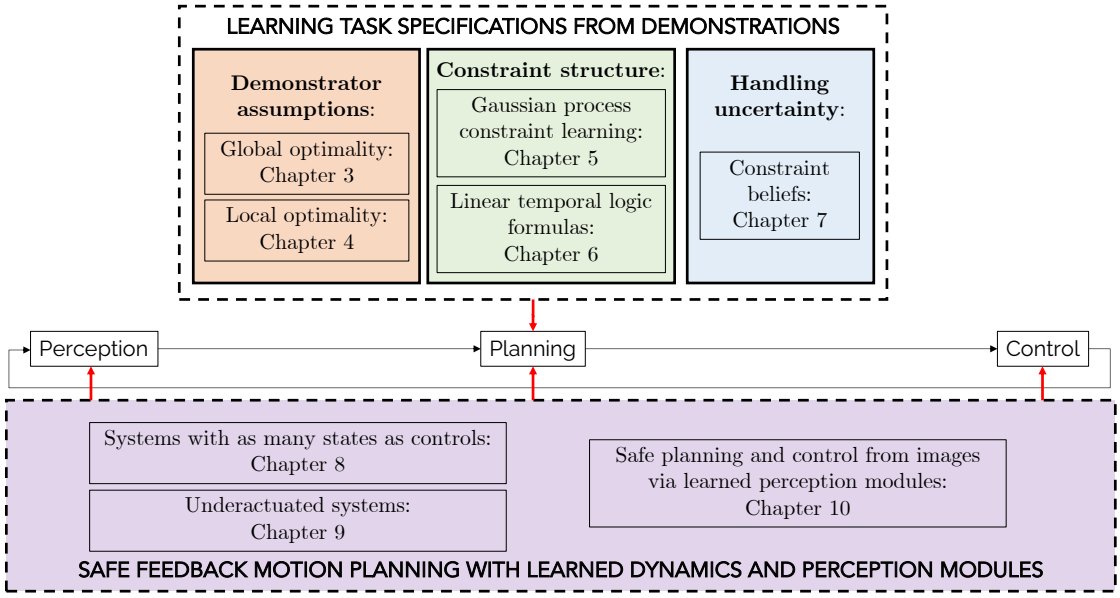


Figure 1.1: An overview of the methods and contributions of this thesis

demonstrations (Chapter III, IV, V, VI)

- Planning to complete tasks despite uncertainty in their specification (Chapter IV, V, VII)
- Quantifying task uncertainty and how to update that uncertainty based on data seen online (Chapter V, VII)
- Long-horizon motion planning with learned dynamics models with safety guarantees (Chapter VIII, IX)
- Having well-calibrated measures of model error, which is crucial for obtaining practical safety guarantees (Chapter VIII, IX, X)
- Probabilistically verifying safety and task completion through the integration of perception, planning, and feedback control (Chapter VIII, IX, X)
- Safety analysis that scales to high-dimensional nonlinear dynamical systems (Chapter VIII, IX)
- Safety analysis that scales to high-dimensional nonlinear observations, with challenges caused by partial observation (Chapter X)

We describe these contributions in more detail in the following, and categorize them visually in Figure 1.1.

1.1.1 Summary of Contributions

Part 1: Learning task specifications for safe planning: In the first part of this thesis, we will discuss how we can learn unknown task specifications from demonstrations. These specifications come in the form of constraints, both time-invariant and time-varying (linear temporal logic formulas). We will also touch on how we can quantify the uncertainty in the learned task and update it with execution data.

In Chapter III, we discuss how approximately globally-optimal demonstrations can be used to learn the unknown constraints. The main idea is to exploit knowledge of the dynamics and approximate knowledge of the demonstrator's cost function in order to sample dynamically-feasible trajectories which have lower cost than the demonstrations, and are thus constraint-violating. We show that the demonstrations and these synthetically-generated unsafe trajectories can be used to learn both grid-based and parametric representations of the unknown constraint by solving an associated mixed integer program. Both representations have their own strengths and drawbacks – a grid-based parameterization requires minimal *a priori* information about the structure of the unknown constraint, but suffers from poor scalability in high dimensions, whereas a parametric constraint representation requires more knowledge on the constraint structure but scales more gracefully.

In Chapter IV, we relax the global optimality assumption on the demonstrations to one of local optimality; that is, that there exists no local perturbation to the demonstration which enables a decrease in cost without the violation of some constraint. This is formalized using the Karush-Kuhn-Tucker (KKT) conditions from constrained optimization, which are necessary conditions for a candidate solution to an optimization problem to be locally-optimal. In this chapter, we assume a known parametric representation of the constraint and also assume that the cost function may be uncertain, and search for constraint and cost function parameters that jointly make the demonstrations satisfy their repspective KKT conditions. Put another way, we use the KKT conditions to restrict the set of possible parameters that can explain the demonstrations. Relative to the work discussed in Chapter III, the KKT conditions provide a more efficient, one-shot means of enforcing a notion of demonstrator optimality, compared to exhaustive trajectory sampling. We demonstrate that this approach can learn constraints for a 7DOF arm and a quadrotor in simulation.

In Chapter V, we address a major drawback of the method presented in Chapter IV – the requirement of a known constraint parameterization. In this chapter, we show how the KKT conditions of the demonstrations can be used in a parametrization-agnostic fashion – more specifically, we show that the KKT conditions can provide information about which points on the demonstrations must be tight against the unknown constraint boundary, as well as the gradient of the constraint evaluated at those points. Compared to the work in Chapter IV, which uses the KKT conditions to restrict the set of consistent parameters, for a fixed and known constraint representation, the work in this chapter uses the KKT conditions to directly restrict the set of consistent locations and gradients of the unknown constraint without assuming any representation. We use this constraint value and gradient information to train a Gaussian process (GP)-based constraint representation, and we develop a novel plan-

ner which exploits the jointly Gaussian structure of the GP representation to plan trajectories which exactly satisfy the learned constraint with a desired probability. Our approach enables a natural means of imposing priors over the constraint (e.g., smoothness, via the GP kernel) and a means to incorporate constraint gradient information, while the grid-based representation in Chapter III cannot. Our approach enables constraint learning on simulated quadrotor and arm examples with minimal *a priori* assumptions on the constraint structure.

In Chapters III - V, we restrict ourselves to learning time-invariant constraints. In Chapter VI, we move towards learning multi-stage constrained tasks by showing how similar notions of optimality can be used to learn linear temporal logic (LTL) formulas from suboptimal demonstrations. Temporal logic provides a means to specify complex temporally-extended tasks with time-varying, history-dependent constraints. Specifically, we use the KKT conditions together with a counterexample-guided falsification approach to learn the atomic propositions (defining low-level state space constraint regions) and logical structure of the unknown LTL formula (determining the high-level flow of the task), respectively. Compared to the constraints learned in previous chapters, we can model and learn complex, multi-stage tasks, like a sequential multi-object retrieval and delivery task on a real 7DOF arm.

Finally, in Chapter VII, we tackle the ill-posedness of the constraint learning problem – specifically, the fact that there may be (infinitely) many constraints which are consistent with the optimality conditions of the demonstrations. To address this, we consider the case of a parametric constraint representation and construct a "belief" over constraint parameters by obtaining the set of all constraint parameters which can make the demonstrations optimal. Then, we demonstrate how we can plan probabilistically-safe trajectories under this belief, and how we can perform belief updates in execution to refine the constraint uncertainty if additional information about the constraint is observed online. We evaluate the approach on uncertain constraint learning problems demonstrated on a quadrotor and 7DOF arm in simulation.

Part 2: Safe planning and feedback control with learned dynamics and perception modules: In the second part of this thesis, we will discuss how we can plan with complex learned models of high-dimensional systems while guaranteeing safety and goal reachability in execution, when controlling from high-dimensional observations (e.g., images) generated by the system at runtime. The method is first developed for the state-feedback case for systems with the same number of control inputs as states (Chapter VIII), and is then extended to a class of underactuated systems (Chapter IX). Finally, the method is extended to the output-feedback setting for planning and control from images in Chapter X.

In Chapter VIII, we develop an algorithm for feedback motion planning of systems with unknown dynamics which provides probabilistic guarantees on safety, reachability, and goal stability. Using a given dataset of transitions from the unknown dynamics, we learn a neural-network represented control-affine approximation of the dynamics and estimate a bound on the model error within a domain around the training data. This bound is obtained by estimating, with high probability of validity, the Lipschitz constant of the error within this domain. Moreover, for systems with the same number of control inputs as states, we develop a one-step feedback law which

can stabilize the true system in execution around a planned trajectory. We embed conditions on the feedback law's existence as a constraint in a sampling-based planner, which we call LMTD-RRT, that biases the planner towards returning plans that will experience low model error in execution, and which can be safely tracked with a desired probability. We evaluate our approach on models of simulated fully-actuated quadrotor and arm dynamics and show it enables safe plan execution, whereas baselines return plans that cannot be safely tracked.

Then, in Chapter IX, we extend the work in Chapter VIII to provide similar guarantees for a broad class of underactuated systems by leveraging the rich theory of control contraction metrics (CCMs). Specifically, we learn a CCM together with a contracting controller and a control-affine dynamics approximation; all three of these models are represented with neural networks. Our method certifies that the learned controller contracts with respect to the learned CCM (with high probability) everywhere within a domain around the training data and also estimates a model error bound within that domain, which is a valid upper bound with high probability. We derive novel bounds on the tracking error that the system can experience in execution while tracking a plan, which can be obtained given a CCM and model error bound. We show that these tracking tubes can be integrated into a similar planning framework as LMTD-RRT, which we call LMTCD-RRT, which can also bias planning towards regions of the space which have low model error and where the system can be tracked safely, with high probability, despite its underactuation. In particular, the flexibility of our framework enables safe planning on high-dimensional underactuated systems such as a 22-dimensional rope manipulation task in simulation.

Finally, in Chapter X, we extend the work in Chapter IX to safely steer uncertain, potentially underactuated, nonlinear systems to a desired goal region (with high probability), when controlling the system using feedback from high-dimensional RGB(-D) observations and a learned perception module. To achieve this, we leverage contraction-based state estimators in interconnection with the contraction-based controllers described in the previous paragraph, and derive trajectory tracking error bounds for the interconnected estimator and controller when the interconnection is subjected to errors arising from learning errors in the perception module and uncertainties in the dynamics model. We use this tracking error bound to inform the motion planner, constraining it to stay in regions where both the perception module and the dynamics are accurate; we call this modified planner CORRT. Trajectories generated by CORRT are guaranteed to safely reach the goal region with high probability when using the contraction-based estimator to estimate the state from the image observations generated online, and then using that estimated state to perform contraction-based feedback control to stabilize around the planned nominal trajectory. We demonstrate that this approach in simulation, safely steering a nonholonomic ground vehicle and an underactuated planar quadrotor to a desired goal region. We also demonstrate that our approach can be used for active perception (i.e., it can plan trajectories that reduce estimation error below a desired threshold), by using the generated estimation error bounds to guide planning towards regions where the perception module is more accurate, and demonstrate this on a pose estimation problem on a 7DOF arm grasping problem.

1.1.2 Additional Contributions

Some work done throughout my PhD will not be covered in this thesis:

- an efficient algorithm for online time series segmentation *Chou et al.* (2018b)
- generation of challenging test cases for autonomous driving controllers via controlled invariant sets *Chou et al.* (2018c)
- learning constraints from visibility-constrained demonstrators *Knuth et al.* (2021b)
- guaranteeing safety in decentralized multi-agent systems via notions of responsibility-sensitive safety and controlled invariant sets *Rutledge et al.* (2021)

CHAPTER II

Related Work

We organize the research areas related to this thesis in three main categories: 1) learning from demonstration (Sec. 2.1), which is relevant to our contributions in learning constrained task specifications from demonstrations (Chapters III-VII), 2) motion planning (Sec. 2.2), which is relevant to our contributions in planning safely with learned dynamics models (Chapters VIII-IX), and 3) perception-based control (Sec. 2.3), which is relevant to our contributions in ensuring safety and goal reachability for planning and control from images (Chapter X).

2.1 Learning from Demonstration

The focus of Chapters III - VII is on leveraging demonstrations to learn constrained task specifications for downstream planning tasks. Our work draws upon and contributes to a large body of work in the area of learning from demonstration (LfD) (see *Argall et al.* (2009)), also known as imitation learning (IL) (see *Osa et al.* (2018)). At a high level LfD/IL algorithms fall into two categories: behavior cloning algorithms, which directly aim to learn a policy that imitates the demonstrations, and inverse optimal control (IOC) / inverse reinforcement learning (IRL) algorithms, which aim to learn a cost/reward function that recovers the demonstrations when optimized. Our work is closer in spirit to IOC and other approaches that aim to learn more parsimonious representations of the demonstrated behavior, which we will overview in this section.

2.1.1 Inverse optimal control

Inverse optimal control (IOC) and inverse reinforcement learning (IRL) encompasses a broad set of techniques that seeks to learn a reward function that can explain observed demonstrations. In general, IOC (first introduced by *Kalman* (1964) and studied extensively in the subsequent decades (*Johnson et al.*, 2013; *Keshavarz et al.*, 2011; *Ratliff et al.*, 2006)) assumes that the demonstrations are the solution to an unconstrained (typically deterministic) optimization problem (referred to as the "demonstrator's problem"), where the cost function being optimized is unknown, and its recovery is the goal of the learning problem. The motivation for this problem

setup is that in some settings, the cost function is a generalizable representation of the desired behavior; thus, by recovering the cost function from the demonstrations, the robot may be able to synthesize similar behaviors that imitate the demonstrations under similar conditions. As will be argued in Chapters III - VII, assuming that the demonstrator is solving an unconstrained problem is insufficient to accurately model behaviors that arise from hard constraints (e.g., safety constraints like avoiding obstacles or keeping a cup upright to avoid spills), or more general task constraints (like first retrieving a mug before going to the coffee machine). Moreover, these hard constraints are generally unknown and also need to be recovered from the data – this is the core problem studied in Chapters III - VII. While there exists prior work that models the demonstrator's problem as a constrained optimization problem, e.g., *Englert et al.* (2017); *Menner et al.* (2019), thereby acknowledging the presence of hard constraints in the demonstration, these methods do not infer the constraints, instead opting to learn a cost function which is consistent with the optimality conditions of the constrained problem, assuming that the constraints themselves are known. Other similar lines of work aim to address the fact that the demonstrator may be trading off multiple objectives (e.g., *Amin et al.* (2017); *Babes et al.* (2011); *Choi and Kim* (2012); *Nguyen et al.* (2015)); however, these approaches essentially soften the unknown constraints to a reward/cost penalty, and thus are not designed to guarantee constraint satisfaction.

IRL takes a similar approach as IOC, but is instead formalized in a probabilistic framework, assuming that demonstrations are observations from an optimal policy in a Markov Decision Process (MDP). Then, the goal is to recover the unknown reward function which induces that optimal policy. First proposed in *Ng and Russell* (2000), IRL has attracted a substantial amount of attention over the years *Abbeel and Ng* (2004); *Abbeel et al.* (2010); *Ramachandran and Amir* (2007); *Sadigh et al.* (2017); *Ziebart et al.* (2008). The work in this thesis is built off the deterministic setting of IOC rather than IRL, though recent work has begun to investigate similar constraint learning problems in the IRL setting *Scobee and Sastry* (2020).

A core challenge of IOC/IRL is the underspecification of the cost/reward function learning problem – in general, there may be (infinitely) many cost/reward functions which make the demonstrations optimal. For instance, a cost/reward function which is zero everywhere (e.g., making the demonstrator's problem a feasibility problem) makes any demonstration optimal. To address this challenge, IOC/IRL methods often regularize the problem via additional priors, e.g., via the principle of maximum entropy *Ziebart et al.* (2008). Other methods attempt to learn a distribution over possible reward functions *Ramachandran and Amir* (2007). In Chapter VII, we will provide our own solution to addressing the underspecification problem for constraint learning, which we do by constructing a belief over possible constraints.

In recent years, IOC/IRL techniques have begun to take advantage of new advances in machine learning which have reduced the need for feature engineering – a commonplace requirement for previous IOC/IRL algorithms, most of which assume that the unknown cost/reward function can be described by a linear combination of known feature functions. In contrast, recent approaches that leverage deep neural network and Gaussian process cost/reward function representations have been

able to break free of this assumption, e.g., *Finn et al.* (2016); *Levine et al.* (2011); *Wulfmeier et al.* (2016). In Chapter V, we will also leverage Gaussian processes to learn constraints without an *a priori* known constraint representation.

2.1.2 Safe imitation learning

In recent years, there have been several algorithms for safe or uncertainty-aware imitation learning, which build upon traditional imitation learning algorithms like DAGGER *Ross et al.* (2011) by estimating the uncertainty in the learned policy *Brown et al.* (2020); *Choi et al.* (2018); *Cui et al.* (2019); *Menda et al.* (2019); *Thakur et al.* (2019); *Zhang and Cho* (2016). However, these approaches only use the uncertainty as a sign to switch to a safe backup policy *Choi et al.* (2018); *Cui et al.* (2019); *Menda et al.* (2019), to evaluate the quality of a given policy *Brown et al.* (2020); *Thakur et al.* (2019), or to query the demonstrator for more data *Zhang and Cho* (2016). In our setting, we do not assume access to an interactive demonstrator or an *a priori* known safe policy; instead, we must directly quantify and use the uncertainty in the demonstrations to determine how to behave. This will be discussed further in Chapter VII.

2.1.3 Constraint learning

While our constraint learning work is among the first set of methods that explicitly ties constraint learning to inverse optimal control (i.e., using optimality assumptions on the data to regularize the constraint learning problem), there is still relevant prior work in the area of constraint learning from demonstrations.

There exists prior work in learning geometric constraints in the workspace. In *Armesto et al.* (2017), a method is proposed for learning Pfaffian constraints, recovering a linear constraint parametrization. In *Pérez-D'Arpino and Shah* (2017), a method is proposed to learn geometric constraints which can be described by the classes of considered constraint templates. Other methods aim to learn task space equality constraints (*Lin et al.*, 2015, 2017). In contrast, our work formulates and can learn constraints in arbitrary constraint spaces (as long as that space is known), and is not limited to geometric constraints in the task space (e.g., we can learn state space constraints, control constraints).

Learning local trajectory-based constraints has also been explored in the literature. The method in *Li and Berenson* (2016) samples feasible poses around waypoints of a single demonstration; areas where few feasible poses can be sampled are assumed to be constrained. Similarly, *Mehr et al.* (2016) performs online constraint inference in the feature space from a single trajectory, and then learns a mapping to the task space. The methods in *Calinon and Billard* (2007, 2008); *Pais et al.* (2013); *Ye and Alterovitz* (2011) also learn constraints in a single task. These methods are inherently local since only one trajectory or task is provided, unlike our constraint learning work, which aims to learn a global safety constraint or task specification that is shared across different trajectories.

2.1.4 Learning temporal logic formulas from data

One advantage of representing the demonstrated task using constraints is that it is easier to unambiguously specify a complex multi-stage task with a set of hard constraints instead of as penalties in a reward function; generally, IRL-based methods can struggle to represent multi-stage, long-horizon tasks *Krishnan et al.* (2019). In our work, we will use the formalism of linear temporal logic (LTL) to model our constraints; see Chapter VI. We now overview some previous approaches that attempt to use IRL/IOC to model multi-stage tasks. In *Krishnan et al.* (2019); *Ranchod et al.* (2015), the proposed mehtods learn sequences of reward functions, but in contrast to temporal logic, these methods are restricted to learning tasks which can be described by a single fixed sequence. Temporal logic *Baier and Katoen* (2008); *Kress-Gazit et al.* (2009) generalizes this, being able to represent tasks that involve more choices and can be completed with multiple different sequences. Some work *Papusha et al.* (2018); *Zhou and Li* (2018) aims to learn a reward function given that the demonstrator satisfies a known temporal logic formula; in our work, we will learn both jointly, as generally the task (which is encoded in the LTL formula) is unknown.

There is extensive literature on inferring temporal logic formulas from data via decision trees *Bombara et al.* (2016), genetic algorithms *Bufo et al.* (2014), and Bayesian inference *Shah et al.* (2018); *Vazquez-Chanlatte et al.* (2018). However, most of these methods require positive and negative examples as input *Camacho and McIlraith* (2019); *Kong et al.* (2014, 2017); *Neider and Gavran* (2018). This requirement for positive and negative examples is ill-suited for robotics applications, as this implies we would need to collect negative (i.e., unsafe demonstrations, or demonstrations that fail the task) in order to learn the desired formula. In contrast, our work (Chapter VI is designed to only use positive examples. Other methods require a space-discretization *Araki et al.* (2019); *Vaidyanathan et al.* (2017); *Vazquez-Chanlatte et al.* (2018), while our approach in Chapter VI learns LTL formulas in the original continuous space. Some methods learn AP parameters, but do not learn logical structure or perform an incomplete search, relying on formula templates *Bakhirkin et al.* (2018); *Leung et al.* (2019); *Xu et al.* (2019), while other methods learn structure but not AP parameters *Shah et al.* (2018). Perhaps the method most similar to our proposed approach in Chapter VI is *Jha et al.* (2019), which learns parametric signal temporal logic (pSTL) formulas from positive examples by fitting formulas that the data tightly satisfies. However, the search over logical structure in *Jha et al.* (2019) is incomplete, and tightness may not be the most informative metric given goal-directed demonstrations. More broadly, to our knowledge, the method presented in Chapter VI is the first method in the literature for learning LTL formula structure and parameters in continuous spaces on high-dimensional systems from only positive examples.

2.2 Motion Planning

Motion planning has been widely studied in the past few decades, with algorithms ranging from discretization-based methods like A* *Hart et al.* (1968), sampling-based approaches like probabilistic roadmaps *Kavraki et al.* (1996) (PRMs) and

rapidly-exploring random trees *LaValle and James J. Kuffner* (2001) (RRTs), and gradient/sampling-based trajectory optimization algorithms like CHOMP *Zucker et al.* (2013), TrajOpt *Schulman et al.* (2014), and MPPI *Williams et al.* (2016). However, despite the heavy attention that motion planning has received, several major challenges remain. Arguably one of the most serious difficulties lies in how to handle the uncertainty and inaccuracy present in the models, environments, and task specifications that are given as input to these motion planning algorithms. This section is most related to our work in Chapters VIII-X, which are proposed motion planners that estimate and consider the errors in learned dynamics and perception modules to constrain a motion planner to generate trajectories that can safely reach the goal when tracked at runtime. In the following, we will overview several areas of research in this area.

2.2.1 Feedback motion planning

Feedback motion planning (*LaValle*, 2006, Chapter 8) consists of a broad class of methods that aim to address dynamics uncertainty in motion planning by designing tracking controllers around nominal trajectories generated by a motion planner. The general idea is that if the tracking controller can keep the system within some bound r of the nominal trajectory, if the planned trajectory remains at least r distance away from any obstacles, the system can remain collision-free in execution.

Prominent methods in this vein include LQR-trees *Tedrake* (2009) and funnel libraries *Majumdar and Tedrake* (2017). These approaches design feedback controllers (either based on LQR or on polynomial controllers synthesized via sum-of-squares programming) around a set of nominal trajectories, and then compute Lyapunov functions to analyze the region of attraction of the closed-loop system. Then, for a known disturbance description, these methods can determine if the system, starting from some initial condition in the region of attraction, can be guaranteeably steered to a desired location. While these methods are powerful, they assume a valid disturbance description has been provided, which can often be difficult to obtain, especially when planning with a learned model of the dynamics (see Chapters VIII, IX).

Another set of methods for feedback motion planning are based on exact reachability analysis performed by solving Hamilton-Jacobi (HJ) partial differential equations via level-set approaches *Mitchell et al.* (2005). At a high level, HJ analysis formulates the robust control problem as a minimax game between the controller and an adversary (e.g., some disturbance arising from the environment). This analysis enables the computation of controlled invariant sets *Bertsekas* (1972), which describe the set of states that a system can remain within given the disturbance description and feedback controller. These tools have been applied to robust tracking control in *Herbert et al.* (2017), which wraps trajectories planned with a low-fidelity model with a corresponding controlled invariant set, where the adversary here comes from the model mismatch between the low-fidelity planning model and the true high-fidelity dynamics. Unfortunately, HJ-based methods generally have trouble scaling to high-dimensional systems due to the grid-based discretization required to solve the HJ PDE. Moreover, an accurate disturbance bound is also required in this setting.

There are also other methods which use approximate reachability analysis to obtain better scaling, e.g., *Kousik et al.* (2017), which computes overapproximations of forward reachable sets for a high-fidelity dynamics model tracking trajectories planned using a low-fidelity model (similar to *Herbert et al.* (2017)), and *Singh et al.* (2018), which utilizes sum-of-squares programming to obtain better scaling than *Herbert et al.* (2017). Similarly, in all of these methods, a known bound on the model error is required.

Tube model predictive control (tube MPC) is another set of similar approaches that wrap a nominal trajectory with a tracking controller, keeping the system within a "tube" of states around the nominal trajectory. Tube MPC has been widely studied for linear systems (e.g., *Mayne et al.* (2005); *Rakovic et al.* (2012)) and more recently also for nonlinear systems *Köhler et al.* (2021). However, as with the methods previously discussed, these approaches still require an accurate bound on the uncertainty, which can be difficult to obtain.

Finally, a growing body of work leverages contraction theory *Lohmiller and Slotine* (1998) to obtain tracking tubes around nominal trajectories. The theory of (control) contraction metrics studies the incremental (stabilizability) stability of a system *Manchester and Slotine* (2017), that is, it analyzes how the distance between trajectories of the system change with time. At a high level, the idea is that if two trajectories of the system always converge to each other, then the system is determined to be contracting. This is a particularly useful notion of stability for the purpose of trajectory tracking – we precisely want to determine if the system can be made to track (i.e., converge) to the planned trajectory, within some bound, when subjected to disturbance. Recent work in this area has applied contraction theory for the purpose of feedback motion planning *Singh et al.* (2019); however, these methods require polynomial dynamics and again require an accurate bound on the model uncertainty. Other methods *Lakshmanan et al.* (2020); *Lopez et al.* (2021) apply contraction to adaptive control under known model uncertainty structure, i.e., the uncertainty lies in the range of known basis functions; similarly, this uncertainty structure is generally unknown *a priori* in our setting.

Overall, in Chapters VIII and IX, we take steps towards designing feedback motion planning algorithms that can provide probabilistic guarantees on safety and goal reachability when planning with learned dynamics. This is done by estimating a bound between the learned dynamics and true system with statistical techniques, and using it to bound the impact of the error on the downstream planning and feedback control modules. We demonstrate the approach on several nonlinear systems, including a 4D nonholonomic car, a 6D planar quadrotor, and a 22D rope dragging problem.

2.2.2 Planning under uncertainty

Instead of utilizing a deterministic bound on the disturbance, a family of alternative approaches imposes a stochastic description of the uncertainty in planning. These methods perform planning via chance-constrained optimization, approximate planning in belief space, and most generally, by solving partially observable Markov

Decision Processes (POMDPs).

In chance-constrained optimization, the decision-maker seeks to find a solution that satisfies a set of constraints with a desired probability. In the context of chance-constrained motion planning, the uncertainty may arise from sensing *Burns and Brock* (2007), state estimation *Bry and Roy* (2011), motion *Aoude et al.* (2013), and the environment. These methods typically return an open-loop plan which satisfies the constraints with a desired probability, and often assume linear dynamics and Gaussian uncertainty for tractability (since Gaussian uncertainty propagated through linear dynamics remains Gaussian). While some recent approaches have begun to relax these assumptions on the dynamics *Lew et al.* (2020) and on the uncertainty *Wang et al.* (2020), feedback control is generally required to keep the uncertainty from growing too quickly in order for long-horizon planning to be tractable; this combination of planning and control under uncertainty is examined in Chapters VIII-IX.

A related body of work seeks to plan under environment or obstacle uncertainty. *Blackmore et al.* (2006) plans in an uncertain obstacle field by solving a chance-constrained optimization, which is made tractable by assuming polytopic obstacles and linear Gaussian dynamics. Simiarly, *Luders et al.* (2010) and *Luders et al.* (2013) incorporate chance constraints into an RRT *LaValle and James J. Kuffner* (2001) and RRT* *Karaman and Frazzoli* (2010) planner, respectively, returning probabilistically-safe plans under similar structural assumptions as *Blackmore et al.* (2006). Other methods directly assume a Gaussian distribution on each halfspace constraint in a polytopic obstacle *Axelrod et al.* (2017) or assume constraint convexity *Vitus et al.* (2016). Alternatively, methods based on the "scenario approach" sample possible constraints/obstacles and enforce them all, but this can often leads to infeasibility and poor computation time in planning *Grammatico et al.* (2016). Other approaches attempt to perform high-quality short-range planning in unknown environments, e.g., *Janson et al.* (2018); *Richter et al.* (2015) and assume minimal knowledge of the global map. These methods are related to our work on handling uncertainty in the learned constraint during planning (Chapters V and VII). Relative to the state-of-the-art, our contributions in Chapter VII are to show how demonstrations can reduce environment uncertainty, yielding better global plans, and to provide a method to plan chance-constrained trajectories for a class of uncertainty distributions with complex support. In Chapter V, we present a method for chance-constrained planning with Gaussian process (GP)-represented constraints, and it does not require structural assumptions on the constraints and dynamics.

A different set of approaches directly reason about the probability distribution over states which the system may be currently at; this distribution is referred to as the "information state" or "belief". Planning over these distributions is known as belief space planning *Bonet and Geffner* (2000); *Kaelbling et al.* (1998), and can be formulated as a partially observable Markov Decision Process (POMDP). While belief space planning enables a principled treatment of uncertainty and partial observability in planning, POMDPs are known to be intractable to solve exactly. While there has been progress in recent years towards more scalable approximate POMDP solvers by restricting attention to the set of reachable beliefs *Kurniawati et al.* (2008) or by sampling *Somani et al.* (2013), POMDPs remain challenging to solve in general

settings. Thus, several belief space planning algorithms make additional simplfying assumptions, such as linear dynamics and Gaussian uncertainty *Agha-mohammadi et al.* (2014), *Prentice and Roy* (2009), *van den Berg et al.* (2011), or by making simplifying assumptions on future uncertainty (i.e., by assuming that future uncertainty takes its maximum likelihood value *Jr. et al.* (2010)). While our work in Chapters VIII and IX assume that the system is fully observable, we are able to make stronger correctness guarantees while making less stringent assumptions on the dynamics and motion uncertainty. Moreover, to address this limitation, we will discuss our work on safe planning from observations with learned perception modules in Chapter XI.

Finally, there is some limited work in the context of planning with uncertain linear temporal logic formulas. In particular, *Shah et al.* (2020) plans given a belief over possible LTL formulas and proposes different criteria which can guide the planning (e.g., satisfying the most likely formula, or satisfying as many formulas as possible, etc). Also related to this problem is the minimum violation LTL planning problem *Tumova et al.* (2013), where it may be infeasible to find a trajectory which satisfies a given formula, so instead the planner must return a trajectory which violates the formula as minimally as possible.

2.2.3 Learning-based planning

Recent years have seen an explosion of interest in learning-based approaches for motion planning and control. Of specific interest in our setting is model-based reinforcement learning (MBRL); see *Moerland et al.* (2020) for a detailed survey. There are several flavors of MBRL. Some approaches assume that the dynamics are known, and aim to learn a value function or a policy for completing some task from data. Other approaches do not assume that the dynamics are known, and attempt to learn a value function or a policy jointly with a dynamics model. Also related to MBRL is the problem of planning with learned dynamics without construction of a globally-valid value function or policy. In the following, we will overview methods that plan with learned models, as well as safety-focused variants of policy learning algorithms.

There is a large body of work that plans with learned dynamics models. For instance, *Hafner et al.* (2019); *Ichter and Pavone* (2019); *Watter et al.* (2015) learn latent spaces (either from observations or from a higher-dimensional state) along with dynamics models in these spaces, and then use these models to plan in the latent space. A primary challenge in planning with learned dynamics models arises from their inaccuracy – thus, it is vital to have some measure of uncertainty or model accuracy estimate in order for the planner to be reliable. In this spirit, *Guzzi et al.* (2020) estimates the confidence that a controller can move between states to guide planning. *McConachie et al.* (2020); *Mitrano et al.* (2021) works in a similar setting, learning when a reduced-order dynamics model can be used reliably. *Chua et al.* (2018) uses ensembles of dynamics models to estimate the uncertainty in the dynamics for planning, while *Deisenroth and Rasmussen* (2011) uses Gaussian processes to quantify this uncertainty. While these methods can work well, they can be heuristic and lack guarantees on safety or reachability – designing a planning algorithm with learned models that offers such (probabilistic) guarantees is the focus of Chapters

VIII-IX.

Another class of methods use Gaussian Processes (GPs) to estimate the mean and covariance of the dynamics, providing probabilistic bounds on safety and reachability when learning the dynamics or a policy. For example, *Koller et al.* (2018) probabilistically bounds the reachable set of a fixed horizon trajectory. Similarly *Akametalu et al.* (2014) explores the environment while ensuring (with some probability) safety via HJ reachability analysis. In many contexts, GPs are used to derive confidence bounds that can provide probabilistic safety guarantees *Berkenkamp et al.* (2016, 2017); *Schreiter et al.* (2015); *Turchetta et al.* (2016). While these approaches provide stronger guarantees than the approaches discussed previously, they still suffer from some drawbacks. For instance, the GP-based methods struggle with long-horizon planning due to the unbounded growth of the covariance ellipse unless a known feedback controller exists *Koller et al.* (2018). Another primary challenge arises from the fact that the uncertainty bounds used by these methods are only valid if the true underyling function conforms with the assumptions made by the Gaussian process – e.g., it is compatible with the kernel used. Oftentimes, this is not the case (e.g., if the dynamics are not smooth, or if the learned GP hyperparameters overfit to the data and do not extrapolate well elsewhere). The goal of the methods presented in Chapters VIII-IX is to instead provide model error bounds which are estimated directly from the data, instead of imposing a GP-based prior on the errors we may see. Our methods also design the controller, and do not require that it is known. However, a key limitation of our approach is that does not currently handle stochastic dynamics, due to limitations in the representation of our model error bound, while GP-based approaches handle this naturally.

A different set of safe reinforcement learning algorithms is based on constrained Markov decision processes *Altman* (1999). These methods *Chow et al.* (2018, 2019) attempt to find policies that optimize a primary reward function while ensuring that some auxiliary objectives do not fall below a known threshold, in expectation. However, these approaches require an initial safe policy (which is unlikely to be known in the settings that we consider); moreover, while the algorithms ensure safety during training in theory, constraint violation still occurs in practice when deploying the policy, due to errors induced by function approximation *Chow et al.* (2019). Our work in Chapters VIII-IX consider a related, but distinct, setting, where we are provided a large dataset collected offline to train a dynamics model for model-based planning, instead of the known initial safe policy. In doing so, we are able to provide probabilistic guarantees on safety without requiring an *a priori* known safe policy.

Our work is also related to methods that learn stability certificates from data. By stability certificates, we refer to the different means by which a controller can be certified to be stabilizing, e.g., a Lyapunov function which ensures the closed-loop system is stable, a barrier function which can override some performance controller within a domain that is certified to be invariant, or a contraction metric which ensures that system trajectories converge to each other within some domain. Many methods learn stability certificates for a single equilbirium point *Boffi et al.* (2020); *Manek and Kolter* (2019); *Richards et al.* (2018), but this is insufficient for point-to-point motion planning, which is the focus of Chapters VIII and IX – this would necessitate learning

a large number of Lyapunov functions for each trajectory that needs to be tracked; in contrast, in our work, we will rely on controllers based on contraction theory, which are more directly suited for trajectory tracking. A different approach *Fan et al.* (2020) directly learns tracking tubes around trajectories using deep quantile regression, but it is unclear how close plans must stay to the training data for the guarantees to hold. Other approaches marry deep learning-based representations with contraction theory *Sun et al.* (2020), *Tsukamoto and Chung* (2021), modeling the contraction metric as an neural network and learning it from data, assuming that the dynamics are known but possibly subjected to a disturbance of known uniform upper bound. Our method differs by learning the dynamics and CCM together to optimize planning performance under model error. Also related to these contraction-based techniques is *Singh et al.* (2020), which learns a dynamics model jointly with a control contraction metric in order to promote stabilizability of the learned model, but does not consider how model error affects tracking. In Chapters VIII-IX, we will learn a feedback controller which is contracting with respect to the standard Euclidean metric (or a learned Riemannian metric – a control contraction metric), while also estimating what disturbance will be applied to the learned system as a result of model error.

Finally, a key tool that we will use in Chapters VIII - IX is the concept of using an estimated Lipschitz constant together with data density within a domain to bound the model error within that domain. Prior work has used data coverage and Lipschitz constant regularization to ensure properties of a learned function. *Dean et al.* (2020a) shows that a linearization of a nonlinear state estimator generalizes with bounded error by estimating the maximum slope (related to the Lipschitz constant) of the error. *Robey et al.* (2020) learns a control barrier function (CBF) from data, ensuring its validity through Lipschitz constant regularity and by checking the CBF conditions at a finite set of points. In contrast, we apply and extend these ideas to plan with learned dynamics, using the Lipschitz constant of the error dynamics to provide safety, reachability, and stability-like guarantees (Chapters VIII-IX).

2.3 Perception-based control

Perception-based control, or the problem of solving control tasks from observations (such as images), is a final body of work relevant to the contributions of this thesis. In particular, our work in Chapter X aims to safely integrate motion planning with knowledge of the capabilities of the upstream learned perception module, as well as the downstream trajectory tracking controller, in order to design plans which are guaranteed to remain safe and robustly reach the goal despite errors arising from imperfections in the dynamics model and perception system.

Perception-based control has a long history in both the control and estimation theory communities as output-feedback control *Åström and Murray* (2004), recent years have given rise to interest in perception-based control from the robotics and machine learning communities. Recent techniques, for the most part, revolve around applying a wide variety of learning-based techniques for solving perception-based control problems. In the following, we will overview the literature in this area, as well

as our contributions relative to this existing work.

Output-feedback control has its beginnings in the 1960s with estimation for state space systems from observations (outputs), starting with notions of observability *Kalman* (1960) and the development of the first state observers (e.g., the Luenberger observer) *Luenberger* (1971). Around the same time, it was determined that for linear systems, a "separation principle" *Joseph and Tou* (1961); *Potter* (1964) exists for output-feedback control, namely, that synthesizing an optimal output-feedback controller is equivalent to synthesizing an optimal observer, and feeding the output of this observer into a separately synthesized optimal state feedback controller. This strong separation principle can simplify the synthesis of output-feedback controllers; however, it does not hold in general for nonlinear systems, which limits its utility for the nonlinear robotic systems of interest in this thesis. There is more recent work on output-feedback control for nonlinear systems, for instance see the survey in *Findeisen et al.* (2003). The most relevant body of work in output-feedback control is the subset of work based on ideas from contraction theory. Akin to the guarantees for state-feedback control discussed previously, contraction theory can also be used to provide convergence guarantees for state estimation without control input, e.g., *Bonnabel and Slotine* (2015); *Dani et al.* (2015); *Tsukamoto and Chung* (2021). These contraction-based estimators can be used in interconnection with contraction-based controllers to be used in the output-feedback setting. However, these existing approaches generally assume that the system dynamics and observation map are known, and that the observations are low-dimensional; these are all assumptions that are difficult to justify in unstructured robotics applications. In Chapter X, we build upon ideas from the output-feedback contraction-based control method in *Manchester and Slotine* (2014), and develop an algorithm for probabilistically-safe output-feedback motion planning from high-dimensional image observations. To achieve this, our work seeks to reconstruct the full state from the observations; however, full reconstruction may not be necessary in general to stabilize the system *Sadraddini and Tedrake* (2020); *van Willigenburg and Koning* (1999). Building upon reduced order estimators for control may be a fruitful future direction for improving robustness when the observations may, for instance, be subjected to occlusions that can be challenging for controllers requiring full state reconstruction.

Our work also more generally relates to planning under uncertainty from visual input. Funnel-based methods have also been shown to scale to visual control, by buffering motion primitives with tracking tubes under vision-based *Veer and Majumdar* (2020) feedback control. In contrast, we do not rely on precomputed primitives, and can plan novel trajectories. Other methods *Agha-mohammadi et al.* (2014); *Bahreinian et al.* (2021); *van den Berg et al.* (2011) consider measurement error in planning but are either restricted to linear systems or simple sensor models. These methods are instances of the generally intractable belief-space planning problem. The belief-space planning problem can be posed as a POMDP, and solving this problem requires simplifying assumptions *Kurniawati et al.* (2008); *Sunberg and Kochenderfer* (2018), or the involvement of black-box learning components *Deglurkar et al.* (2021); *Igl et al.* (2018); *Karkus et al.* (2017) that may compromise safety and robustness when the policies are deployed online. We do not solve the full belief-space planning

problem; instead of tracking belief distributions, our set-based approach in Chapter X bounds the reachable states and state estimates under the worst-case error.

Our work is also related to recent advances in perception-based control from the machine learning community. Differentiable filtering *Haarnoja et al.* (2016); *Jonschkowski et al.* (2018); *Karkus et al.* (2018); *Kloss et al.* (2021) learns state estimators from images in an end-to-end fashion. Latent space planning *Banijamali et al.* (2018); *Hafner et al.* (2019); *Watter et al.* (2015); *Yan et al.* (2020); *Zhang et al.* (2019) seek to learn mappings (encoders) from the observation space to a lower-dimensional latent space in which motion planning and control are performed. Both of these bodies of work, which while empirically successful, do not provide guarantees on safety or goal reachability; our contribution in Chapter X seeks to make progress towards closing this gap.

There is also recent work on safe control from high-dimensional observations coming from the control community. For instance, *Dean et al.* (2020a) safely controls linear systems using learned observation maps; other methods use Control Barrier/Lyapunov Functions (CBF/CLFs) to guarantee safety for nonlinear systems by robustifying the CBF condition to measurement errors *Cosner et al.* (2021); *Dean et al.* (2020b); however, these methods use simple sensor models or require that the entire state is invertible from one observation, precluding their use on states that must be estimated over time, e.g., velocities. In contrast, our method in Chapter X only seeks to invert a *subset* of the state, which is then used in a dynamic observer to estimate the unobserved states. Other work *Dawson et al.* (2022) combines CLFs and CBFs to safely reach goals from observations, but focuses on simpler LiDAR sensor models.

Finally, certified perception is a recent relevant body of work. Representative work in this area, e.g., *Yang and Carlone* (2022); *Yang et al.* (2021), aims to certify when a solution to a geometric estimation problem is optimal; other work seeks to improve the robustness *Li et al.* (2020), generalization *Ren et al.* (2020), and out-of-distribution detection *Sharma et al.* (2021) of learning-based perception modules. However, such techniques have yet to be integrated with downstream planners and controllers. In Chapter X, we do not employ these certified algorithms within our perception module for simplicity, but an interesting direction for future work lies in determining how these certifiable algorithms for perception can be used to improve robustness and certification for end-to-end perception-based controllers.

CHAPTER III

Learning Constraints from Globally-Optimal Demonstrations

In this chapter, we extend the learning from demonstration paradigm by providing a method for learning unknown constraints shared across tasks, using demonstrations of the tasks, their cost functions, and knowledge of the system dynamics and control constraints. Given safe demonstrations, our method uses hit-and-run sampling to obtain lower cost, and thus unsafe, trajectories. Both safe and unsafe trajectories are used to obtain a consistent representation of the unsafe set via solving an integer program. Our method generalizes across system dynamics and learns a guaranteed subset of the constraint. Additionally, by leveraging a known parameterization of the constraint, we modify our method to learn parametric constraints in high dimensions. We also provide theoretical analysis on what subset of the constraint and safe set can be learnable from safe demonstrations. We demonstrate our method on linear and nonlinear system dynamics, show that it can be modified to work with suboptimal demonstrations, and that it can also be used to learn constraints in a feature space. This chapter is based off the papers *Chou et al.* (2018a, 2019, 2021a).

3.1 Introduction

Inverse optimal control and inverse reinforcement learning (IOC/IRL) (*Abbeel and Ng* (2004); *Argall et al.* (2009); *Ng and Russell* (2000); *Ratliff et al.* (2006)) have proven to be powerful tools in enabling robots to perform complex goal-directed tasks. These methods learn a cost function that replicates the behavior of an expert demonstrator when optimized. However, planning for many robotics and automation tasks also requires knowing constraints, which define what states or trajectories are safe. For example, the task of safely and efficiently navigating an autonomous vehicle can naturally be described by a cost function trading off user comfort and efficiency and by the constraints of collision avoidance and executing only legal driving behaviors. In some situations, constraints can provide a more interpretable representation of a behavior than cost functions. For example, in safety critical environments, recovering a hard constraint or an explicit representation of an unsafe set in the environment is more useful than learning a "softened" cost function representation of the constraint

as a penalty term in the Lagrangian. Consider the autonomous vehicle, which absolutely must avoid collision, not simply give collisions a cost penalty. Furthermore, learning global constraints shared across many tasks can be useful for generalization. Again consider the autonomous vehicle, which must avoid the scene of a car accident: this is a shared constraint that holds regardless of the task it is trying to complete.

While constraints are important, it can be impractical for a user to exhaustively program into a robot all the possible constraints that it should obey when performing its repertoire of tasks. To avoid this, we consider in this chapter the problem of recovering the latent constraints within expert demonstrations that are shared across tasks in the environment. Our method is based on the key insight that each safe, optimal demonstration induces a set of lower-cost trajectories that must be unsafe due to violation of an unknown constraint. Our method samples these unsafe trajectories, ensuring they are also consistent with the known constraints (system dynamics, control constraints, and start/goal constraints), and uses these unsafe trajectories together with the safe demonstrations as constraints in an "inverse" integer program which recovers a consistent unsafe set. Our contributions are fivefold:

- We pose the novel problem of learning a shared constraint across tasks.
- We propose an algorithm that, given known constraints and boundedly suboptimal demonstrations of state-control sequences, extracts unknown constraints defined in a wide range of constraint spaces (not limited to the trajectory or state spaces) shared across demonstrations of different tasks.
- We propose a variant of the aforementioned algorithm which can scale more gracefully to constraints in high dimensions by assuming and leveraging parametric structure in the constraint.
- We provide theoretical analysis on the limits of what subsets of a constraint can be learned, depending on the demonstrations, the system dynamics, and the trajectory discretization. We also prove that our method can recover guaranteed inner approximations of both the unsafe set and the safe set.
- We provide experiments that justify our theory and show that our algorithm can recover an unsafe set with few demonstrations, across different types of linear and nonlinear dynamics, and can be adapted to work with boundedly suboptimal demonstrations. We also demonstrate that our method can learn constraints in high-dimensional state spaces and parameter spaces.

3.2 Preliminaries and Problem Statement

The goal of this work is to recover unknown constraints shared across a collection of optimization problems, given boundedly suboptimal solutions, the cost functions, and knowledge of the dynamics, control constraints, and start/goal constraints. We discuss the forward problem, which generates the demonstrations, and the inverse problem: the core of this work, which recovers the constraints.

3.2.1 Forward optimal control problem

Consider an agent described by a state in some state space $x \in \mathcal{X}$. It can take control actions $u \in \mathcal{U}$ to change its state. The agent performs tasks Π drawn from a set of tasks $\mathcal{P}$, where each task Π can be written as a constrained optimization problem over state trajectories ξ_x in state trajectory space $\mathcal{T}^x$ and control trajectories ξ_u in control trajectory space $\mathcal{T}^u$:

Problem III.1 (Forward problem / "task" Π)**.**

$$\begin{array}{ll} \underset{\xi_x,\xi_u}{\text{minimize}} & c_\Pi(\xi_x,\xi_u) \\ \text{s.t.} & \phi(\xi_x,\xi_u) \in \mathcal{S} \subseteq \mathcal{C} \\ & \bar{\phi}(\xi_x,\xi_u) \in \bar{\mathcal{S}} \subseteq \bar{\mathcal{C}} \\ & \phi_\Pi(\xi_x,\xi_u) \in \mathcal{S}_\Pi \subseteq \mathcal{C}_\Pi \end{array} \tag{3.1}$$

where $c_\Pi(\cdot) : \mathcal{T}^x \times \mathcal{T}^u \to \mathbb{R}$ is a cost function for task Π and $\phi(\cdot,\cdot) : \mathcal{T}^x \times \mathcal{T}^u \to \mathcal{C}$ is a known feature function mapping state-control trajectories to some constraint space $\mathcal{C}$, elements of which are referred to as "constraint states". Mappings $\bar{\phi}(\cdot,\cdot) : \mathcal{T}^x \times \mathcal{T}^u \to \bar{\mathcal{C}}$ and $\phi_\Pi(\cdot,\cdot) : \mathcal{T}^x \times \mathcal{T}^u \to \mathcal{C}_\Pi$ are known and map to potentially different constraint spaces $\bar{\mathcal{C}}$ and $\mathcal{C}_\Pi$, containing a known shared safe set $\bar{\mathcal{S}}$ and a known task-dependent safe set $\mathcal{S}_\Pi$, respectively. $\mathcal{S}$ is an unknown safe set, and the inverse problem aims to recover its complement, $\mathcal{A} \doteq \mathcal{S}^c$[1], the "unsafe" set. In this chapter, we focus on constraints separable in time: $\phi(\xi_x,\xi_u) \in \mathcal{A} \Leftrightarrow \exists t \in \{1,\dots,T\}\ \phi(\xi_x(t),\xi_u(t)) \in \mathcal{A}$, where we overload ϕ so it applies to the instantaneous values of the state and the input. An analogous definition holds for the continuous time case. Our method can also learn constraints which are partially or completely inseparable in time (i.e., $\phi(\xi_x,\xi_u) \in \mathcal{A} \Leftrightarrow \exists\{t_i,\dots,t_j\} \in \{1,\dots,T\}\ \phi(\xi_x(t),\xi_u(t)) \in \mathcal{A}, \forall t \in \{t_i,\dots,t_j\}$)[2].

A demonstration, $\xi_{xu} \doteq (\xi_x,\xi_u) \in \mathcal{T}^{xu}$, is a state-control trajectory which is a boundedly suboptimal solution to Problem 3.1, i.e., the demonstration satisfies all constraints and its cost is at most a factor of δ above the cost of the optimal solution ξ^*_{xu}, i.e., $c(\xi^*_x,\xi^*_u) \le c(\xi_x,\xi_u) \le (1+\delta)c(\xi^*_x,\xi^*_u)$. Furthermore, let T be a finite time horizon which is allowed to vary. If ξ_{xu} is a discrete-time trajectory ($\xi_x = \{x_1,\dots,x_T\}$, $\xi_u = \{u_1,\dots,u_T\}$), Problem III.1 is a finite-dimensional optimization problem, while Problem III.1 becomes a functional optimization problem if ξ_{xu} is a continuous-time trajectory ($\xi_x : [0,T] \to \mathcal{X}$, $\xi_u : [0,T] \to \mathcal{U}$). We emphasize that this setup does not restrict the unknown constraint to be defined on the trajectory space; it allows for constraints to be defined on any space described by the range of some known feature function ϕ.

We assume the trajectories are generated by a dynamical system $\dot{x} = f(x,u,t)$ or $x_{t+1} = f(x_t,u_t,t)$ with control constraints $u_t \in \mathcal{U}$, for all t, and that the dynamics, control constraints, and start/goal constraints are known. We further denote the set of state-control trajectories satisfying the unknown shared constraint, the known shared

[1]To be exact, the safe set is assumed to be compact for the forward problem (Problem III.1) to be well-defined, and we aim to learn the closure of the unsafe set $\mathcal{A} \doteq \mathrm{cl}(\mathcal{S}^c)$.

[2]This can be done by writing the constraints of Problem III.2 as sums over partially separable/inseparable feature components instead of completely separable components.

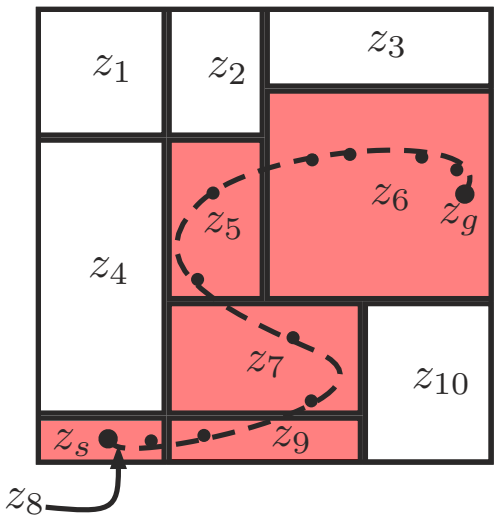


Figure 3.1: Discretized constraint space with cells $z_1, \ldots, z_{10}$. The trajectory's constraint values are assigned to the red cells.

constraint, and the known task-dependent constraint as $\mathcal{T}_{\mathcal{S}}$, $\mathcal{T}_{\bar{\mathcal{S}}}$, and $\mathcal{T}_{\mathcal{S}_\Pi}$, respectively. Lastly, we also denote the set of trajectories satisfying all known constraints but violating the unknown constraint as $\mathcal{T}_{\mathcal{A}}$.

3.2.2 Inverse constraint learning problem

The goal of the inverse constraint learning problem is to recover an unsafe set, $\mathcal{A} \subseteq \mathcal{C}$, using N_s provided safe demonstrations $\xi^*_{s_j}, j = 1, \ldots, N_s$, known constraints, and $N_{\neg s}$ inferred unsafe trajectories, $\xi_{\neg s_k}, k = 1, \ldots, N_{\neg s}$, generated by our method, which can come from multiple tasks. The safe and unsafe trajectories can together be thought of as a set of constraints on the possible assignments of unsafe constraint states in $\mathcal{C}$.

Depending on the amount of structure that we assume we know about the constraint, there are two approaches. If no structure about the constraint is known at all, the constraint space can be gridded and unsafeness can be learned on a grid-by-grid basis (Section 3.2.2.1). Otherwise, if the constraint is known to be described by some parameterization, the parameters can be learned, which leads to better scalability (Section 3.2.2.2).

Inferring unsafe trajectories, i.e., obtaining $\xi_{\neg s_k}, k = 1, \ldots, N_{\neg s}$, is the most difficult part of this problem, since finding lower-cost trajectories consistent with known constraints that complete a task is essentially a planning problem. Much of Section 3.3 shows how to efficiently obtain $\xi_{\neg s_k}$.

3.2.2.1 Recovering a gridded constraint

To recover a gridded approximation of the unsafe set $\mathcal{A}$ that is consistent with these trajectories, we first discretize $\mathcal{C}$ into a finite set of G discrete cells $\mathcal{Z} \doteq$

$\{z_1, \ldots, z_G\}$ and define an occupancy function, $\mathcal{O}(\cdot)$, which maps each cell to its safeness: $\mathcal{O}(\cdot) : \mathcal{Z} \to \{0, 1\}$, where $\mathcal{O}(z_i) = 1$ if $z_i \in \mathcal{A}$, and 0 otherwise[3]. Continuous space trajectories are gridded by concatenating the set of grid cells z_i that $\phi(x_1), \ldots, \phi(x_T)$ lie in. With slight abuse of notation, we will use $z_i \in \phi(\xi_{s_i})$ to denote $z_i \in \{\phi(\xi_{s_i}(1)), \ldots, \phi(\xi_{s_i}(T_i))$. The grid discretization is graphically shown in Figure 3.1 with a non-uniform grid. Then, the problem can be written down as an integer feasibility problem:

Problem III.2 (Inverse feasibility problem)**.**

$$\begin{aligned} \text{find} \quad & \mathcal{O}(z_1), \ldots, \mathcal{O}(z_G) \in \{0, 1\}^G \\ \text{s.t.} \quad & \sum_{z_i \in \phi(\xi^*_{s_j})} \mathcal{O}(z_i) = 0, \quad \forall j = 1, \ldots, N_s \\ & \sum_{z_i \in \phi(\xi_{\neg s_k})} \mathcal{O}(z_i) \geq 1, \quad \forall k = 1, \ldots, N_{\neg s} \end{aligned} \tag{3.2}$$

Further details on Problem III.2, including conservativeness guarantees, incorporating a prior on the constraint, and a continuous relaxation is presented in Section 3.3.4.

3.2.2.2 Recovering a parametric constraint

Suppose that the unsafe set can be described by some parameterization $\mathcal{A}(\theta) \doteq \{\kappa \in \mathcal{C} \mid g(\kappa, \theta) \leq 0\}$, where $g(\cdot, \cdot)$ is known and θ are parameters to be learned. Then, a feasibility problem analogous to Problem III.2 can be written to find a feasible θ consistent with the data:

Problem III.3 (Parametric constraint recovery problem)**.**

$$\begin{aligned} \text{find} \quad & \theta \\ \text{s.t.} \quad & g(\kappa, \theta) > 0, \quad \forall \kappa \in \phi(\xi^*_{s_i}), \quad \forall i = 1, \ldots, N_s \\ & \big(\exists \kappa \in \phi(\xi_{\neg s_j}), \quad g(\kappa, \theta) \leq 0\big), \quad \forall j = 1, \ldots, N_{\neg s} \end{aligned}$$

Further details on Problem III.3, including conservativeness guarantees and specific mixed-integer programming formulations for common constraint parameterizations are presented in Section 3.3.5.

3.3 Method

The key to our method lies in finding lower-cost trajectories that do not violate the known constraints, given a demonstration with boundedly-suboptimal cost satisfying all constraints. Such trajectories must then violate the unknown constraint, and we extend existing sampling algorithms to be more efficient for trajectory-space sampling

[3]To avoid complications when states lie on the boundary shared between two grid cells, grid cells are defined to be disjoint open sets.

Dynamics	Cost function	Control constraints	Sampling method
Linear	Quadratic	Convex	Ellipsoid hit-and-run (Section 3.3.2.1)
Linear	Convex	Convex	Convex hit-and-run (Section 3.3.2.2)
Else			Non-convex hit-and-run (Section 3.3.2.3)

Table 3.1: Sampling methods for different classes of dynamics models, cost functions, and feasible control sets.

under various assumptions on the dynamics. Our goal is to determine an unsafe set in the constraint space from these trajectories using either Problem III.2 or Problem III.3. In the following, Section 3.3.1 describes lower-cost trajectories consistent with the known constraints; Section 3.3.2 describes how to sample such trajectories; Section 3.3.3 describes how to get more information from unsafe trajectories; Section 3.3.4 describes details and extensions to Problem 3.2; Section 3.3.5 describes details for parametric constraint learning and extensions to Problem III.3; Section 3.3.6 discusses how to extend our method to suboptimal demonstrations. The complete flow of our method is described in Algorithm III.2.

3.3.1 Trajectories satisfying known constraints

Consider the forward problem (Problem 3.1). We define the set of unsafe state-control trajectories induced by an optimal, safe demonstration ξ^*_{xu}, $\mathcal{T}_{\mathcal{A}}^{\xi^*_{xu}}$, as the set of state-control trajectories of lower cost that obey the known constraints:

$$\mathcal{T}_{\mathcal{A}}^{\xi^*_{xu}} \doteq \{\xi_{xu} \in \mathcal{T}_{\bar{\mathcal{S}}} \cap \mathcal{T}_{\mathcal{S}_\Pi} \mid c(\xi_x, \xi_u) < c(\xi^*_x, \xi^*_u)\}. \tag{3.3}$$

In this chapter, we deal with the known constraints from the system dynamics, the control limits, and task-dependent start and goal state constraints. Hence, $\mathcal{T}_{\bar{\mathcal{S}}} = \mathcal{D}^{\xi_{xu}} \cap \mathcal{U}^{\xi_{xu}}$, where $\mathcal{D}^{\xi_{xu}}$ denotes the set of dynamically feasible trajectories and $\mathcal{U}^{\xi_{xu}}$ denotes the set of trajectories using controls in $\mathcal{U}$ at each time-step. $\mathcal{T}_{\mathcal{S}_\Pi}$ denotes trajectories satisfying start and goal constraints. We develop the method for discrete time trajectories, but analogous definitions hold in continuous time. For discrete time, length T trajectories, $\mathcal{U}^{\xi_{xu}}$, $\mathcal{D}^{\xi_{xu}}$, and $\mathcal{T}_{\mathcal{S}_\Pi}$ are the following subsets of $\mathcal{T}^{xu}$:

$$\begin{aligned}
\mathcal{U}^{\xi_{xu}} &\doteq \{\xi_{xu} \mid u_t \in \mathcal{U},\ \forall t \in \{1, \ldots, T-1\}\ \}, \\
\mathcal{D}^{\xi_{xu}} &\doteq \{\xi_{xu} \mid x_{t+1} = f(x_t, u_t),\ \forall t \in \{1, \ldots, T-1\}\ \}, \\
\mathcal{T}_{\mathcal{S}_\Pi} &\doteq \{\xi_{xu} \mid x_1 = x_s, x_T = x_g\}.
\end{aligned} \tag{3.4}$$

3.3.2 Sampling trajectories satisfying known constraints

We sample from $\mathcal{T}_{\mathcal{A}}^{\xi^*_{xu}}$ to obtain lower-cost trajectories obeying the known constraints using hit-and-run sampling (*Kiatsupaibul et al.* (2011)), a method guaranteeing convergence to a uniform distribution of samples over $\mathcal{T}_{\mathcal{A}}^{\xi^*_{xu}}$ in the limit; the method is detailed in Algorithm III.1 and an illustration is shown in Figure 3.2. Hit-and-run starts from an initial point within the set, chooses a direction uniformly at

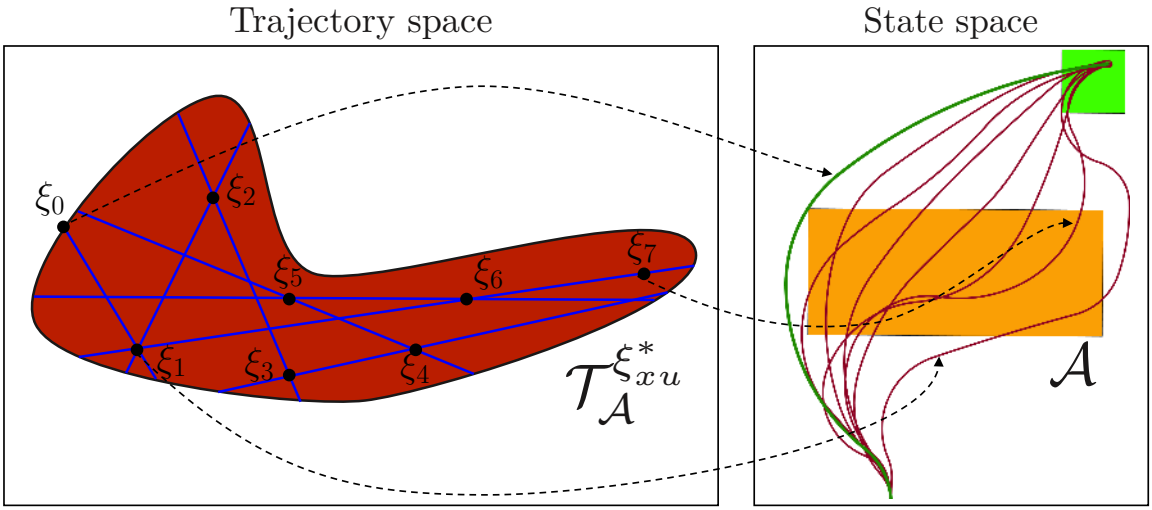


Figure 3.2: Illustration of hit-and-run. **Left**: Blue lines denote sampled random directions, black dots denote samples. **Right**: Each point in $\mathcal{T}_{\mathcal{A}}^{\xi^*_{xu}}$ corresponds to an unsafe trajectory in the constraint space $\mathcal{C}$, and in this case, $\mathcal{C} = \mathcal{X}$.

random, moves a random amount in that direction such that the new point remains within the set, and repeats.

Depending on the convexity of the cost function and the control constraints and on the form of the dynamics, different sampling techniques can be used, organized in Table 3.1. The following sections describe each sampling method.

Algorithm III.1: Hit-and-run

Output: out $\doteq \{\xi_1, \ldots, \xi_{N_{\neg s}}\}$
Input : $\mathcal{T}_{\mathcal{A}}^{\xi^*_{xu}}, \xi^*_{xu}, N_{\neg s}$
$\xi_{xu} \leftarrow \xi^*_{xu}$; out $\leftarrow \{\}$;
for $i = 1{:}N_{\neg s}$ **do**
  $r \leftarrow$ sampleRandDirection();
  $\mathcal{L} \leftarrow \mathcal{T}_{\mathcal{A}}^{\xi^*_{xu}} \cap \{\xi'_{xu} \in \mathcal{T} \mid \xi'_{xu} = \xi_{xu} + \beta r\}$;
  $L_-, L_+ \leftarrow$ endpoints($\mathcal{L}$);
  $\xi_{xu} \sim$ Uniform(L_-, L_+);
  out $\leftarrow$ out $\cup\, \xi_{xu}$;

3.3.2.1 Ellipsoid hit-and-run

When we have a linear system with quadratic cost and convex control constraints, a very common setup in the optimal control literature, the set of lower-cost trajectories satisfying the known constraints

$$\mathcal{T}_{\mathcal{A}}^{\xi^*_{xu}} \doteq \{\xi_{xu} \mid c(\xi_{xu}) < c(\xi^*_{xu})\} \cap \mathcal{D}^{\xi_{xu}} \equiv \{\xi_{xu} \mid \xi_{xu}^\top V \xi_{xu} < \xi_{xu}^{*\top} V \xi^*_{xu}\} \cap \mathcal{D}^{\xi_{xu}}$$

is an ellipsoid in the trajectory space, which can be efficiently sampled via a specially-tailored hit-and-run method. Here, the quadratic cost is written as $c(\xi_{xu}) \doteq \xi_{xu}^\top V \xi_{xu}$, where V is a matrix of cost parameters, and we omit the control and task constraints for now. Consider the intersection of the random line chosen from hit-and-run (r in Algorithm III.1) with the lower-cost trajectory set $\mathcal{T}_{\mathcal{A}}^{\xi_{xu}^*}$; denote this line segment as $\mathcal{L}$ and its endpoints as L_- and L_+ (cf. Algorithm III.1). Furthermore, denote β as a step-size in direction r; hence, L_- and L_+ put bounds on the allowable step-sizes β. Without dynamics, L_-, L_+ can be found by solving a quadratic equation $L_-^\top V L_+ = (\xi_{xu} + \beta_1 r)^\top V (\xi_{xu} + \beta_2 r) = \xi_{xu}^{*\top} V \xi_{xu}^*$. We show that this can still be done with linear dynamics by writing $\mathcal{T}_{\mathcal{A}}^{\xi_{xu}^*}$ in a special way. $\mathcal{D}^{\xi_{xu}}$ can be written as an eigenspace of a singular "dynamics consistency" matrix, D_1, which converts any arbitrary state-control trajectory to one that satisfies the dynamics, one time-step at a time. Precisely, if the dynamics can be written as $x_{t+1} = Ax_t + Bu_t$, we can write a matrix D_1:

$$\underbrace{\begin{bmatrix} x_1 \\ u_1 \\ x_2 \\ u_2 \\ \tilde{x}_3 \\ \vdots \\ u_{T-1} \\ \tilde{x}_T \end{bmatrix}}_{\hat{\xi}_{xu}} = \underbrace{\begin{bmatrix} I & 0 & 0 & 0 & 0 & \cdots & \cdots & 0 \\ 0 & I & 0 & 0 & 0 & \cdots & \cdots & 0 \\ A & B & 0 & 0 & 0 & \cdots & \cdots & 0 \\ 0 & 0 & 0 & I & 0 & \cdots & \cdots & 0 \\ 0 & 0 & A & B & 0 & \cdots & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \ddots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & \cdots & 0 & I & 0 \\ 0 & 0 & 0 & \cdots & \cdots & A & B & 0 \end{bmatrix}}_{D_1} \underbrace{\begin{bmatrix} x_1 \\ u_1 \\ \tilde{x}_2 \\ u_2 \\ \tilde{x}_3 \\ \vdots \\ u_{T-1} \\ \tilde{x}_T \end{bmatrix}}_{\xi_{xu}} \tag{3.5}$$

that fixes the controls and the initial state and performs a one-step rollout, replacing the second state with the dynamically correct state. In (3.5), we denote by $\tilde{x}_{t+1}$ a state that cannot be reached by applying control u_t to state x_t. Multiplying the one-step corrected trajectory $\hat{\xi}_{xu}$ by D_1 again changes $\tilde{x}_3$ to the dynamically reachable state x_3. Applying D_1 to the original T-time-step infeasible trajectory $T-1$ times results in a dynamically feasible trajectory, $\xi_{xu}^{\text{feas}} = D_1^{T-1} \xi_{xu}$. Further, note that the set of dynamically feasible trajectories is $\mathcal{D}^{\xi_{xu}} \doteq \{\xi_{xu} \mid D_1 \xi_{xu} = \xi_{xu}\}$, which is the span of the eigenvectors of D_1 associated with eigenvalue 1. Thus, obtaining a feasible trajectory via repeated multiplication is akin to finding the eigenspace via power iteration (*Golub and Van Loan* (1996)). One can also interpret this as propagating through the dynamics with a fixed control sequence. Now, we can write $\mathcal{T}_{\mathcal{A}}^{\xi_{xu}^*}$ as another ellipsoid:

$$\mathcal{T}_{\mathcal{A}}^{\xi_{xu}^*} \doteq \{\xi_{xu} \mid \xi_{xu}^\top {D_1^{T-1}}^\top V D_1^{T-1} \xi_{xu} \leq \xi_{xu}^{*\top} V \xi_{xu}^*\}. \tag{3.6}$$

Like for the kinematic case, this ellipsoid can be efficiently sampled after finding L_-, L_+ by solving a quadratic equation $(\xi_{xu} + \beta_1 r)^\top {D_1^{T-1}}^\top V D_1^{T-1} (\xi_{xu} + \beta_2 r) = \xi_{xu}^{*\top} V \xi_{xu}^*$.

We deal with control constraints separately, as the intersection of $\mathcal{U}^{\xi_{xu}}$ and (3.6)

is in general not an ellipsoid. To ensure control constraint satisfaction, we reject samples with controls outside of $\mathcal{U}^{\xi_{xu}}$; this works if $\mathcal{U}^{\xi_{xu}}$ is not measure zero. For task constraints, we ensure all sampled rollouts obey the goal constraints by adding a large penalty term to the cost function: $\tilde{c}(\cdot) \doteq c(\cdot) + \alpha_c \|x_g - x_T\|_2^2$, where α_c is a large scalar, which can be incorporated into (3.6) by modifying V and including x_g in ξ_{xu}; all trajectories sampled in this modified set satisfy the goal constraints to an arbitrarily small tolerance ε, depending on the value of α_c. The start constraint is satisfied trivially: all rollouts start at x_s. Note the demonstration cost remains the same, since the demonstration satisfies the start and goal constraints; this modification is made purely to ensure these constraints hold for sampled trajectories.

3.3.2.2 Convex hit-and-run

For general convex cost functions, the same sampling method holds, but L_+, L_- cannot be found by solving a quadratic function. Instead, we solve $c(\xi_{xu} + \beta r) = c(\xi_{xu}^*)$ via a root finding algorithm or line search.

3.3.2.3 Non-convex hit-and-run

If $\mathcal{T}_{\mathcal{A}}^{\xi_{xu}^*}$ is non-convex, $\mathcal{L}$ can now in general be a union of disjoint line segments. In this scenario, we perform a "backtracking" line search by setting β to lie in some initial range: $\beta \in [\underline{\beta}, \overline{\beta}]$; sampling β_s within this range and then evaluating the cost function to see whether or not $\xi_{xu} + \beta_s r$ lies within the intersection. If it does, the sample is kept and hit-and-run proceeds normally; if not, then the range of possible β values is restricted to $[\beta_s, \overline{\beta}]$ if β_s is negative, and $[\underline{\beta}, \beta_s]$ otherwise. Then, new βs are re-sampled until either the interval length shrinks below a threshold or a feasible sample is found. This altered hit-and-run technique still converges to a uniform distribution on the set in the limit, but has a slower mixing time than for the convex case, where mixing time describes the number of samples needed until the total variation distance to the steady state distribution is less than a small threshold (*Abbasi-Yadkori et al.* (2017)). Further, we accelerate sampling spread by relaxing the goal constraint to a larger tolerance $\hat{\varepsilon} > \varepsilon$ but keeping only the trajectories reaching within ε of the goal.

3.3.3 Improving learnability using cost function structure

Naïvely, the sampled unsafe trajectories may provide little information. Consider an unsafe, length-T discrete-time trajectory ξ, with start and end states in the safe set. This only says there exists at least one intermediate unsafe state in the trajectory, but says nothing directly about which state was unsafe. The weakness of this information can be made concrete using the notion of a version space. In machine learning, the version space is the set of consistent hypotheses given a set of examples (*Russell and Norvig* (2003)). In our setting, hypotheses are possible unsafe sets, and examples are the safe and unsafe trajectories. Knowing ξ is unsafe only disallows unsafe sets that mark every constraint state in the constraint space that ξ traverses as safe: $(\mathcal{O}(z_2) = 0) \wedge \ldots \wedge (\mathcal{O}(z_{T-1}) = 0)$. If $\mathcal{C}$ is gridded into G cells, this information

Algorithm III.2: Overall method

Output: $\mathcal{O} \doteq \mathcal{O}(z_1), \ldots, \mathcal{O}(z_G)$ (Problems III.2, III.4, III.5)
θ (Problems III.3, III.7)
Input : $\xi_s = \{\xi_1^*, \ldots, \xi_{N_s}^*\}$, $c_\Pi(\cdot)$, known constraints}

$\xi_\mathrm{u} \leftarrow \{\}$;
for *i = 1:N_s* **do**
  `/* Sample unsafe` ξ `*/`
  if *lin., quad., conv.* **then**
    $\xi_\mathrm{u} \leftarrow \xi_\mathrm{u} \cap \mathsf{ellipsoidHNR}(\xi_i^*)$;
  else if *lin., conv., conv.* **then**
    $\xi_\mathrm{u} \leftarrow \xi_\mathrm{u} \cap \mathsf{convexHNR}(\xi_i^*)$;
  else
    $\xi_\mathrm{u} \leftarrow \xi_\mathrm{u} \cap \mathsf{nonconvexHNR}(\xi_i^*)$;
`/* Constraint recovery */`
if *gridded* **then**
  $\mathcal{O} \leftarrow$ Problem X(ξ_s, ξ_u);
  `/* X = Problem III.5 if prior, continuous */`
  `/* X = Problem III.4 if prior, binary */`
  `/* X = Problem III.2 if no prior */`
else if *parameterization* **then**
  $\theta \leftarrow$ Problem Y(ξ_s, ξ_u);
  `/* Y = Problem III.7 if polytope param. */`
  `/* Y = Problem III.3 if general param. */`

invalidates at most 2^{G-T+2} out of 2^G possible unsafe sets. We could do exponentially better if we reduced the number of cells that ξ implies could be unsafe.

We can achieve this by sampling sub-segments (or sub-trajectories) of the larger demonstrations, holding other portions of the demonstration fixed. For example, say we fix all but one of the points on ξ when sampling unsafe lower-cost trajectories. Since only one state can be different from the known safe demonstration, the unsafeness of the trajectory can be uniquely localized to whatever new point was sampled: then, this trajectory will reduce the version space by at most a factor of 2, invalidating at most $2^G - 2^{G-1} = 2^{G-1}$ unsafe sets. One can sample these sub-trajectories in the full-length trajectory space by fixing appropriate waypoints during sampling: this ensures the full trajectory has lower cost and only perturbs desired waypoints. However, to speed up sampling, sub-trajectories can be sampled directly in the lower dimensional sub-trajectory space if the cost function $c(\cdot)$ that is being optimized is strictly monotone (*Morin* (1982)): for any costs $c_1, c_2 \in \mathbb{R}$, control $u \in \mathcal{U}$, and state $x \in \mathcal{X}$, $c_1 < c_2 \Rightarrow h(c_1, x, u) < h(c_2, x, u)$, for all x, u, where $h(c, x, u)$ represents the cost of starting with initial cost c at state x and taking control u. Strictly monotone cost functions include separable cost functions with additive or multiplicative stage costs, which are common in motion planning and optimal control. If the cost function is strictly monotone, we can sample lower-cost trajectories from sub-segments of the optimal path; otherwise it is possible that even if a new sub-segment with lower cost than the original sub-segment were sampled, the full trajectory containing the sub-segment could have a higher cost than the demonstration.

3.3.4 Gridded integer program formulation

As mentioned in Sections 3.2.2.1 and 3.2.2.2, we can solve various optimization problems after sampling to find an unsafe set consistent with the safe and unsafe trajectories. We now discuss the details of this process, starting with the gridded formulation (Problem III.2).

3.3.4.1 Conservative estimate

One can obtain a conservative estimate of the unsafe set $\mathcal{A}$ from Problem III.2 by intersecting all possible solutions: if the unsafeness of a cell is shared across all feasible solutions, that cell must be occupied. In practice, it may be difficult to directly find all solutions to the feasibility problem, as in the worst case, finding the set of all feasible solutions is equivalent to exhaustive search in the full gridded space (*Papadimitriou and Steiglitz* (1982)). A more efficient method is to loop over all G grid cells $z_1, \ldots, z_G$ and set each one to be safe, and see if the optimizer can still find a feasible solution. Cells where there exists no feasible solution are guaranteed unsafe. This amounts to solving G binary integer feasibility problems, which can be trivially parallelized. Furthermore, any cells that are known safe (from demonstrations) do not need to be checked. We use this method to compute the "*learned guaranteed*

unsafe cells", $\mathcal{G}^z_{\neg s}$, in Section 3.5, which we define as:

$$\mathcal{G}^z_{\neg s} = \{z \in \{z_1, \ldots, z_G\} \mid \mathcal{O}(z) = 1,\ \forall \mathcal{O} \in \mathcal{F}^z\} \tag{3.7}$$

where $\mathcal{F}^z$ is the feasible set of Problem III.2.

3.3.4.2 A prior on the constraint

As will be further discussed in Section 3.4.1, it may be fundamentally impossible to recover a unique unsafe set. If we have some prior on the nature of the unsafe set, such as it being simply connected, or that certain regions of the constraint space are unlikely to be unsafe, we can make the constraint learning problem more well-posed. Assume that this prior knowledge can be encoded in some "energy" function $E(\cdot, \ldots, \cdot) : \{0,1\}^G \to \mathbb{R}$ mapping the set of binary occupancies to a scalar value, which indicates the desirability of a particular unsafe set configuration. Using E as the objective function in Problem III.2 results in a binary integer program, which finds an unsafe set consistent with the safe and unsafe trajectories, and minimizes the energy:

Problem III.4 (Inverse binary minimization constraint recovery)**.**

$$\begin{aligned} \underset{\substack{\mathcal{O}(z_1),\ldots,\mathcal{O}(z_G)\\ \in\{0,1\}^G}}{\text{minimize}} \quad & E(\mathcal{O}(z_1), \ldots, \mathcal{O}(z_G)) \\ \text{s.t.} \quad & \sum_{z_i \in \phi(\xi^*_{s_j})} \mathcal{O}(z_i) = 0, \quad \forall j = 1, \ldots, N_s \\ & \sum_{z_i \in \phi(\xi_{\neg s_k})} \mathcal{O}(z_i) \geq 1, \quad \forall k = 1, \ldots, N_{\neg s} \end{aligned} \tag{3.8}$$

3.3.4.3 Probabilistic setting and continuous relaxation

A similar problem can be posed in a probabilistic setting, where grid cell occupancies represent beliefs over unsafeness: instead of the occupancy of a cell being an indicator variable, it is instead a random variable Z_i, where Z_i takes value 1 with probability $\tilde{\mathcal{O}}(Z_i)$ and value 0 with probability $1 - \tilde{\mathcal{O}}(Z_i)$. Here, the occupancy probability function maps cells to occupancy probabilities $\tilde{\mathcal{O}}(\cdot) : \mathcal{Z} \to [0, 1]$.

Trajectories can now be unsafe with some probability. We obtain analogous constraints from the integer program in Section 3.3.4 in the probabilistic setting. Known safe trajectories traverse cells that are unsafe with probability 0; we enforce this with the constraint $\sum_{Z_i \in \phi(\xi^*_{s_j})} \tilde{\mathcal{O}}(Z_i) = 0$: if the unsafeness probabilities are all zero along a trajectory, then the trajectory must be safe. Trajectories that are unsafe with probability p_k satisfy $\sum_{Z_i \in \phi(\xi_{\neg s_k})} \tilde{\mathcal{O}}(Z_i) = \mathbb{E}[\sum_{Z_i \in \phi(\xi_{\neg s_k})} Z_i] = (1 - p_k) \cdot 0 + p_k \cdot S_k \geq p_k$ where we denote the number of unsafe grid cells $\phi(\xi_{\neg s_k})$ traverses when the trajectory is unsafe as S_k, where $S_k \geq 1$. The following problem directly optimizes over occupancy probabilities:

Problem III.5 (Inverse continuous minimization constraint recovery)**.**

$$\begin{aligned} \underset{\substack{\mathcal{O}(Z_1),\ldots,\mathcal{O}(Z_G)\\ \in[0,1]^G}}{\text{minimize}} \quad & E(\mathcal{O}(Z_1),\ldots,\mathcal{O}(Z_G)) \\ \text{s.t.} \quad & \sum_{z_i\in\phi(\xi^*_{s_j})} \tilde{\mathcal{O}}(Z_i) = 0, \quad \forall j = 1,\ldots,N_s \\ & \sum_{z_i\in\phi(\xi_{\neg s_k})} \tilde{\mathcal{O}}(Z_i) \geq p_k, \quad \forall k = 1,\ldots,N_{\neg s} \end{aligned} \tag{3.9}$$

When $p_k = 1$, for all k (i.e., all unsafe trajectories are unsafe for sure), this probabilistic formulation coincides with the continuous relaxation of Problem III.4. This justifies interpreting the solution of the continuous relaxation as occupancy probabilities for each cell. Note that Problem III.4 and III.5 have no conservativeness guarantees and use prior assumptions to make the problem more well-posed. However, we observe that they improve constraint recovery in our experiments.

3.3.5 Parameter space integer program

Having discussed extensions to the gridded constraint recovery problem, we now turn to analogous results for the parametric case.

3.3.5.1 Conservative estimate

Denote by $\mathcal{F}$ the feasible set of Problem III.3. Denote by $\mathcal{G}_{\neg s}$ and $\mathcal{G}_s$ the set of constraint states which are learned guaranteed unsafe and safe, respectively; that is, a constraint state $\kappa \in \mathcal{G}_{\neg s}$ or $\kappa \in \mathcal{G}_s$ if κ is classified unsafe or safe for all $\theta \in \mathcal{F}$:

$$\mathcal{G}_{\neg s} \doteq \bigcap_{\theta\in\mathcal{F}} \{\kappa \mid g(\kappa,\theta) \leq 0\} \tag{3.10}$$

$$\mathcal{G}_s \doteq \bigcap_{\theta\in\mathcal{F}} \{\kappa \mid g(\kappa,\theta) > 0\} \tag{3.11}$$

In contrast to $\mathcal{G}^z_{\neg s}$, which is the set of guaranteed learned unsafe grid cells (the analogue of $\mathcal{G}_{\neg s}$ for grid cells), $\mathcal{G}_s$ and $\mathcal{G}_{\neg s}$ are defined directly over the constraint space $\mathcal{C}$.

Note that unlike Problem III.2, for Problem III.3, it is possible to learn that a constraint state not lying on a demonstration is guaranteed safe. This is due to the parameterization: given a particular set of safe and unsafe trajectories, there may not be any feasible parameter $\theta \in \mathcal{F}$ where κ is classified unsafe. For example, consider the case in Figure 3.3: given the interval parameterization $g(\kappa, \theta = [\bar{\kappa}, \underline{\kappa}]) = (\bar{\kappa}-\kappa)(\underline{\kappa}-\kappa)$, it is not possible for any constraint state left of κ_1 or right of κ_2 to be classified unsafe and be consistent with the data. On the other hand, due to the independence of the grids in Problem III.2, learning that a given grid cell is safe or unsafe cannot ever imply that another grid cell is guaranteed safe.

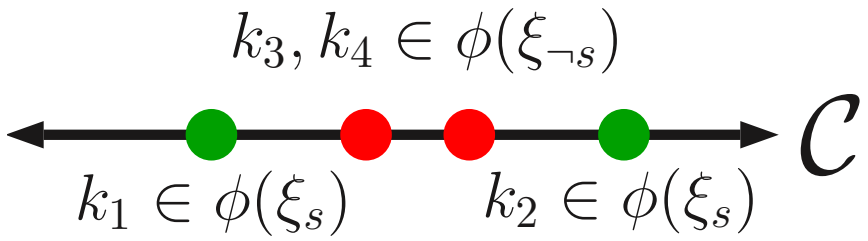


Figure 3.3: Given an interval parameterization of an unsafe set, there does not exist any interval which can both explain the data and label and constraint state left of κ_1 or right of κ_2 as unsafe.

Similarly to Problem III.2, one can check if a constraint state $\kappa \in \mathcal{G}_s$ or $\kappa \in \mathcal{G}_{\neg s}$ by adding a constraint $g(\kappa, \theta) \leq 0$ or $g(\kappa, \theta) > 0$ to Problem III.3 and checking feasibility of the resulting program; if the program is infeasible, $\kappa \in \mathcal{G}_s$ or $\kappa \in \mathcal{G}_{\neg s}$. In other words, solving this modified integer program can be seen as querying an oracle about the safety of a constraint state κ. The oracle can then return that κ is guaranteed safe (program infeasible after forcing κ to be unsafe), guaranteed unsafe (program infeasible after forcing κ to be safe), or unsure (program remains feasible despite forcing κ to be safe or unsafe).

Since Problem III.3 works in the continuous constraint space, it is not possible to exhaustively check if each constraint state is guaranteed learned safe or unsafe, unlike the discrete gridded case in Problem III.2. For general parameterizations, only individual states can be checked for membership in $\mathcal{G}_s$ or $\mathcal{G}_{\neg s}$. However, for some particularly common parameterizations, there are more efficient methods for recovering subsets of $\mathcal{G}_s$ and $\mathcal{G}_{\neg s}$:

- Axis-aligned hyper-rectangle parameterization: $\mathcal{C} \subseteq \mathbb{R}^n$, $\theta = [\underline{\kappa}_1, \bar{\kappa}_1, \ldots, \underline{\kappa}_n, \bar{\kappa}_n]$, $g(\kappa, \theta) \leq 0 \Leftrightarrow H(\theta) \leq h(\theta)$, where $H(\theta)k = [I_{n\times n}, -I_{n\times n}]^\top$ and $h(\theta) = [\bar{k}_1, \ldots, \bar{k}_n, \underline{k}_1, \ldots, \underline{k}_n]^\top$. Here, $\underline{\kappa}_i$ and $\bar{\kappa}_i$ are the lower and upper bounds of the hyper-rectangle for coordinate i.

  As the set of axis-aligned hyper-rectangles is closed under intersection, $\mathcal{G}_{\neg s}$ is also an axis-aligned hyper-rectangle, the axis-aligned bounding box of any two constraint states $\kappa_1, \kappa_2 \in \mathcal{G}_{\neg s}$ is also contained in $\mathcal{G}_{\neg s}$. This also implies that $\mathcal{G}_{\neg s}$ can be fully described by finding the top and bottom corners $[\underline{\kappa}_1, \ldots, \underline{\kappa}_n]^\top$ and $[\bar{\kappa}_1, \ldots, \bar{\kappa}_n]^\top$. Suppose we start with a known $\kappa \in \mathcal{G}_{\neg s}$. Then, finding $[\underline{\kappa}_1, \ldots, \underline{\kappa}_n]^\top$ amounts to performing a binary search for each of the n dimensions, and the same holds for finding $[\bar{\kappa}_1, \ldots, \bar{\kappa}_n]^\top$.

  Recovering $\mathcal{G}_s$ is not as straightforward, as the complement of axis-aligned boxes is not closed under intersection. However, an inner approximation of $\mathcal{G}_s$ can be efficiently obtained as follows: starting at a constraint state $\kappa \in \mathcal{G}_{\neg s}$, $2n$ line searches can be performed to find the two points of transition to $\mathcal{G}_{\neg s}$ in each constraint coordinate. Denote as $\hat{\mathcal{G}}_s$ the complement of the axis-aligned bounding box of these $2n$ points; $\hat{\mathcal{G}}_s$ is an inner approximation of $\mathcal{G}_s$, as $\mathcal{G}_s = (\bigcap_{\theta\in\mathcal{F}}\{x \mid g(x, \theta) \leq 0\})^c \supseteq \text{AABB}(\bigcap_{\theta\in\mathcal{F}}\{x \mid g(x, \theta) \leq 0\})^c$, where AABB$(\cdot)$

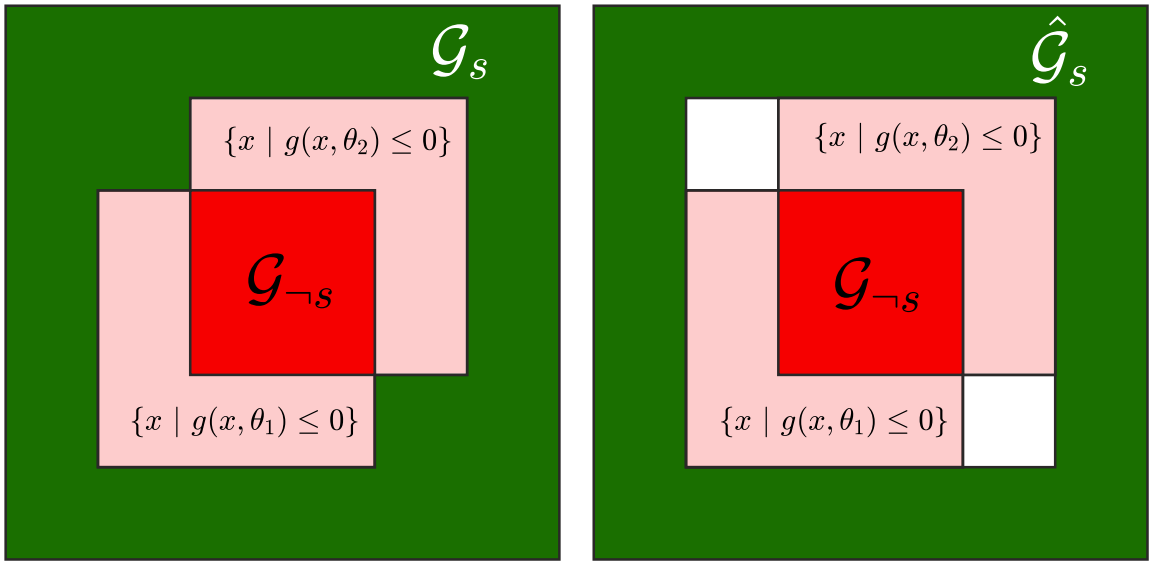


Figure 3.4: Comparison of the true $\mathcal{G}_s$ (left, in green) and the extracted inner approximation $\hat{\mathcal{G}}_s$ (right, in green).

denotes the axis-aligned bounding box of a set of points and the complement acts on the axis-aligned bounding box.

For example, consider the scenario in Figure 3.4 where there are only two feasible parameters, θ_1 and θ_2. Here, $\mathcal{G}_s$ is $(\mathcal{A}(\theta_1) \cup \mathcal{A}(\theta_2))^c$ and $\hat{\mathcal{G}}_s$ under-approximates the safe set ($\mathcal{G}_s$ is in general not representable as the complement of an axis-aligned box).

- Convex parameterization: for fixed θ, $\{\kappa \mid g(\kappa,\theta) \leq 0\}$ is convex.

  While in this case, it is not easy to recover $\mathcal{G}_{\neg s}$ exactly, a subset of $\mathcal{G}_{\neg s}$ can be extracted efficiently by taking the convex hull of any $\kappa_1, \kappa_2, \ldots \in \mathcal{G}_{\neg s}$, since the convex hull is the minimal convex set containing $\kappa_1, \kappa_2, \ldots$.

An alternative solution seeks to check if a set of states in the neighborhood of some constraint state κ_{query} is contained within $\mathcal{G}_{\neg s}$; this can be done by solving the following problem:

Problem III.6 (Volume extraction)**.**

$$\begin{aligned}
\min_{\theta,\varepsilon} \quad & \varepsilon \\
\text{s.t.} \quad & g(\kappa_i,\theta) > 0, \quad \forall \kappa_i \in \phi(\xi^*_{s_j}), \quad \forall j = 1,\ldots,N_s \\
& \exists \kappa_i \in \phi(\xi_{\neg s_k}), \quad g(\kappa_i,\theta) \leq 0, \quad \forall k = 1,\ldots,N_{\neg s} \\
& \exists \kappa_{near} \in \{\kappa_{near} \mid \|\kappa_{near} - \kappa_{query}\|_\infty \leq \varepsilon\}, \quad g(\kappa_{near},\theta) > 0
\end{aligned}$$

In words, Problem III.6 finds the smallest ε-hypercube centered at κ_{query} containing a $\kappa \notin \mathcal{G}_{\neg s}$; thus, any hypercube of size $\hat{\varepsilon} < \varepsilon$ is contained within $\mathcal{G}_{\neg s}$: $\{\kappa \mid \|\kappa - \kappa_{\text{query}}\|_\infty \leq \hat{\epsilon}\} \subseteq \mathcal{G}_{\neg s}$. We can write a similar problem to check the neighborhood of κ_{query} for membership in $\mathcal{G}_s$. Volumes of safe/unsafe space can thus be

produced by repeatedly solving Problem III.6 for different κ_{query}, and these volumes can be passed to a planner to generate new trajectories that are guaranteed safe.

The same approaches apply for recovering $\mathcal{G}_s$ when it is instead the safe set which is an axis-aligned hyper-rectangle or a convex set.

3.3.5.2 Choice of parameterization

In this section, we identify classes of parameterizations for which Problem III.3 can be efficiently solved:

- $g(\kappa, \theta)$ is defined by a Boolean conjunction of linear inequalities, i.e., $\mathcal{A}(\theta)$ can be defined as the union and intersection of half-spaces:

  For this case, mixed-integer programming can be employed. As an example for the particular case where $g(\kappa, \theta) \leq 0$ is a single polytope, i.e., $g(\kappa, \theta) \Leftrightarrow H(\theta)k \leq h(\theta)$, where $H(\theta)$ and $h(\theta)$ are affine in θ, the following mixed integer feasibility problem can be solved to find a feasible θ:

  Problem III.7 (Polytopic constraint recovery problem)**.**

  $$\begin{aligned}
  \text{find} \quad & \theta, \{b_s^i\}_{i=1}^{N_s}, \{b_{\neg s}^i\}_{i=1}^{N_{\neg s}} \\
  \text{s.t.} \quad & H(\theta)\kappa_i > h(\theta) - M(1 - b_s^i), \quad b_{s_j}^i \in \{0,1\}^{N_h}, \\
  & \sum_{i=1}^{N_h} b_{s_j}^i \geq 1, \forall \kappa_i \in \phi(\xi_{s_j}), i = 1, \ldots, T_j, j = 1, \ldots, N_s \qquad (3.12a) \\
  & H(\theta)\kappa_i \leq h(\theta) + M(1 - b_{\neg s_k}^i)\mathbf{1}_{N_h}, \quad b_{\neg s_k}^i \in \{0,1\}, \\
  & \sum_{i=1}^{T_j} b_{\neg s_k}^i \geq 1, \quad \forall \kappa_i \in \phi(\xi_{\neg s_k}), \quad \forall k = 1, \ldots, N_{\neg s} \qquad (3.12b)
  \end{aligned}$$

  where M is a large positive number and $\mathbf{1}_{N_h}$ is a column vector of ones of length N_h. Constraints (3.12a) and (3.12b) use the big-M formulation to enforce that each safe constraint state lies outside $\mathcal{A}(\theta)$ and that at least one constraint state on each unsafe trajectory lies inside $\mathcal{A}(\theta)$.

  Similar problems can be written down when the safe or unsafe set can be described by unions of polytopes.

  As an alternative to mixed integer programming, satisfiability modulo theories (SMT) solvers can also be employed to solve Problem III.3 if $g(\kappa, \theta)$ is defined by a Boolean conjunction of linear inequalities.

- $g(\kappa, \theta)$ is defined by a Boolean conjunction of convex inequalities, i.e., $\mathcal{A}(\theta)$ can be described as the union and intersection of convex sets:

  For this case, satisfiability modulo convex optimization (SMC) (*Shoukry et al.* (2018)) can be employed to find a feasible θ.

3.3.5.3 Unknown parameterizations

For many realistic applications, we do not have access to a known parameterization which can represent the unsafe set. Despite this, complex unsafe/safe sets can often be approximated as the union of many simple unsafe/safe sets. Along this line of thought, we present a method for incrementally growing a parameterization based on the complexity of the demonstrations and unsafe trajectories.

Suppose that the true parameterization $g(\kappa, \theta)$ of the unsafe set $\mathcal{A}(\theta) = \{\kappa \mid g(\kappa, \theta) \leq 0\}$ is unknown but can be exactly or approximately expressed as the union of N^* simple sets $\mathcal{A}(\theta) \cong \bigcup_{i=1}^{N^*} \{\kappa \mid g_s(\kappa, \theta_i) \leq 0\} \doteq \bigcup_{i=1}^{N^*} \mathcal{A}(\theta_i)$, where each simple set $\mathcal{A}(\theta_i)$ has a known parameterization $g_s(\cdot, \cdot)$ and N^*, the minimum number of simple sets needed to reconstruct $\mathcal{A}$, is unknown.

A lower bound on N^*, $\underline{N}$, can be estimated by incrementally adding simple sets until Problem III.3 becomes feasible. However, for $\underline{N} < N^*$, the extracted $\mathcal{G}_s$ and $\mathcal{G}_{\neg s}$ are not guaranteed to be conservative estimates of $\mathcal{S}$ and $\mathcal{A}$ (Theorem III.24), and $\mathcal{G}_s$ and $\mathcal{G}_{\neg s}$ are only guaranteed to be conservative if $\hat{N} \geq N^*$, where $\hat{N}$ is the chosen number of simple sets (see Theorem III.23). Unfortunately, inferring a guaranteed overestimation of N^* only from data is not possible, as there can always be subsets of the constraint which are not activated by the given demonstrations. Two facts mitigate this:

- If an upper bound on the number of simple sets needed to describe $\mathcal{A}(\theta)$, $\bar{N}_{\text{loose}} \geq N^*$, is known (where this bound can be trivially loose), $\mathcal{G}_s \subseteq \mathcal{S}$ and $\mathcal{G}_{\neg s} \subseteq \mathcal{A}$ by using $\bar{N}_{\text{loose}}$ simple sets in solving Problem III.3. Hence, by using $\bar{N}_{\text{loose}}$, $\mathcal{G}_s$ and $\mathcal{G}_{\neg s}$ can be made guaranteed conservative (see Theorem III.23), at the cost of the resulting $\mathcal{G}_s$ and $\mathcal{G}_{\neg s}$ being potentially small.

- As the demonstrations begin to cover the space, $\underline{N} \to N^*$. Hence, by using $\underline{N}$ simple sets, $\mathcal{G}_s$ and $\mathcal{G}_{\neg s}$ are asymptotically conservative.

In our experiments, we choose our simple sets as axis-aligned hyper-rectangles in $\mathcal{C}$, which is motivated by: 1) any open set in $\mathcal{C}$ can be approximated as a countable/finite union of open axis-aligned hyper-rectangles *Tao* (2016); 2) unions of hyper-rectangles are easily representable in Problem III.7.

3.3.5.4 Remarks on parameter space problem

We close this subsection with some remarks on implementation and extensions to Problem III.3.

- For suboptimal demonstrations or imperfect lower-cost trajectory sampling, Problem III.7 can become infeasible. To address this, slack variables can be introduced: replace constraint $\sum_{i=1}^{T_j} b_{\neg s}^i \geq s_k, s_k \in \{0, 1\}$ and change the feasibility problem to minimization of $\sum_{k=1}^{N_{\neg s}} (1 - s_k)$.

- In addition to recovering sets of guaranteed learned unsafe and safe constraint states, a probability distribution over possibly unsafe constraint states can be estimated by sampling unsafe sets from the feasible set of Problem III.3.

3.3.6 Bounded suboptimality of demonstrations

If we are given a δ-suboptimal demonstration $\hat{\xi}$, where $c(\xi^*) \leq c(\hat{\xi}) \leq (1+\delta)c(\xi^*)$, where ξ^* is an optimal demonstration, we can still apply the sampling techniques discussed in earlier sections, but we must ensure that sampled unsafe trajectories are truly unsafe: a sampled trajectory ξ' of cost $c(\xi') \geq c(\xi^*)$ can be potentially safe. Two options follow: one is to only keep trajectories with cost less than $\frac{c(\hat{\xi})}{1+\delta}$, but this can cause little to be learned if δ is large. Instead, if we assume a distribution on suboptimality, i.e., given a trajectory of cost $c(\hat{\xi})$, we know that a trajectory of cost $c(\xi') \in [\frac{c(\hat{\xi})}{1+\delta}, c(\hat{\xi})]$ is unsafe with probability p_k, we can then use these values of p_k to solve Problem III.5.

3.4 Analysis

In this section, we provide theoretical analysis on our constraint learning algorithm. In particular, we analyze the limits of what constraint states can be learned guaranteed unsafe for both the gridded and parametric cases (Sections 3.4.1 and 3.4.3) as well as the conditions under which our algorithm is guaranteed to learn an inner approximation of the safe and unsafe sets (Sections 3.4.2 and 3.4.4). For ease of reading, the proofs and some remarks are omitted and can be found in the appendix.

We begin with an overview of the theoretical results:

- Theorem III.9 shows that all states that can be guaranteed unsafe must lie within some distance to the boundary of the unsafe set. Corollary III.10 shows that the set of guaranteed unsafe states shrinks to a subset of the boundary of the unsafe set when using a continuous demonstration directly to learn the constraint.

- Corollary III.15 shows that under assumptions on the alignment of the grid and unsafe set for the discrete time case, the guaranteed learned unsafe set is a guaranteed inner approximation of the true unsafe set.

- For continuous trajectories that are then discretized, Theorem III.16 shows us that the guaranteed unsafe set can be made to contain states on the interior of the unsafe set, but at the cost of potentially labeling states within some distance outside of the unsafe set as unsafe as well.

- Theorem III.19 shows that for the parametric case, all states that can be guaranteed unsafe must be implied unsafe by the states within some distance to the boundary of the unsafe set and the parameterization.

- Theorem III.21 shows that for the discrete time case, the guaranteed safe and guaranteed unsafe sets are inner approximations of the true safe and unsafe sets, respectively. For the continuous time case, the recovered sets are inner approximations of a padded version of the true sets.

- Theorems III.23 and III.24 present conservativeness results when the constraint parameterization is not exactly known.

3.4.1 Learnability

We provide analysis on the learnability of unsafe sets, given the known constraints and cost function. Most analysis assumes unsafe sets defined over the state space: $\mathcal{A} \subseteq \mathcal{X}$, but we extend it to the feature space in Corollary III.17. We provide some definitions and state a result bounding ${\mathcal{G}^z_{\neg s}}^*$, the set of all states that **can** be learned guaranteed unsafe. We first define the signed distance:

Definition III.8 (Signed distance)**.** Signed distance from point $p \in \mathbb{R}^m$ to set $\mathcal{S} \subseteq \mathbb{R}^m$, $\mathsf{sd}(p, \mathcal{S}) = -\inf_{y \in \partial \mathcal{S}} \|p - y\|$ if $p \in \mathcal{S}$; $\inf_{y \in \partial \mathcal{S}} \|p - y\|$ if $p \in \mathcal{S}^c$.

Theorem III.9 (Learnability (discrete time))**.** *For trajectories generated by a discrete time dynamical system satisfying $\|x_{t+1} - x_t\| \leq \Delta x$ for all t, the set of learnable guaranteed unsafe states is a subset of the outermost Δx shell of the unsafe set: ${\mathcal{G}^z_{\neg s}}^* \subseteq \{x \in \mathcal{A} \mid -\Delta x \leq \mathsf{sd}(x, \mathcal{A}) \leq 0\}$ (see Section A.1.1, Figure* A.1 *for an illustration).*

Corollary III.10 (Learnability (continuous time))**.** *For continuous trajectories $\xi(\cdot) : [0, T] \to \mathcal{X}$, the set of learnable guaranteed unsafe states shrinks to the boundary of the unsafe set: ${\mathcal{G}^z_{\neg s}}^* \subseteq \{x \in \mathcal{A} \mid \mathsf{sd}(x, \mathcal{A}) = 0\}$.*

Depending on the cost function, ${\mathcal{G}^z_{\neg s}}^*$ can become arbitrarily small: some cost functions are not very informative for recovering a constraint. For example, the path length cost function used in many of the experiments (which was chosen due to its common use in the motion planning community), prevents any lower-cost sub-trajectories from being sampled from straight sub-trajectories. The overall control authority that we have on the system also impacts learnability: the more controllable the system, the more of the Δx shell is reachable. In particular, a necessary condition for any unsafe states to be learnable from a demonstration of length $T + 1$ starting from x_0 and ending at x_T is for there to be more than one trajectory which steers from x_0 to x_T in $T + 1$ steps while satisfying the dynamics and control constraints.

3.4.2 Conservativeness

We discuss conditions on $\mathcal{A}$ and discretization which ensure our method provides a conservative estimate of $\mathcal{A}$. For analysis, we assume $\mathcal{A}$ has a Lipschitz boundary (*Dacorogna* (2015)). We begin with notation (an explanatory illustration is in Section A.1.2, Figure A.2):

Definition III.11 (Normal vectors)**.** Denote the outward-pointing normal vector at a point $p \in \partial\mathcal{A}$ as $\hat{n}(p)$. Furthermore, at non-differentiable points on $\partial\mathcal{A}$, $\hat{n}(p)$ is replaced by the set of normal vectors for the sub-gradient of the Lipschitz function describing $\partial\mathcal{A}$ at that point (*Allaire et al.* (2016)).

Definition III.12 (γ-offset padding)**.** Define the γ-offset padding $\partial\mathcal{A}_\gamma$ as: $\partial\mathcal{A}_\gamma = \{x \in \mathcal{X} \mid x = y + d\hat{n}(y), d \in [0, \gamma], y \in \partial\mathcal{A}\}$.

Definition III.13 (γ-padded set)**.** We define the γ-padded set of the unsafe set $\mathcal{A}$, $\mathcal{A}(\gamma)$, as the union of the γ-offset padding and $\mathcal{A}$: $\mathcal{A}(\gamma) \doteq \partial\mathcal{A}_\gamma \cup \mathcal{A}$.

Definition III.14 (Maximum grid size)**.** Let $R(z_i)$ be the radius of the smallest ball which contains grid cell z_i: $R(z_i) = \min_r \min_{x_i} r$, subject to $z_i \subseteq B_r(x_i)$, for some optimal center x_i.

Furthermore, let R^* be the radius of the smallest ball which contains each grid cell $z_i, i = 1, \ldots, G$: $R^* = \max(R(z_1), \ldots, R(z_G))$.

We also introduce the following assumption, which is illustrated in Figure A.3 for clarity:
Assumption 1: The unsafe set $\mathcal{A}$ is aligned with the grid (i.e., there does not exist a grid cell z containing both safe and unsafe states in its interior).

Theorem III.15 (Discrete time conservative recovery of unsafe set)**.** *For a discrete-time system, if Assumption 1 holds, $\mathcal{G}^z_{\neg s} \subseteq \mathcal{A}$. If Assumption 1 does not hold, then $\mathcal{G}^z_{\neg s} \subseteq \mathcal{A}(R^*)$.*

If we use continuous trajectories directly, the guaranteed learnable set $\mathcal{G}^z_{\neg s}{}^*$ shrinks to a subset of the boundary of the unsafe set, $\partial\mathcal{A}$ (cf. Corollary III.10). However, if we discretize these trajectories, we show that we can learn unsafe states lying in the interior, at the cost of conservativeness holding only for a padded unsafe set. We then show that a similar result holds when discretizing a continuous trajectory in a feature space. For the following results, we make an additional assumption, which is illustrated in Figure A.4 for clarity:
Assumption 2: The time discretization of the unsafe trajectory $\xi : [0, T] \to \mathcal{X}$, $\{t_1, \ldots, t_N\}, t_i \in [0, T]$, for all i, is chosen such that there exists at least one discretization point in the interior of each cell that the continuous trajectory passes through (i.e., if $\exists t \in [0, T]$ such that $\xi(t) \in z$, then $\exists t_i \in \{t_1, \ldots, t_N\}$ such that $\xi(t_i) \in z$.

Theorem III.16 (Continuous-to-discrete time conservativeness)**.** *The following results hold for continuous time systems:*

1. *Suppose that both Assumptions 1 and 2 hold. Then, the learned guaranteed unsafe set $\mathcal{G}^z_{\neg s}$, defined in Section 3.3.4.1, is contained within the true unsafe set $\mathcal{A}$.*

2. *Suppose that only Assumption 2 holds. Then, the learned guaranteed unsafe set $\mathcal{G}^z_{\neg s}$ is contained within the R^*-padded unsafe set, $\mathcal{A}(R^*)$.*

3. *Suppose that neither Assumption 1 nor Assumption 2 holds. Furthermore, suppose that Problems III.2, III.4, and III.5 are using M sub-trajectories sampled with Algorithm III.1 as unsafe trajectories, and that each sub-trajectory is defined over the time interval $[a_i, b_i], i = 1, \ldots, M$. Denote $D_\xi([a, b]) \doteq$*

$\sup_{t_1\in[a,b],t_2\in[t_1,b]} \|\xi(t_1)-\xi(t_2)\|_2$, *for some trajectory* ξ. *Denote* $D^* \doteq \max_{i\in\{1,\ldots,M\}} D^*_{\xi_i}([a_i,b_i])$. *Then, the learned guaranteed unsafe set* $\mathcal{G}^z_{\neg s}$ *is contained within the* $D^* + R^*$*-padded unsafe set,* $\mathcal{A}(D^* + R^*)$.

Corollary III.17 (Continuous-to-discrete feature space conservativeness)**.** *Let the feature mapping* $\phi(x)$ *from the state space to the constraint space be Lipschitz continuous with Lipschitz constant* L. *Then, the following results hold:*

1. *Suppose both Assumptions 1 and 2 (used in Theorem III.16) hold. Then, our method ensures* $\mathcal{G}^z_{\neg s} \subseteq \mathcal{A}$.
2. *Suppose only Assumption 2 holds. Then, our method recovers a guaranteed subset of the* LR^**-padded unsafe set,* $\mathcal{A}(LR^*)$*, in the feature space.*
3. *Suppose neither Assumption 1 nor Assumption 2 holds. Then, our method recovers a guaranteed subset of the* $L(D^* + R^*)$*-padded unsafe set,* $\mathcal{A}(L(D^* + R^*))$*, where* D^* *is as defined in Theorem III.16.*

3.4.3 Parametric learnability

In this section, we develop results for learnability of the unsafe set in the parametric case. For clarity, we prove the results for $\mathcal{C} = \mathcal{X}$. We begin with the following notation:

Definition III.18 (Implied unsafe set)**.** For some set $\mathcal{B} \subseteq \Theta$, denote

$$I(\mathcal{B}) \doteq \bigcap_{\theta\in\mathcal{B}} \{x \mid g(x,\theta) \leq 0\} \tag{3.13}$$

as the set of states that are implied unsafe by restricting the parameter set to $\mathcal{B}$. In words, $I(\mathcal{B})$ is the set of states for which all $\theta \in \mathcal{B}$ mark as unsafe.

The following result states that in discrete time, the learnable set of unsafe states $\mathcal{G}^*_{\neg s}$ is contained by the set of states which must be implied unsafe by learning that all states in the outer Δx shell of the unsafe set, $\mathcal{A}_{\Delta x}$, are unsafe. Furthermore, in continuous time, the same holds, except the Δx shell is replaced by the boundary of the unsafe set, $\partial\mathcal{A}$.

Theorem III.19 (Discrete time learnability for parametric constraints)**.** *For trajectories generated by discrete time systems,* $\mathcal{G}_{\neg s} \subseteq \mathcal{G}^*_{\neg s} \subseteq I(\mathcal{F}_{\Delta x})$*, where*

$$\mathcal{F}_{\Delta x} = \{\theta \mid \forall i \in \{1,\ldots,N_s\}, \forall x \in \xi^*_i, g(x,\theta) > 0, \\ \forall x \in \mathcal{A}_{\Delta x}, g(x,\theta) \leq 0\}$$

Corollary III.20 (Continuous-time learnability for parametric constraints)**.** *For trajectories generated by continuous time systems,* $\mathcal{G}_{\neg s} \subseteq \mathcal{G}^*_{\neg s} \subseteq I(\mathcal{F}_{\partial\mathcal{A}})$*, where*

$$\mathcal{F}_{\partial\mathcal{A}} = \{\theta \mid \forall x \in \xi^*_i, \forall i \in \{1,\ldots,N_s\}, g(x,\theta) > 0, \\ \forall x \in \partial\mathcal{A}, g(x,\theta) \leq 0\}$$

3.4.4 Parametric conservativeness

We write conditions for conservative recovery of the unsafe set and safe set when solving Problems III.3 and III.7 for discrete time and continuous time systems. In the following two results, we assume that the constraint parameterization is known.

Theorem III.21 (Conservative recovery for discrete time systems with parametric constraints)**.** *For a discrete-time system, if M in Problem III.7 is chosen to be greater than* $\max(M_1, M_2)$*, where* $M_1 = \max_{x_i \in \xi_s} \max_\theta \max_j (H(\theta)x_i - h(\theta))_j$ *and* $M_2 = \max_{x_i \in \xi_{\neg s}} \max_\theta \max_j (H(\theta)x_i - h(\theta))_j$*,* $\mathcal{G}_{\neg s} \subseteq \mathcal{A}$ *and* $\mathcal{G}_s \subseteq \mathcal{S}$*.*

Corollary III.22 (Conservative recovery for continuous time systems with parametric constraints)**.** *For a continuous-time system, where demonstrations are time-discretized as previously discussed, if M is chosen as in Theorem III.21,* $\mathcal{G}_s \subseteq \mathcal{S}$ *and* $\mathcal{G}_{\neg s} \subseteq \mathcal{A}(D^*)$*, where D^* is as defined in Theorem III.16.*

Now, let's consider the case where the true parameterization is not known and we use the incremental method described in Section 3.3.5.3, where $g_s(x, \theta)$ is the simple parameterization. We consider the over-parameterized case (Theorem III.23) and the under-parameterized case (Theorem III.24). We analyze the case where the true, under-, and over-parameterization are defined respectively as:

$$g(x, \theta) \leq 0 \Leftrightarrow \bigvee_{i=1}^{N^*} \big(g_s(x, \theta_i) \leq 0\big) \tag{3.14}$$

$$g(x, \theta) \leq 0 \Leftrightarrow \bigvee_{i=1}^{\underline{N}} \big(g_s(x, \theta_i) \leq 0\big), \quad \underline{N} < N^* \tag{3.15}$$

$$g(x, \theta) \leq 0 \Leftrightarrow \bigvee_{i=1}^{\bar{N}} \big(g_s(x, \theta_i) \leq 0\big), \quad \bar{N} > N^*. \tag{3.16}$$

Theorem III.23 (Conservativeness: Over-parameterization)**.** *Suppose the true parameterization and over-parameterization are defined as in* (3.14) *and* (3.16)*. Then,* $\mathcal{G}_{\neg s} \subseteq \mathcal{A}$ *and* $\mathcal{G}_s \subseteq \mathcal{S}$*.*

Theorem III.24 (Conservativeness: Under-parameterization)**.** *Suppose the true parameterization and under-parameterization are defined as in* (3.14) *and* (3.15)*. Furthermore, assume that we incrementally grow the parameterization as described in Section 3.3.5.3. Then, the following are true:*

1. *$\mathcal{G}_{\neg s}$ and $\mathcal{G}_s$ are not guaranteed to be contained in $\mathcal{A}$ (unsafe set) and $\mathcal{S}$ (safe set), respectively.*
2. *Each recovered simple unsafe set $\mathcal{A}(\theta_i)$, $i = 1, \ldots, \underline{N}$, for any $\theta_1, \ldots, \theta_{\underline{N}} \in \mathcal{F}$, touches the true unsafe set (there are no spurious simple unsafe sets): for $i = 1, \ldots, \underline{N}$, for $\theta_1, \ldots, \theta_{\underline{N}} \in \mathcal{F}$, $\mathcal{A}(\theta_i) \cap \mathcal{A} \neq \emptyset$ ($\underline{N}$ is as defined in Section 3.3.5.3).*

	Fig. 3.8, Row 1	Fig. 3.8, Row 2	Fig. 3.8, Row 3	Fig. 3.9	Fig. 3.10	Fig. 3.11	Fig. 3.13R1-2	Fig. 3.13R3-4	Fig. 3.15
Discretization	0.1	0.25	0.5	1	1	n/a	n/a	n/a	n/a
No. trajs. sampled	300000	150000	10000	10000	10000	100000	250000	250000	10000
No. trajs. used	300000	150000	10000	10000	10000	125	15000	1500	500
ε	n/a	n/a	10^{-3}	10^{-3}	10^{-3}	n/a	n/a	n/a	10^{-3}
$\hat{\varepsilon}$	n/a	n/a	10^{-2}	10^{-2}	10^{-2}	n/a	n/a	n/a	10^{-2}
α_c	10^{10}	10^4	1	1	1	10^{10}	10^{10}	10^{10}	1
Min. $\mathcal{L}$ length	n/a	n/a	10^{-10}	10^{-10}	10^{-10}	n/a	n/a	n/a	10^{-10}
D^*	n/a	n/a	7.85	10.5	0.04	n/a	n/a	n/a	n/a
R^*	0.07	0.35	0.35	0.70	0.005	n/a	n/a	n/a	n/a

Table 3.2: Parameters used for each experiment.

Experiment	Time (sampling trajectories)	Time (constraint recovery)
Single integrator, U-shape, gridded; Fig. 3.8, Row 1	11.5 min	3 min
Double integrator, gridded; Fig. 3.8, Row 2	4.5 min	4.5 min
Dubins' car, gridded; Fig. 3.8, Row 3	2 hrs	4 min
Dubins' car, suboptimal, gridded; Fig. 3.9	1 hr	2 min
Dubins' car, feature space, gridded; Fig. 3.10	30 min	4 min
Single integrator, U-shape, parametric; Fig. 3.11	1.5 min	27.3 seconds
7-DOF arm; Fig. 3.11	12.5 min	1.2 seconds
7-DOF arm, suboptimal; Fig. 3.11	9 min	1.2 seconds
Quadrotor; Fig 3.15	8.5 min	11.9 seconds

Table 3.3: Approximate runtimes for each experiment.

3.5 Evaluations: Gridded formulation

In this section and the next (Section 3.6), we evaluate the effectiveness of both our gridded and parametric variants of the constraint recovery problem on a variety of examples. Experiment parameters and approximate runtimes for all examples can be found in Tables 3.2 and 3.3. All experiments were conducted on a 4-core 2017 Macbook Pro with a 3.1 GHz Core i7 processor. All code was implemented in MATLAB.

We evaluate the gridded variant of our method on a variety of constraint recovery problems in this section. In particular, we provide examples showing the effectiveness of using unsafe trajectories to reduce the ill-posedness of the constraint-recovery problem (Section 3.5.1), that our method has advantages over inverse reinforcement learning (Section 3.5.2), that our method can be applied for discrete-time, continuous-time, linear, and nonlinear system dynamics (Section 3.5.3), that our method can be adapted to work with suboptimal demonstrations (Section 3.5.4), and that our method can also learn constraints in arbitrary feature spaces (Section 3.5.5).

3.5.1 Version space example

Consider a simple 5×5 8-connected grid world in which the tasks are to go from a start to a goal, minimizing Euclidean path length while staying out of the unsafe "U-shape", the outline of which is drawn in black (Fig. 3.5). Four demonstrations are provided, shown in Fig. 3.5 on the far left. Initially, the version space contains 2^{25} possible unsafe sets. Each safe trajectory of length T reduces the version space at most by a factor of 2^T, invalidating at most $2^{25} - 2^{25-T}$ possible unsafe sets. Unsafe trajectories are computed by enumerating the set of trajectories going from the start to the goal at lower cost than the demonstration. The numbers of unsafe sets consistent with the safe and unsafe trajectories for varying numbers of safe trajectories

	1	2	3	4
Safe	262144	4096	1024	256
Safe & unsafe	11648	48	12	3

Table 3.4: Number of consistent unsafe sets, varying the number of demonstrations, using/not using unsafe trajectories (cf. the example in Section 3.5.1).

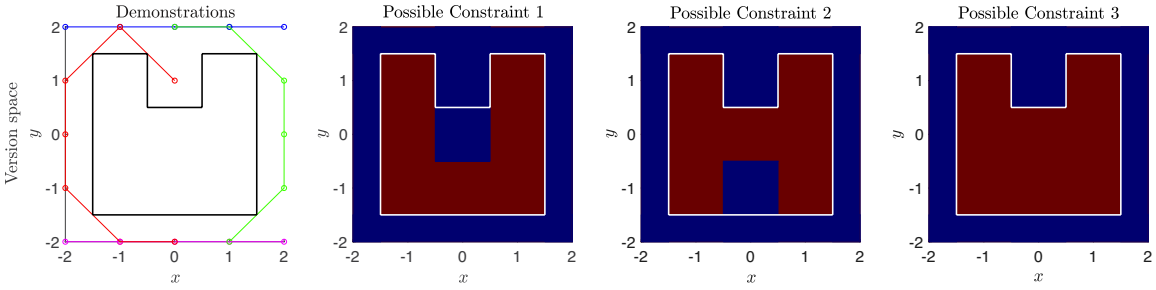


Figure 3.5: **Leftmost**: Demonstrations and unsafe set. **Rest**: Set of possible constraints. Postulated unsafe cells are plotted in red, safe states in blue.

are given in Table 3.4.

Ultimately, it is impossible to distinguish between the three unsafe sets on the right in Fig. 3.5. This is because there exists no task where a trajectory with cost lower than the demonstration can be sampled which only goes through one of the two uncertain states. Further, though the uncertain states are in the Δx shell of the constraint, due to the limitations of the cost function, we can only learn a subset of that shell (cf. Theorem III.9).

There are two main takeaways from this experiment. First, by generating unsafe trajectories, we can reduce the uncertainty arising from the ill-posedness of constraint learning: after 4 demonstrations, using unsafe demonstrations enables us to reduce the number of possible constraints by nearly a factor of 100, from 256 to 3. Second, due to limitations in the cost function, it may be impossible to recover a unique unsafe set, but the version space can be reduced substantially by sampling unsafe trajectories.

3.5.2 Comparison with inverse reinforcement learning

In this section, we illustrate some advantages of explicitly learning a hard constraint from demonstrations over learning a softened penalty through two examples.

3.5.2.1 Gridded example

Consider the grid world in Figure 3.6(a), where the available actions at each state are to move up, down, left, right (except when doing so goes out of bounds), and an "exit" action, which takes the agent to a terminal state. In this setting, the objective of the demonstrator is to minimize the path length to the goal (green square) while avoiding the unsafe set (red squares), and we are given one demonstration doing so

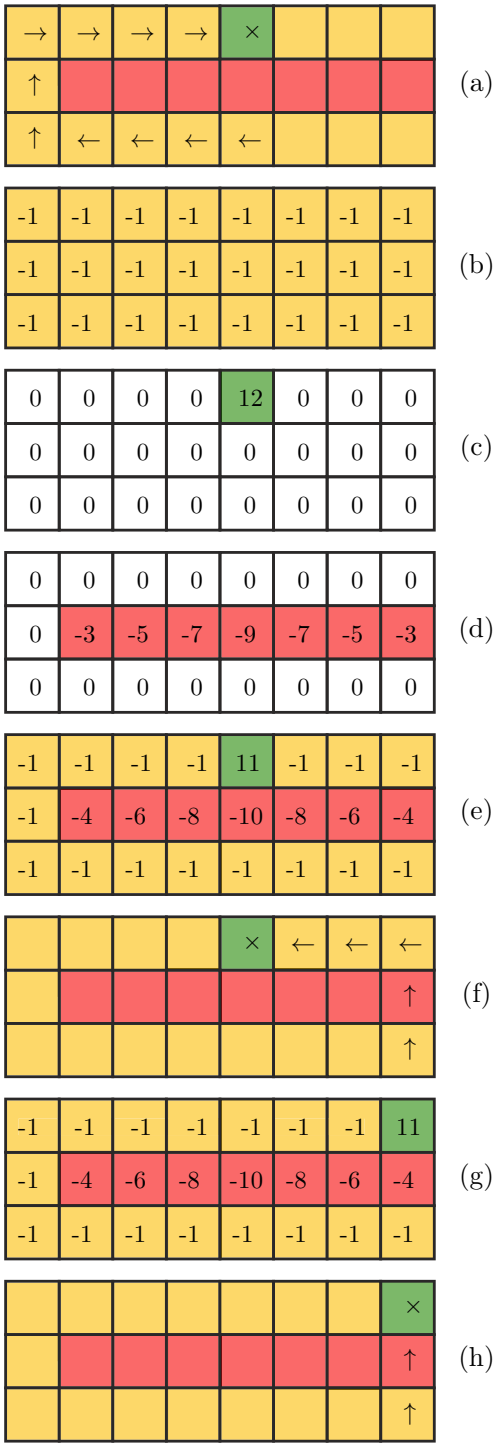


Figure 3.6: IRL comparison (gridded). (a) Demonstration. (b) Path length component of reward function $r_{\text{path}}(x)$ (numbers indicate the reward obtained upon reaching a state). (c) Goal component of reward function $r_{\text{goal}}(x)$. (d) A consistent softened constraint reward function. (e) Combined reward function. (f) Unsafe optimal trajectory from a new initial condition under the combined reward function. (g) Combined reward function for a different goal. (h) Unsafe optimal trajectory when planning with a different goal.

(see Figure 3.6(a)).

Suppose that as the learner, we know the objective is path length (see Figure 3.6(b)) and we also know the goal (see Figure 3.6(c)), and we would like to learn the unknown red constraint by representing it as an unknown penalty component in the reward function and learning it via inverse reinforcement learning (*Ng and Russell* (2000)). We can write this problem in the framework of IRL by representing the grid world as a deterministic, finite horizon Markov decision process $\langle S, A, P, R, T \rangle$, where the state space S, action space A, and transition probabilities P are as described in the previous paragraph, and T is the time horizon ($T = 11$ for this problem). The reward function $R(s, a)$ is assumed to have a known component $R_{\text{goal}}(s, a) + R_{\text{path}}(s, a)$ and unknown component $R_{\text{unsafe}}(s, a)$, which is to be learned from the demonstrations. Specifically, the path length aspect of the cost function is modeled by a small negative reward upon reaching each state (see Figure 3.6(b)), $R_{\text{goal}}(s, a)$ is zero except for a large positive reward obtained by taking the exit action at the goal state (see Figure 3.6(c)), and we take the reward function to be of the form $R(s, a) = R_{\text{goal}}(s, a) + R_{\text{path}}(s, a) + R_{\text{unsafe}}(s, a)$. One penalty function $R_{\text{unsafe}}(s, a)$ which is consistent with the demonstration and the known reward function component is shown in Figure 3.6(d) (here, the numerical labels on each state correspond to the reward obtained when taking an action that reaches that state). This is because by using this penalty, there exists no trajectory that achieves a larger cumulative reward than the demonstration, under the combined reward function (Figure 3.6(e)).

However, using this learned penalty when starting from the bottom right state leads to an optimal path which is unsafe (Figure 3.6(f)), as the learned penalty is only consistent with the observed demonstrations but does not necessarily adequately enforce the constraint starting from novel initial states. On the other hand, using our method, by sampling lower-cost trajectories, we can learn that each state on the middle row except for the leftmost state is guaranteed unsafe. Using this learned constraint and planning a path starting from the bottom right state leads to a path which avoids the unsafe set. Similarly, the learned penalty will not necessarily be valid when changing the known component of the reward function (i.e., the goal state) because the learned penalty values will depend on the values of the known component, while the learned constraint is agnostic to the known component and will transfer across different known reward functions, and unsafe paths can be planned using the learned penalty (Figure 3.6(f)-(g)).

Overall, the key takeaways of this example are to show that representing a constraint as a reward penalty may lead to unsafe behavior when planning trajectories from new start states or to new goal states, while explicitly learning the constraint transfers more reliably across tasks.

3.5.2.2 Parametric example

Consider the problem illustrated in Figure 3.7, where the demonstrator's objective is to minimize path length while avoiding the red obstacle and satisfying input constraints: that is, the demonstrator solves Problem III.1, where the obstacle avoidance constraint is encoded in the unsafe set $\mathcal{A} = \{x \mid [I_{2\times 2}, -I_{2\times 2}]^\top x \leq \theta\}$, where $\theta =$

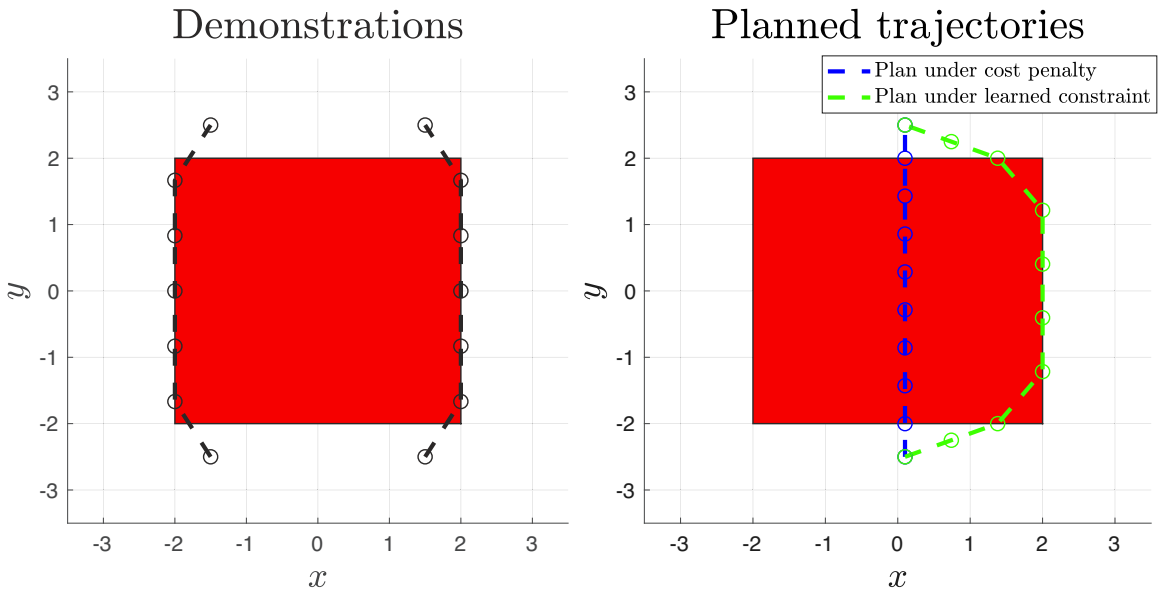


Figure 3.7: IRL comparison (parametric). **Left**: We are given two demonstrations avoiding a red obstacle. **Right**: Paths planned with the cost penalty may be unsafe, whereas trajectories planned with the learned constraint remain safe.

$[2, 2, 2, 2]^\top$ and the path length objective is encoded in $c_\Pi(\xi_x, \xi_u) = \sum_{t=1}^{T-1} \|x_{t+1} - x_t\|_2^2$. We consider two variants of the inverse optimization problem: in the first variation, we solve Problem III.7 to directly recover the parameters θ defining the unknown constraint. In the second variation, we modify Problem III.1 to soften the obstacle avoidance constraint to a cost penalty, posing the problem as:

$$\begin{aligned} \underset{\xi_x, \xi_u}{\text{minimize}} \quad & \sum_{t=1}^{T-1} \|x_{t+1} - x_t\|_2^2 + \lambda \sum_{t=1}^{T} \mathbf{1}_{x_t \notin \mathcal{S}} \\ \text{s.t.} \quad & \bar{\phi}(\xi_x, \xi_u) \in \bar{\mathcal{S}} \subseteq \bar{\mathcal{C}} \\ & \phi_\Pi(\xi_x, \xi_u) \in \mathcal{S}_\Pi \subseteq \mathcal{C}_\Pi \end{aligned} \tag{3.17}$$

where the constraints encode the input and start/goal constraints, $\mathbf{1}_{(\cdot)}$ denotes the indicator function for the event $(\cdot)$, and λ is a nonnegative penalty coefficient. In this variation of the learning problem, we assume that $\mathcal{S}$ is known, and we only aim to learn a suitable penalty coefficient λ which makes the demonstrations globally-optimal. At this point, we should also note that it can be challenging to determine a suitable cost penalty parameterization; for example, while $\lambda \sum_{t=1}^{T} \|x_t\|_\infty$ may appear to be a good penalty parameterization, we could not find a value of λ for this parameterization that replicated the demonstrated behavior presented in Figure 3.7.

By solving Problem III.7 using the two provided demonstrations and sampled lower-cost trajectories, θ can be learned exactly, and the guaranteed safe/unsafe sets match with the true safe/unsafe sets ($\mathcal{G}_{\neg s} = \mathcal{A}$ and $\mathcal{G}_s = \mathcal{S}$). On the other hand, choosing $\lambda = 0.15$ in (3.17) is a sufficiently large penalty to make the solution of (3.17) match with the demonstrations. However, like the example in Figure 3.6, planning new trajectories from different start or goal states can lead to unsafe trajectories

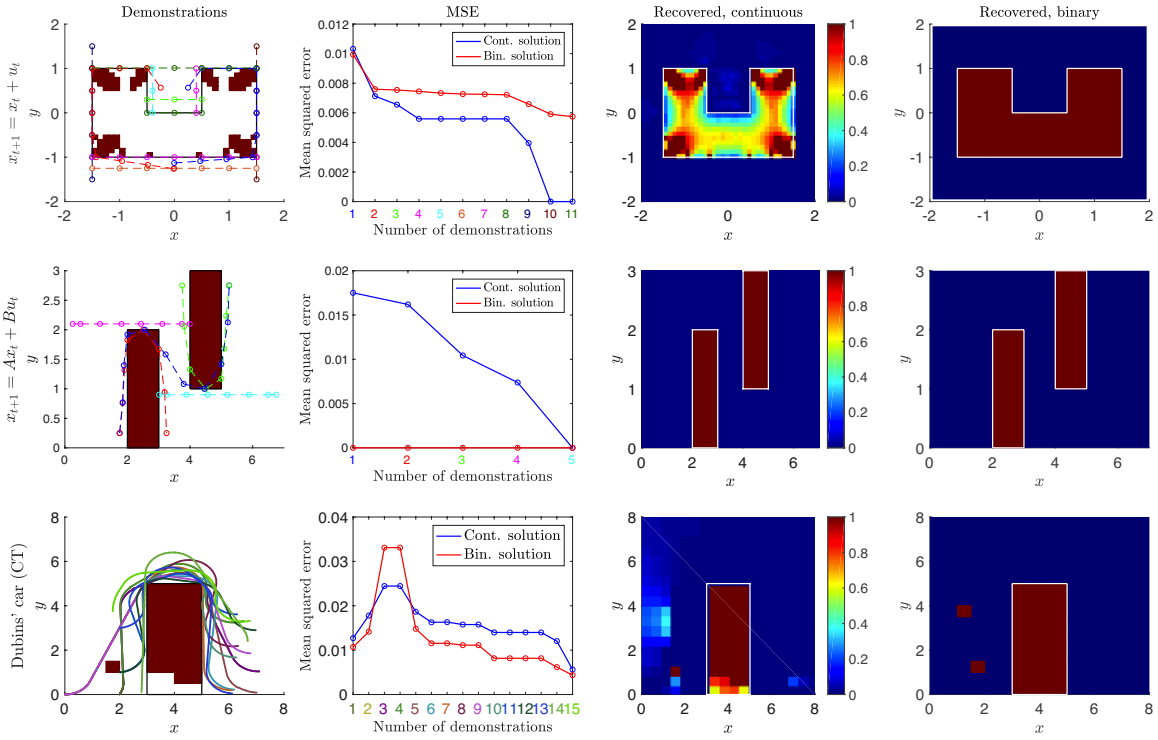


Figure 3.8: Results for various dynamical systems and time discretization. **Rows (top-to-bottom)**: Single integrator; double integrator; Dubins' car (CT). **Columns (left-to-right)**: Demonstrations are plotted together with the outline of the true unsafe set $\mathcal{A}$, and the learned guaranteed unsafe set $\mathcal{G}^z_{\neg s}$ is overlaid (the red cells); mean squared error between the output of Problem III.4 or Problem III.5 and the ground truth; Problem III.5 solution, using all demonstrations; Problem III.4 solution, using all demonstrations.

under the learned cost function (see Figure 3.7, right). Furthermore, the notion of guaranteed unsafeness of a state is not meaningful for the softened case, as states that are avoided ("unsafe") for one pair of start/goal states may be visited ("safe") for a different pair, provided it is less costly to receive a penalty for violating the constraint compared to planning a higher-cost trajectory that satisfies the constraint.

3.5.3 Dynamics and discretization

Experiments in Fig. 3.8 show that our method can be applied to several types of system dynamics, can learn non-convex/multiple unsafe sets, and can use continuous trajectories. All unsafe sets $\mathcal{A}$ are open sets. We solve Problems III.4 and III.5, with an energy function promoting smoothness by penalizing squared deviations of the occupancy of a grid cell z_i from its 4-connected neighbors $N(z_i)$: $\sum_{i=1}^{G} \sum_{z_j \in N(z_i)} \|\mathcal{O}(z_i) - \mathcal{O}(z_j)\|_2^2$. In all experiments, the mean squared error (MSE) is computed as $\frac{1}{G}\sqrt{\sum_{i=1}^{G} \|\mathcal{O}(z_i)^* - \mathcal{O}(z_i)\|_2^2}$, where $\mathcal{O}(z_i)^*$ is the ground truth occupancy. The demonstrations are color-matched with their corresponding number on

the x-axis of the MSE plots. For experiments with more demonstrations, only those causing a notable change in the MSE were color-coded. The learned guaranteed unsafe states $\mathcal{G}^z_{\neg s}$ are colored red on the left column.

First, we recover a non-convex "U-shaped" unsafe set in the state space using trivial 2D single-integrator dynamics: $x = [\chi, y]^\top$, $x_{t+1} = x_t + u_t$, with control constraints $\|u_t\| \leq 0.5$, for all t. The demonstrator minimizes $c(\xi_x, \xi_u) = \sum_{t=1}^{T-1} \|u_t\|_2^2$. The results are shown in row 1 of Fig. 3.8. The solutions to both Problems III.5 and III.4 return reasonable results, and the solution of Problem III.4 achieves zero error.

The second row shows learning two polyhedral unsafe sets in the state space with 4D double integrator linear dynamics: $x = [\chi, \dot{\chi}, y, \dot{y}]^\top$, where $x_{t+1} = Ax_t + Bu_t$, where $A = \exp\left(\text{diag}\left(\begin{bmatrix}0 & 1\\ 0 & 0\end{bmatrix}, \begin{bmatrix}0 & 1\\ 0 & 0\end{bmatrix}\right)\right)$, $B = \int_0^1 \exp(A\tau)d\tau \begin{bmatrix}0 & 1 & 0 & 1\end{bmatrix}^\top$, with control constraints $|u_t| \leq [20, 10]^\top$, for all t. The demonstrator minimizes $c(\xi_x, \xi_u) = \sum_{t=1}^{T-1} \|x_{t+1} - x_t\|_2^2$. The learning procedure yields similar results. We note the linear interpolation of some demonstrations in row 1 and 2 enter $\mathcal{A}$; this is because both sets of dynamics are in discrete time and only the discrete waypoints must stay out of $\mathcal{A}$.

The third row shows learning a polyhedral unsafe set in the state space, with time-discretized continuous, nonlinear Dubins' car dynamics, which has a 3D state $x \doteq \begin{bmatrix}\chi & y & \theta\end{bmatrix}^\top$ and dynamics $\dot{x} = [\cos(\theta), \sin(\theta), u]^\top$ with control constraints $|u| \leq 1$. The demonstrator minimizes $c(\xi_x, \xi_u) = \sum_u \tau_{u_i}$, where τ_{u_i} is the total time duration of applied control input (i.e., the time it took to go from start to goal). These dynamics are more constrained than the previous cases, so sampling lower cost trajectories becomes more difficult, but despite this we can still achieve near zero error solving Problem III.4. Some over-approximation results from some sampled unsafe trajectories entering regions not covered by the safe trajectories, i.e., there are red guaranteed learned unsafe cells outside the true unsafe set. For example, in the right column, the two red blocks to the top left of $\mathcal{A}$ are generated by lower-cost trajectories that trade off the increased cost of entering these grid cells by entering $\mathcal{A}$. This phenomenon is consistent with Theorem III.16; we recover a set that is contained within a $D^* + R^*$-padding of $\mathcal{A}$ (here, $D^* + R^* = 8.2$). Learning curve spikes occur when over-approximation occurs.

Overall, we note that for the gridded case, $\mathcal{G}^z_{\neg s}$ tends to be a significant underapproximation of $\mathcal{A}$ due to the chosen cost function and limited demonstrations. For example, in row 1 of Fig. 3.8, $\mathcal{G}^z_{\neg s}$ cannot contain the portion of $\mathcal{A}$ near long straight edges, since there exists no shorter path going from any start to any goal with only one state within that region. For row 3 of Fig. 3.8, we learn less of the bottom part of $\mathcal{A}$ due to most demonstrations' start and goal locations making it harder to sample feasible control trajectories going through that region; with more demonstrations, this issue becomes less pronounced. In Section 3.6.1, we discuss how using a constraint parameterization can reduce the gap between $\mathcal{G}^z_{\neg s}$ and $\mathcal{A}$.

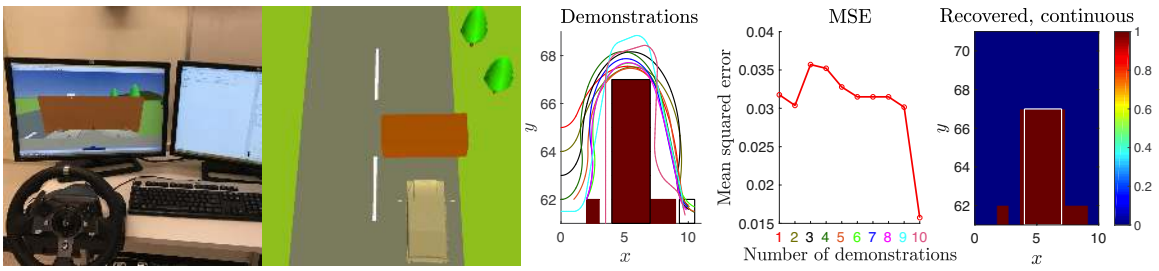


Figure 3.9: Suboptimal demonstrations: **left**: setup, **center**: demonstrations, $\mathcal{A}$, $\mathcal{G}^z_{\neg s}$, **center-right**: MSE, **right**: solution to Problem III.5.

3.5.4 Suboptimal human demonstrations

We demonstrate our method on suboptimal demonstrations collected via a driving simulator, using a car model with CT Dubins' car dynamics identical to those described in Section 3.5.3. Human steering commands were recorded as demonstrations, where the task was to navigate around the orange box and drive between the trees (Fig. 3.9). For a demonstration of cost c, trajectories with cost less than $0.9c$ were believed unsafe with probability 1. Trajectories with cost c' in the interval $[0.9c, c]$ were believed unsafe with probability $1 - ((c' - 0.9c)/0.1c)$. MSE for Problem III.5 is shown in Fig. 3.9 (Problem III.4 is not solved since the probabilistic interpretation is needed). For this problem, D^* is 10 seconds and the unsafe set is grid-aligned; hence, despite suboptimality, the learned guaranteed unsafe set is a subset of $\mathcal{A}(D^*)$. While the MSE is highest here of all experiments, this is expected, as trajectories may be incorrectly labeled safe/unsafe with some probability.

3.5.5 Feature space constraint

We demonstrate that our framework is not limited to the state space by learning a constraint in a feature space. Consider the scenario of planning a safe path for a mobile robot with identical continuous Dubins' car dynamics through hilly terrain, where the magnitude of the terrain's slope is given as a feature map (i.e., $\phi(x) = \|\partial L(\hat{x})/\partial \hat{x}\|_2$, where $\hat{x} = [\chi \; y]^\top$ and $L(\hat{x})$ is the elevation map). The robot will slip if the magnitude of the terrain slope is too large, so we generate a demonstration which obeys the ground truth constraint $\phi(x) < 0.05$; hence, the ground truth unsafe set is $\mathcal{A} \doteq \{x \mid \phi(x) \geq 0.05\}$. From one safe trajectory (Fig. 3.10) generated by RRT* (*Karaman and Frazzoli* (2010)) and gridding the feature space as $\{0, 0.005, \ldots, 0.145, 0.15\}$, we recover the constraint $\phi(x) < 0.05$ exactly.

This example shows how using using a feature parameterization can benefit the sample complexity of our method; in the next section, we show that by using a parameterization, the constraint space gridding used so far can be eliminated to improve our method's scalability.

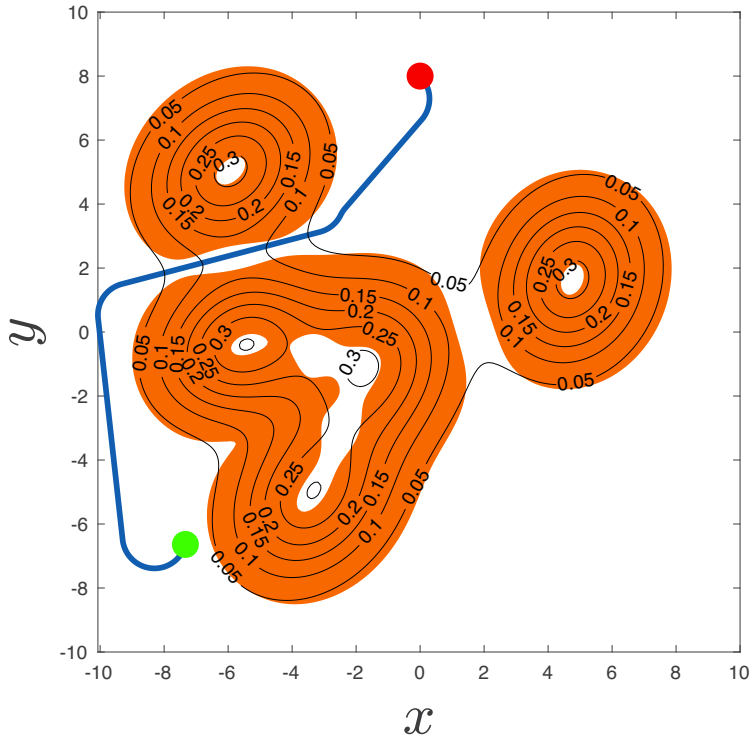


Figure 3.10: Feature space constraint recovery. Unsafe set in the constraint space $\mathcal{A}$ is plotted in orange. The single demonstration is overlaid (red: start, green: goal). Terrain isocontours $L(x) = \text{const}$ are overlaid.

3.6 Evaluations: Parametric

We evaluate the parametric variant of our method on a variety of constraint recovery problems in this section. In particular, we provide examples showing the effect of using a constraint parameterization on sample complexity and guaranteed learnability (Section 3.6.1), the performance of an unknown constraint representation (Section 3.6.2), that our method can be applied to learn a high-dimensional pose constraint (Section 3.6.3), a constraint demonstrated on high-dimensional quadrotor dynamics (Section 3.6.3), and a constraint demonstrated on black-box dynamics (Section 3.6.4).

3.6.1 Comparison to gridded formulation

To demonstrate the advantages provided by assuming a parameterization, we replicate the experiment in the first row of Figure 3.8, assuming that $\mathcal{A}$ can be represented as the union of three axis-aligned boxes. The results are displayed in Figure 3.11. Here, the same grid points are queried for guaranteed safeness or unsafeness as described in Section 3.3.5.

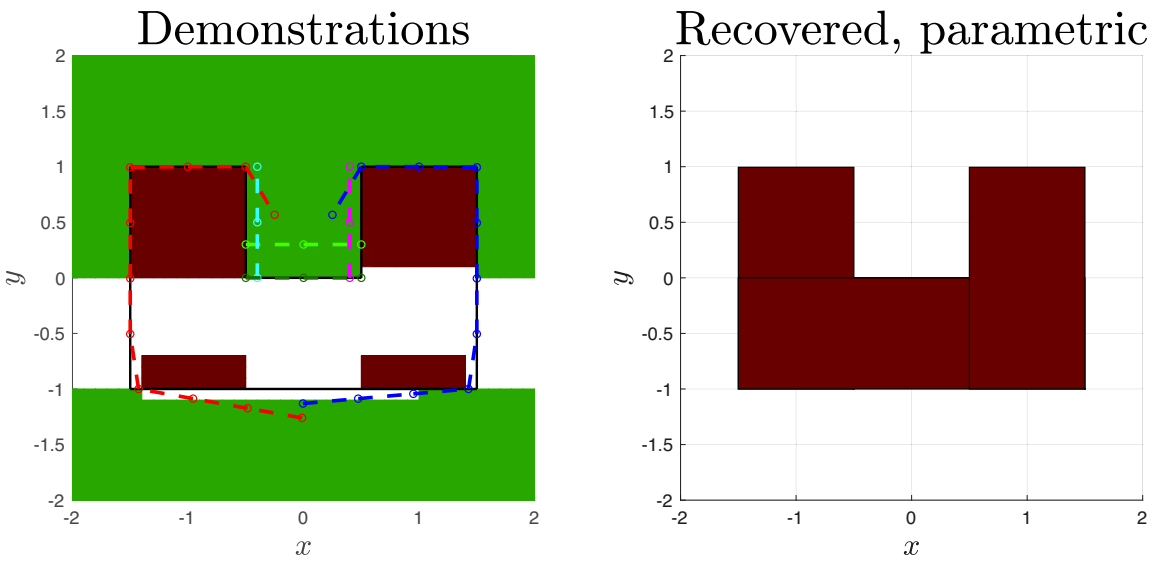


Figure 3.11: Replicating row 1 of Figure 3.8 using a three-box parameterization. **Left**: $\mathcal{G}_{\neg s}$ is shaded in red and $\mathcal{G}_s$ is shaded in green. Demonstrations are overlaid and color-coded to match with row 1 of Figure 3.8. **Right**: Recovered constraint using a variant of Problem III.7.

Compared to the gridded approach, the true unsafe set can be exactly recovered with just six of the original eleven demonstrations (demonstrations 1-5 and 8 in row 1 of Figure 3.8). This improved sample complexity arises from the fact that our method can extrapolate that some unseen states are safe or unsafe using the parameterization. On the contrary, in the gridded formulation, each grid cell is independent and learning that some cell is unsafe can never imply that another cell is unsafe; only learning that a cell is safe can imply that another cell is unsafe.

Note that compared to Figure 3.8, a non-trivial $\mathcal{G}_s$ containing states not explicitly covered by demonstrations can now be recovered. Furthermore, $\mathcal{G}_{\neg s}$ covers a larger fraction of the true unsafe set than compared to the gridded approach. As just discussed, this arises from the fact that given the parameterization and some guaranteed unsafe states, other states can be implied unsafe. As a result, $\mathcal{G}_{\neg s}$ expands to include the set of states which must be unsafe to be compatible with the safe and unsafe trajectories and the parameterization as well.

3.6.2 Unknown parameterization

U-shape: We first present a kinematic 2D example where a U-shape $\mathcal{A}$ is to be learned, but the number of simple unsafe sets needed to represent $\mathcal{A}$ (three) is unknown. In Row 1, Column 1 of Fig. 3.12, we outline $\mathcal{A}$ in black and overlay $\mathcal{G}_{\neg s}$, $\mathcal{G}_s$, and the six provided demonstrations, synthetically generated via trajectory optimization. We note that due to the chosen control constraints and U-shape, there are parts of $\mathcal{A}$ (a subset of the white region in Fig. 3.12, Row 1, Column 1) which cannot be implied unsafe by sampled unsafe trajectories and the parameterization (see Theorem III.19). As a result, $\mathcal{G}_{\neg s}$ may not fully cover $\mathcal{A}$, even with more demon-

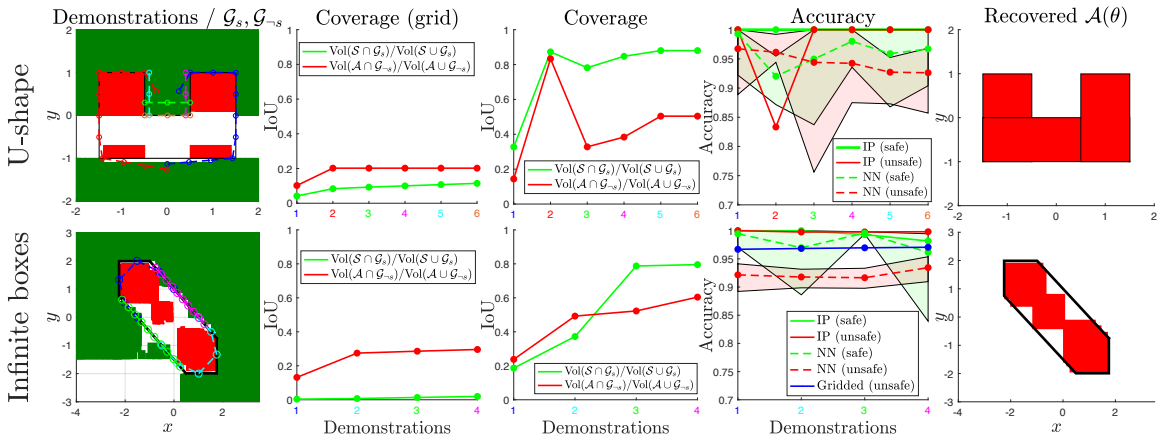


Figure 3.12: Unknown parameterization. **Col. 1**: Red: $\mathcal{G}_{\neg s}$; Green: $\mathcal{G}_s$. Demonstrations are overlaid. **Col. 2**: Coverage of $\mathcal{A}$ and $\mathcal{S}$ with a grid representation. In this (and all later examples), the demonstrations are color-coded with x-axis. **Col. 3**: Coverage of $\mathcal{A}$ and $\mathcal{S}$ with our method. **Col. 4**: Classification accuracy (dotted: average NN accuracy, shaded: range of NN accuracies over 10 random seeds). **Col. 5**: Recovered constraint with multi-polytope variant of Problem III.7.

strations (Fig. 3.12, Row 1, Column 3). Note that the decrease in coverage[4] at the third demonstration is due to a increase from a two-box parameterization to a three-box parameterization. Likewise, the accuracy[5] decreases at the second demonstration due to over-approximation of $\mathcal{A}$ with two boxes (Fig. 3.12, Row 1, Column 4), but this over-approximation vanishes when switching to the three-box parameterization (which is exact; hence $\mathcal{G}_s$ and $\mathcal{G}_{\neg s}$ are guaranteed conservative, cf. Theorem III.21). The grid-based method always has perfect accuracy, since it does not extrapolate beyond the observed trajectories. However, as a result of that, it also yields low coverage (Fig. 3.12, Row 1, Column 2). The NN baseline achieves lower accuracy for the unsafe set as it misclassifies some corners of the U. Recovering a feasible θ using a multi-box variant of Problem III.7 recovers $\mathcal{A}$ exactly (Fig. 3.12, Row 1, Column 5). Finally, we note that in this (and future) examples, demonstrations were specifically chosen to be informative about the constraint. We present a version of this example in Appendix A.2 with random demonstrations and show that the constraint is still learned (albeit needing more demonstrations).

Infinite boxes: To show that our method can still learn a constraint that cannot be easily expressed using a chosen parameterization, we limit our parameterization to an unknown number of axis-aligned boxes and attempt to learn a diagonal "I" unsafe set (see Fig. 3.12, Row 2). This is a particularly difficult example, since an

[4]Coverage is measured as the intersection over union (IoU) of the relevant sets (see legends for exact formula).

[5]In all experiments, computed accuracies are: IP (safe) = $\mathrm{Vol}(\mathcal{G}_s \cap \mathcal{S})/\mathrm{Vol}(\mathcal{G}_s)$, IP (unsafe) = $\mathrm{Vol}(\mathcal{G}_{\neg s} \cap \mathcal{A})/\mathrm{Vol}(\mathcal{G}_{\neg s})$, NN (safe) = $(\sum_{i=1}^{q} \mathbf{I}_{(x_i \in \mathcal{S}) \wedge (\text{NN classified } x_i \text{ as safe})})/\sum_{i=1}^{q} \mathbf{I}_{x_i \in \mathcal{S}}$, NN (unsafe) = $(\sum_{i=1}^{q} \mathbf{I}_{(x_i \in \mathcal{A}) \wedge (\text{NN classified } x_i \text{ as unsafe})})/\sum_{i=1}^{q} \mathbf{I}_{x_i \in \mathcal{A}}$, where $x_1, \ldots, x_q$ are query states sampled from $\mathcal{G}_{\neg s} \cup \mathcal{G}_s$ and $\mathbf{I}_{(\cdot)}$ is the indicator function. Note that NN accuracy is computed only on $(\mathcal{G}_s \cup \mathcal{G}_{\neg s}) \subseteq \mathcal{C}$.

infinite number of axis-aligned boxes will be needed to recover $\mathcal{A}$ exactly. However, for finite data, only a finite number of boxes will be needed; in particular, for 1, 2, 3, and 4 demonstrations (which are synthetically generated assuming kinematic system constraints), 3, 5, 6, and 6 boxes are required to generate a parameterization consistent with the data (see Fig. 3.12, Row 2, Column 1). Also overlaid in Fig. 3.12, Row 2, Column 1 are $\mathcal{G}_{\neg s}$ and $\mathcal{G}_s$, which are approximated by solving Problem III.6 for randomly sampled κ_{center}. Compared to the gridded formulation (see Fig. 3.12, Row 2, Column 3), $\mathcal{G}_s$ and $\mathcal{G}_{\neg s}$ cover $\mathcal{S}$ and $\mathcal{A}$ far better due to the parameterization enabling the IP to extrapolate more from the demonstrations. Furthermore, we note that while the gridded case has perfect accuracy for the safe set, it does not for the unsafe set, due to grid alignment. Overall, the multi-box variant of Problem III.7 recovers $\mathcal{A}$ well (Fig. 3.12, Row 2, Column 5), and the remaining gap can be improved with more data. Last, we note that the NN baseline reaches comparable accuracies here (Fig. 3.12, Row 2, Column 4), since our method suffers from a few disadvantages for this particular example. First, attempting to represent the "I" with a finite number of boxes introduces a modeling bias that the NN does not have. Second, since the system is kinematic and the constraint is low-dimensional, many unsafe trajectories can be sampled, providing good coverage of the unsafe set. We show later that for higher dimensional constraints/systems with highly constrained dynamics, it becomes difficult to gather enough data for the NN to perform well.

3.6.3 High-dimensional examples

6D pose constraint for a 7-DOF robot arm: In this example, we learn a 6D hyper-rectangular pose constraint for the end effector of a 7-DOF Kuka iiwa arm. One such setting is when the robot is to bring a cup to a human while ensuring its contents do not spill (angle constraint) and proxemics constraints (i.e., the end effector never gets too close to the human) are satisfied (position constraint). We examine this problem for the cases of optimal and suboptimal demonstrations.

Demonstration setup: The end effector orientation (parametrized in Euler angles) and position are constrained to satisfy $(\alpha, \beta, \gamma) \in [\underline{\alpha}, \bar{\alpha}] \times [\underline{\beta}, \bar{\beta}] \times [\underline{\gamma}, \bar{\gamma}]$ and $(x, y, z) \in [\underline{x}, \bar{x}] \times [\underline{y}, \bar{y}] \times [\underline{z}, \bar{z}]$ (see Fig. 3.13, Column 1). For the optimal case, we synthetically generate seven demonstrations minimizing joint-space trajectory length. For the suboptimal case, five suboptimal continuous-time demonstrations approximately optimizing joint-space trajectory length are recorded in a virtual reality environment, where a human demonstrator moves the arm from desired start to goal end effector configurations using an HTC Vive (see Fig. A.7). The demonstrations are time-discretized for lower-cost trajectory sampling. In both cases, the constraint is recovered with Problem III.7, where $H(\theta) = [I, -I]^\top$ and $h(\theta) = \theta = [\bar{x}, \bar{y}, \bar{z}, \bar{\alpha}, \bar{\beta}, \bar{\gamma}, \underline{x}, \underline{y}, \underline{z}, \underline{\alpha}, \underline{\beta}, \underline{\gamma}]^\top$. For the suboptimal case, slack variables are added to ensure feasibility of Problem III.7, and for a suboptimal demonstration of cost $\hat{c}$, we only use trajectories of cost less than $0.9\hat{c}$ as unsafe trajectories.

Results: The coverage plots (Fig. 3.13, Rows 1 and 3, Col. 2) show that as the number of demonstrations increases, $\mathcal{G}_s/\mathcal{G}_{\neg s}$ approach the true safe/unsafe sets

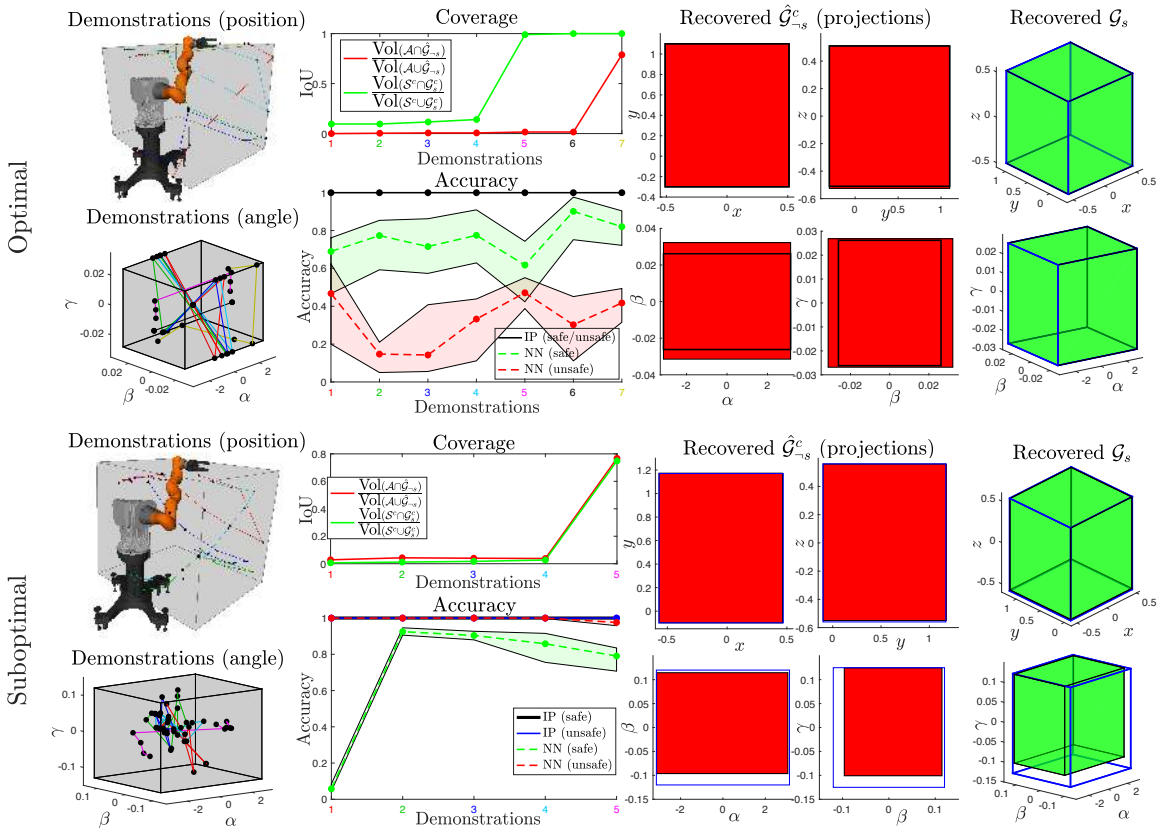


Figure 3.13: **Rows 1:2**: 7-DOF arm, optimal demonstrations **Col. 1**: Experimental setup. Gray boxes are projections of $\mathcal{A}$. Projections of demonstrations in position/angle space are overlaid. **Col. 2**: *Top*: Comparing safe/unsafe set coverage as a function of demonstrations. *Bottom*: Prediction accuracy. **Cols. 3-4**: projections of $\hat{\mathcal{G}}_{\neg s}$ using all demonstrations. For the optimal case, the red boxes over-approximate the blue boxes, as the complement of $\hat{\mathcal{G}}_{\neg s}$ (not $\hat{\mathcal{G}}_{\neg s}$ itself) is plotted. **Col. 5**: projections of $\mathcal{G}_s$ using all demonstrations. **Rows 3:4**: Same for 7-DOF arm, suboptimal demonstrations.

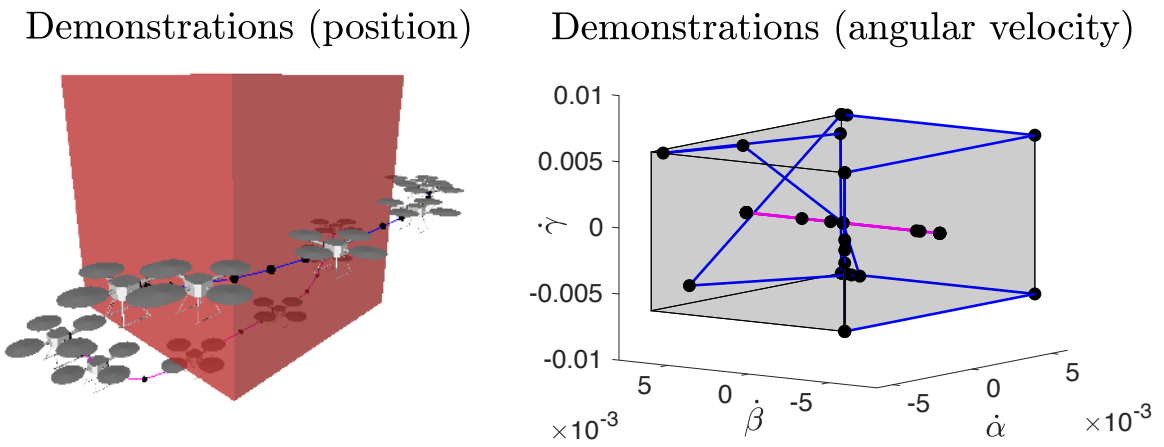


Figure 3.14: **Left**: Known unsafe set in (x, y, z) (red); (x, y, z) components of demonstrations are overlaid. **Right**: Unknown unsafe set in $(\dot{\alpha}, \dot{\beta}, \dot{\gamma})$ (gray); $(\dot{\alpha}, \dot{\beta}, \dot{\gamma})$ components of demonstrations are overlaid.

$\mathcal{S}/\mathcal{A}$ [6]. For the suboptimal case, the low IoU values for lower numbers of demonstrations is due to overapproximation of the unsafe set in the α component (arising from continuous-time discretization and imperfect knowledge of the suboptimality bound); the fifth demonstration, where α takes values near $-\pi, \pi$ greatly reduces this overapproximation. The accuracy plots (Fig. 3.13, Rows 2 and 4, Col. 2) present results consistent with the theory: for the optimal case, all constraint states in $\mathcal{G}_s$ and $\mathcal{G}_{\neg s}$ are truly safe and unsafe (Theorem III.21), and the small over-approximation for the suboptimal case is consistent with the continuous-time conservativeness (Theorem III.22). Note that the NN accuracy is lower and can oscillate with demonstrations, since it finds just a single constraint which is approximately consistent with the data, while our method classifies safety by consulting all possible constraints which are exactly consistent with the data, thus performing more consistently. The NN performs better on the suboptimal case than it does on the optimal case, as more unsafe trajectories are sampled due to the suboptimality, improving coverage of the unsafe set. The projections of $\hat{\mathcal{G}}^c_{\neg s}$ (Fig. 3.13, Cols. 3-4, in red), where $\hat{\mathcal{G}}_{\neg s} \subseteq \mathcal{G}_{\neg s}$ is obtained using the method in Section 3.3.5.1, are compared to the safe set (blue outline), showing that the two match nearly exactly (though the gap for the suboptimal case is larger), and the gap can be likely reduced with more demonstrations. The projections of $\mathcal{G}_s$ (Fig. 3.13, Col. 5) match exactly with $\mathcal{A}$ for the optimal case (true safe set is outlined in blue) and match closely for the suboptimal case. Note that $\mathcal{G}_s \subseteq \mathcal{S}$, as is the case for all axis-aligned box parameterizations.

3D constraint for 12D quadrotor model: We learn a 3D box angular velocity constraint for a quadrotor with discrete-time 12D dynamics (see Appendix A.3 for details). In this scenario, the quadrotor must avoid an a priori known unsafe set in position space while also ensuring that angular velocities are below a threshold:

[6] For the unsafe sets, the IoUs are computed between $\mathcal{G}^c_{\neg s}$ and $\mathcal{A}^c$, as in high dimensions, the IoU changes more smoothly for the complements than the IoU between $\mathcal{G}_{\neg s}$ and $\mathcal{A}$, so we plot the the former for visual clarity.

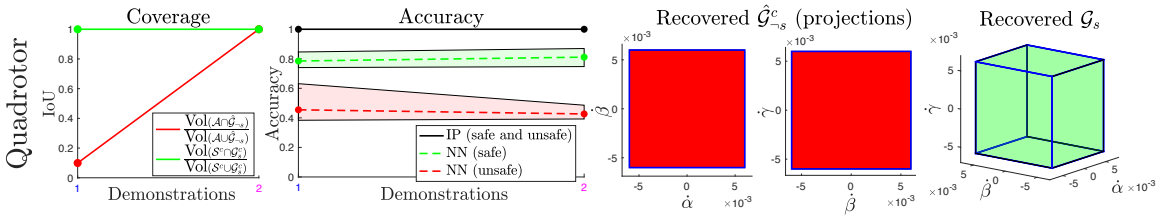


Figure 3.15: Constraint recovery for a 12D quadrotor. **Col. 1**: Coverage of $\mathcal{A}$ and $\mathcal{S}$. **Col. 2**: Classification error between $\mathcal{G}_s/\mathcal{S}$ and $\mathcal{G}_{\neg s}/\mathcal{A}$. **Cols. 3-4**: $\hat{\mathcal{G}}_{\neg s}$ using all demonstrations. **Col. 5**: $\mathcal{G}_s$ using all demonstrations.

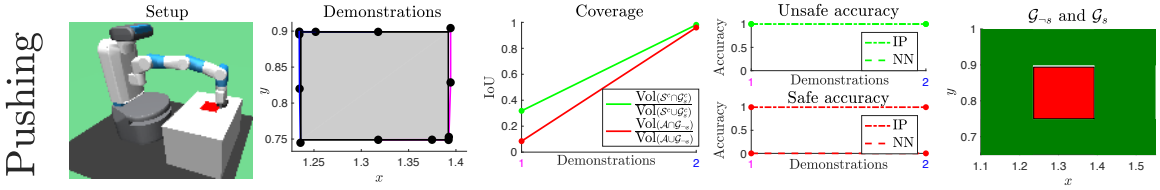


Figure 3.16: Constraint recovery without closed-form dynamics. **Cols. 1-2**: Setup (unsafe set in red) and demonstrations (unsafe set in gray). **Cols. 3-4**: Coverage of $\mathcal{A}$ and $\mathcal{S}$; classification accuracy. **Col. 5**: $\mathcal{G}_{\neg s}$ / $\mathcal{G}_s$ using all demonstrations.

$(\dot{\alpha}, \dot{\beta}, \dot{\gamma}) \in [\underline{\dot{\alpha}}, \bar{\dot{\alpha}}] \times [\underline{\dot{\beta}}, \bar{\dot{\beta}}] \times [\underline{\dot{\gamma}}, \bar{\dot{\gamma}}]$. The $(\dot{\alpha}, \dot{\beta}, \dot{\gamma})$ safe set is to be inferred from two demonstrations (see Fig. 3.14). The constraint is recovered with Problem III.7, where $H(\theta) = [I, -I]^\top$ and $h(\theta) = \theta = [\bar{\dot{\alpha}}, \bar{\dot{\beta}}, \bar{\dot{\gamma}}, \underline{\dot{\alpha}}, \underline{\dot{\beta}}, \underline{\dot{\gamma}}]^\top$. Fig. 3.15 shows that with more demonstrations, $\mathcal{G}_s$ approaches the true safe set $\mathcal{S}$ and $\mathcal{G}_{\neg s}$ approaches the true unsafe set $\mathcal{A}$, respectively. Consistent with Theorem III.21, our method has perfect accuracy in $\mathcal{G}_{\neg s}$ and $\mathcal{G}_s$. Here, the NN struggles more compared to the arm examples since due to the more constrained dynamics, fewer unsafe trajectories can be sampled, and a parameterization needs to be leveraged in order to say more about the unsafe set. The remaining columns of Fig. 3.15 show that we recover $\mathcal{G}_{\neg s}$ and $\mathcal{G}_s$ exactly (the true safe set is outlined in blue).

3.6.4 Planar pushing example

In this section, using the `FetchPush-v1` environment in OpenAI Gym *Plappert et al.* (2018), we aim to learn a 2D box unsafe set on the center-of-mass (CoM) of a block pushed by the Fetch arm (see Fig. 3.16) using two demonstrations. Here, the dynamics of the block CoM are not known in closed form, but rollouts can still be sampled using the simulator. Since the block CoM is highly underactuated, it is not possible to sample short sub-trajectories. Thus, without leveraging a parameterization, the constraint recovery problem is very ill-posed. Furthermore, while our method can explicitly consider the unsafeness in longer unsafe trajectories (at least one state is unsafe), the NN struggles with this example as it fails to accurately model that fact. Overall, Fig. 3.16 presents that $\mathcal{G}_{\neg s}/\mathcal{G}_s$ match up well with $\mathcal{A}/\mathcal{S}$, and our classification accuracy for safeness/unsafeness is perfect across demonstrations.

3.7 Discussion

In this section, we summarize the main takeaways from the theoretical analysis and experiments:

Learnability of unsafe states: When gridding the constraint space, only grid cells that lie within some distance to the boundary of the unsafe set can be learned guaranteed unsafe. For discrete time systems, this distance is the maximum distance the system can travel in one time-step (see Theorem III.9); for continuous systems, without time discretization, only the boundary of the unsafe set can be learned guaranteed unsafe (see Theorem III.10). This is reflected in the first row of Figure 3.8, where cells further from the boundary of the unsafe set are not learned guaranteed unsafe. However, when leveraging a constraint parameterization, the set of constraint states that can be learned guaranteed unsafe expands to states which can be implied unsafe by states within some distance to the boundary and the parameterization (Theorem III.19). This can be seen in all the high-dimensional examples (Section 3.6.3), where states deep in the unsafe set can be learned guaranteed unsafe.

Learnability of safe states: When gridding the constraint space, due to the independence of grid cells (i.e., learning that some cell is guaranteed unsafe can never imply that a different cell is guaranteed safe), the only cells that can be learned guaranteed safe are those visited by demonstrations. However, when using a parameterization, learning that certain states are unsafe can imply that other states must be safe, under the assumption that the true constraint can be represented with the given parametrization (see Figure 3.3 and the examples in Section 3.6.3).

Conservativeness of guaranteed learned unsafe states: When gridding the constraint space, under assumptions on alignment of the grid with the unsafe set and discretization frequency, the set of guaranteed learned unsafe states is conservative (see Theorems III.15 and III.16). This is demonstrated in the results (Figure 3.8, rows 1 and 2). When these assumptions do not hold, the set of guaranteed learned unsafe states is contained within a padded version of the true unsafe set (see Figure 3.8, row 3, and Figure 3.9 for examples where overapproximation occurs due to the time-discretization chosen). When using a parameterization, the set of guaranteed learned unsafe states is conservative for discrete time systems (see Theorem III.21) and is conservative within a padded version of the true unsafe set for continuous time systems (see Corollary III.22). Examples of this conservativeness are shown in Figures 3.13, Rows 1-2 and 3.15, and an example where overapproximation occurs due to continuous dynamics is shown in Figure 3.13, Rows 3-4.

Limitations: Some limitations of our method are as follows:

- Sampling lower-cost trajectories can be slow for systems where the set of lower-cost trajectories satisfying the known constraints, $\mathcal{T}_{\mathcal{A}}^{\xi^*_{xu}}$, is "thin". For these cases, hit-and-run sampling can be forced to take very small steps at each iteration, reducing the spread of samples inside $\mathcal{T}_{\mathcal{A}}^{\xi^*_{xu}}$. This tends to happen when the dynamics are highly constrained. In future work, we will investigate more efficient sampling techniques for when $\mathcal{T}_{\mathcal{A}}^{\xi^*_{xu}}$ takes a specific form.

- While we want the set of guaranteed learned (un)safe states to be a conservative

estimate, the level of conservativeness may be high. Excessive conservativeness can be mitigated for the parametric case by obtaining the set of constraint parameters which are consistent with the demonstrations and computing a probabilistic measure of how (un)safe a given state is based on how many consistent parameters mark it as (un)safe, as is proposed in Chapter VII.

- While our method is resilient to suboptimal demonstrations within a known bound of the globally-optimal cost, it lacks guarantees for locally-optimal demonstrations. An alternative approach using the Karush-Kahn-Tucker optimality conditions is described in Chapter IV, which enables constraint learning from locally-optimal demonstrations. Another method *Knuth et al.* (2021b) has been developed to handle demonstrations with large suboptimality bound, where the suboptimality arises from visual occlusions that limit the demonstrators' knowledge about the environment and thus their ability to plan optimally.

3.8 Conclusion

In this chapter we propose an algorithm that learns constraints from demonstrations, which acts as a complementary method to IOC/IRL algorithms. We analyze the properties of our algorithm as well as the theoretical limits of what subset of a safe set and an unsafe set can be learned from only safe demonstrations. The method works well on a variety of high-dimensional system dynamics and can be adapted to work with suboptimal demonstrations. We develop two variants of our algorithm to learn constraints with various amounts of structure: a gridded version which assumes no constraint structure but scales exponentially with constraint dimension, and a parametric version which assumes known parametric constraint structure and scales gracefully to high dimensional constraint spaces. We further show that our method can also learn constraints in a feature space.

CHAPTER IV

Learning Constraints from Locally-Optimal Demonstrations

In this chapter, we present an algorithm for learning parametric constraints from locally-optimal demonstrations, where the cost function being optimized is uncertain to the learner. Our method uses the Karush-Kuhn-Tucker (KKT) optimality conditions of the demonstrations within a mixed integer linear program (MILP) to learn constraints which are consistent with the local optimality of the demonstrations, by either using a known constraint parameterization or by incrementally growing a parameterization that is consistent with the demonstrations. We provide theoretical guarantees on the conservativeness of the recovered safe/unsafe sets and analyze the limits of constraint learnability when using locally-optimal demonstrations. We evaluate our method on high-dimensional constraints and systems by learning constraints for 7-DOF arm and quadrotor examples, show that it outperforms competing constraint-learning approaches, and can be effectively used to plan new constraint-satisfying trajectories in the environment. This chapter is based off of the paper *Chou et al.* (2020b).

4.1 Introduction

Initial work in Chapter III has taken steps towards identifying constraints from approximately globally-optimal expert demonstrations, assuming that the demonstrator's cost function is known exactly. However, as humans are not always experts at performing a task, requiring them to provide demonstrations which are nearly globally-optimal can be unreasonable. Furthermore, it is rare for the cost function being optimized to be known exactly by the learner. To address these shortcomings, we consider the problem of learning parametric constraints shared across tasks from approximately locally-optimal demonstrations under parametric cost function uncertainty. Our method is based on the insight that locally-optimal, constraint-satisfying demonstrations satisfy the Karush-Kuhn-Tucker (KKT) optimality conditions, which are first-order necessary conditions for local optimality of a solution to a constrained discrete-time optimal control problem. We solve a mixed integer linear program (MILP) to recover constraint and cost function parameters which make the demonstrations locally-optimal. We make the following specific contributions in this chapter:

- We develop a novel algorithm for learning parametric, potentially non-convex constraints from approximately locally-optimal demonstrations, where the parameterization can either be provided or grown incrementally to be consistent with the data. The method can extract volumes of safe/unsafe states (states which satisfy/do not satisfy the constraints) for future guaranteed safe planning and enable planners to query states for safety.
- Our method can learn constraints despite uncertainty in the cost function and can also recover a cost function jointly with the constraint.
- Under mild assumptions, we prove that our method recovers guaranteed conservativeness estimates (that is, inner approximations) of the true safe/unsafe sets, and analyze the learnability of a constraint from locally-optimal compared to globally-optimal demonstrations.
- We evaluate our method on difficult constraint learning problems in high-dimensional constraint spaces (23 dimensions) on systems with complex nonlinear dynamics and demonstrate that our method outperforms previous methods for parametric constraint inference (*Chou et al.* (2018a, 2019), Chapter III).

4.2 Preliminaries and Problem Setup

We consider discrete-time nonlinear systems $x_{t+1} = f(x_t, u_t, t)$, $x \in \mathcal{X}$ and $u \in \mathcal{U}$, performing tasks Π, which are represented as constrained optimization problems over state/control trajectories $\xi_{xu} \doteq (\xi_x, \xi_u)$:

Problem IV.1 (Forward problem / "task" Π)**.**

$$\begin{array}{ll} \underset{\xi_{xu}}{\text{minimize}} & c(\xi_{xu}, \gamma) \\ \text{subject to} & \phi(\xi_{xu}) \in \mathcal{S}(\theta) \subseteq \mathcal{C} \\ & \bar{\phi}(\xi_{xu}) \in \bar{\mathcal{S}} \subseteq \bar{\mathcal{C}} \\ & \phi_\Pi(\xi_{xu}) \in \mathcal{S}_\Pi \subseteq \mathcal{C}_\Pi \end{array} \tag{4.1}$$

where $c(\cdot, \gamma)$ is a potentially non-convex cost function, parameterized by $\gamma \in \Gamma$. In Sec. 4.3.1 to 4.3.3, we assume that γ is known (through possibly inaccurate prior knowledge) for clarity; we later relax this assumption and discuss how to learn γ from the demonstrations. Further, $\phi(\cdot)$ is a known mapping from state-control trajectories to a constraint space $\mathcal{C}$, elements of which are referred to as *constraint states* $\kappa \in \mathcal{C}$. Mappings $\bar{\phi}(\cdot)$ and $\phi_\Pi(\cdot)$ are known and map to constraint spaces $\bar{\mathcal{C}}$ and $\mathcal{C}_\Pi$, containing a known shared safe set $\bar{\mathcal{S}}$ and a known task-dependent safe set $\mathcal{S}_\Pi$, respectively. In this chapter, we encode the system dynamics in $\bar{\mathcal{S}}$ and start/goal constraints in $\mathcal{S}_\Pi$. Grouping the constraints of Problem IV.1 as equality/inequality (eq/ineq) constraints

and known/unknown ($k/\neg k$) constraints, we can write:

$$\begin{aligned} h_{i,k}(\xi_{xu}) = 0, i = 1, ..., N_k^{\text{eq}} \quad &\Leftrightarrow \mathbf{h}_k(\xi_{xu}) = \mathbf{0} \\ g_{i,k}(\xi_{xu}) \leq 0, i = 1, ..., N_k^{\text{ineq}} \quad &\Leftrightarrow \mathbf{g}_k(\xi_{xu}) \leq \mathbf{0} \\ g_{i,\neg k}(\xi_{xu}, \theta) \leq 0, i = 1, ..., N_{\neg k}^{\text{ineq}} \quad &\Leftrightarrow \mathbf{g}_{\neg k}(\xi_{xu}, \theta) \leq \mathbf{0} \end{aligned} \tag{4.2}$$

where $\mathbf{h}_k(\xi_{xu}) \in \mathbb{R}^{N_k^{\text{eq}}}$, $\mathbf{g}_k(\xi_{xu}) \in \mathbb{R}^{N_k^{\text{ineq}}}$, and $\mathbf{g}_{\neg k}(\xi_{xu}, \theta) \in \mathbb{R}^{N_{\neg k}^{\text{ineq}}}$. Note that unknown equality constraints $\mathbf{h}_{\neg k}(\xi_{xu}, \theta) = 0$ can be written equivalently as $\mathbf{h}_{\neg k}(\xi_{xu}, \theta) \leq 0, -\mathbf{h}_{\neg k}(\xi_{xu}, \theta) \leq 0$. As shorthand, let $g(\kappa, \theta) \doteq \max_{i \in \{1,...,N_{\neg k}^{\text{ineq}}\}} \left(g_{i,\neg k}(\kappa, \theta)\right)$. We now define

$$\mathcal{S}(\theta) \doteq \{\kappa \in \mathcal{C} \mid g(\kappa, \theta) \leq 0\} \tag{4.3}$$

$$\mathcal{A}(\theta) \doteq \mathcal{S}(\theta)^c = \{\kappa \in \mathcal{C} \mid g(\kappa, \theta) > 0)\} \tag{4.4}$$

as an unknown safe/unsafe set defined by unknown parameter $\theta \in \Theta$, for possibly unknown parameterizations $g_{i,\neg k}(\cdot, \cdot)$. Last, we restrict Γ and Θ to be unions of polytopes.

Intuitively, a trajectory ξ_{xu} is locally-optimal if all trajectories within a neighborhood of ξ_{xu} have cost greater than or equal to $c(\xi_{xu})$. More precisely, for a trajectory to be locally-optimal, it necessarily satisfies the KKT conditions *Boyd and Vandenberghe* (2004). We define a demonstration ξ^{loc} as a state-control trajectory which we assume approximately solves Problem IV.1 to local optimality, i.e., it satisfies all constraints and is in the neighborhood of a local optimum.

Our goal is to recover the safe set $\mathcal{S}(\theta)$ and unsafe set $\mathcal{A}(\theta)$, given N_s demonstrations $\{\xi_j^{\text{loc}}\}_{j=1}^{N_s}$, known shared safe set $\bar{\mathcal{S}}$, and task-dependent constraints $\mathcal{S}_\Pi$. As a byproduct, our method can also recover unknown cost parameters γ.

4.3 Method

We detail our constraint-learning algorithm. First, we formulate the general KKT-based constraint recovery problem (Sec. 4.3.1) and then develop specific optimization problems for the cases where the constraint is defined as a union of offset-parameterized (Sec. 4.3.2) or affinely-parameterized constraints (Sec. 4.3.4). We show how to extract guaranteed safe/unsafe states (Sec. 4.3.3), handle unknown constraint parameterizations (Sec. 4.3.5), and handle cost function uncertainty (Sec. 4.3.6). In closing, we show how our method can be used within a planner to guarantee safety (Sec. 4.3.7).

4.3.1 Constraint recovery via the KKT conditions

Recall that the KKT conditions are necessary conditions for local optimality of a solution of a constrained optimization problem *Boyd and Vandenberghe* (2004). For constraints (4.2) and Lagrange multipliers λ and ν, the KKT conditions for the jth locally-optimal demonstration ξ_j^{loc}, denoted $\text{KKT}(\xi_j^{\text{loc}})$, are:

$$
\begin{aligned}
&\text{Primal feasibility:} && \mathbf{h}_k(\xi_j^{\text{loc}}) = \mathbf{0}, && (4.5a)\\
& && \mathbf{g}_k(\xi_j^{\text{loc}}) \leq \mathbf{0}, && (4.5b)\\
& && \mathbf{g}_{\neg k}(\xi_j^{\text{loc}}, \theta) \leq \mathbf{0}, && (4.5c)\\
&\text{Lagrange mult.} && \lambda_{i,k}^j \geq 0, \;\; i = 1, ..., N_k^{\text{ineq}} \;\; \Leftrightarrow \;\; \boldsymbol{\lambda}_k^j \geq \mathbf{0} && (4.5d)\\
&\text{nonnegativity:} && \lambda_{i,\neg k}^j \geq 0, \;\; i = 1, ..., N_{\neg k}^{\text{ineq}} \;\; \Leftrightarrow \;\; \boldsymbol{\lambda}_{\neg k}^j \geq \mathbf{0} && (4.5e)\\
&\text{Complementary} && \boldsymbol{\lambda}_k^j \odot \mathbf{g}_k(\xi_j^{\text{loc}}) = \mathbf{0} && (4.5f)\\
&\text{slackness:} && \boldsymbol{\lambda}_{\neg k}^j \odot \mathbf{g}_{\neg k}(\xi_j^{\text{loc}}, \theta) = \mathbf{0} && (4.5g)\\
&\text{Stationarity:} && \nabla_{\xi_{xu}} c_{\Pi}(\xi_j^{\text{loc}}) + \boldsymbol{\lambda}_k^{j\top} \nabla_{\xi_{xu}} \mathbf{g}_k(\xi_j^{\text{loc}}) \\
& && \quad + \boldsymbol{\lambda}_{\neg k}^{j\top} \nabla_{\xi_{xu}} \mathbf{g}_{\neg k}(\xi_j^{\text{loc}}, \theta) && (4.5h)\\
& && \quad + \boldsymbol{\nu}_k^{j\top} \nabla_{\xi_{xu}} \mathbf{h}_k(\xi_j^{\text{loc}}) = \mathbf{0}
\end{aligned}
$$

where $\nabla_{\xi_{xu}}(\cdot)$ takes the gradient with respect to a flattened trajectory ξ_{xu} and $\odot$ denotes elementwise multiplication. For compactness, we vectorize the multipliers $\boldsymbol{\lambda}_k^j \in \mathbb{R}^{N_k^{\text{ineq}}}$, $\boldsymbol{\lambda}_{\neg k}^j \in \mathbb{R}^{N_{\neg k}^{\text{ineq}}}$, and $\boldsymbol{\nu}_k^j \in \mathbb{R}^{N_k^{\text{ineq}}}$. We drop the γ dependency, as the cost is assumed known for now, as well as (4.5a)-(4.5b), as they involve no decision variables. Then, finding a constraint consistent with the local optimality conditions of the N_s demonstrations amounts to finding a constraint parameter θ which satisfies the KKT conditions for each demonstration. That is, we can solve the following feasibility problem:

Problem IV.2 (KKT inverse, locally-optimal)**.**

$$
\begin{aligned}
\text{find} \quad & \theta, \boldsymbol{\lambda}_k^j, \boldsymbol{\lambda}_{\neg k}^j, \boldsymbol{\nu}_k^j, \quad j = 1, ..., N_s \\
\text{s.t.} \quad & \{\text{KKT}(\xi_j^{\text{loc}})\}_{j=1}^{N_s}
\end{aligned}
\tag{4.6}
$$

Further, to address suboptimality (i.e., approximate local-optimality) in the demonstrations, we can relax the stationarity (4.5h) and complementary slackness constraints (4.5f)-(4.5g) and place corresponding penalties into the objective function:

Problem IV.3 (KKT inverse, suboptimal)**.**

$$
\begin{aligned}
\underset{\theta, \boldsymbol{\lambda}_k^j, \boldsymbol{\lambda}_{\neg k}^j, \boldsymbol{\nu}_k^j}{\text{minimize}} \quad & \sum_{j=1}^{N_s} \left(\|\text{stat}(\xi_j^{\text{dem}})\|_1 + \|\text{comp}(\xi_j^{\text{dem}})\|_1 \right) \\
\text{subject to} \quad & (4.5\text{c}) - (4.5\text{e}), \;\; \forall \xi_j^{\text{dem}}, \;\; j = 1, \ldots, N_s
\end{aligned}
\tag{4.7}
$$

where stat(ξ_j^{dem}) *denotes the LHS of Eq.* (4.5h) *and comp*(ξ_j^{dem}) *denotes the concatenated LHSs of Eqs.* (4.5f) *and* (4.5g)*.*

Denote the projection of the feasible set of Problem IV.2 onto Θ as $\mathcal{F}$. We define the set of learned guaranteed safe/unsafe constraint states as $\mathcal{G}_s/\mathcal{G}_{\neg s}$, respectively. For Problem IV.2, a constraint state κ is learned guaranteed safe/unsafe if κ is marked safe/unsafe for all $\theta \in \mathcal{F}$. Formally, we have:

$$\mathcal{G}_s \doteq \bigcap_{\theta \in \mathcal{F}} \{\kappa \mid g(\kappa, \theta) \leq 0\} \tag{4.8}$$

$$\mathcal{G}_{\neg s} \doteq \bigcap_{\theta \in \mathcal{F}} \{\kappa \mid g(\kappa, \theta) > 0\} \tag{4.9}$$

We now formulate variants of Problem IV.2 which are efficiently solvable for specific constraint parameterizations. For legibility, we describe the method assuming $\mathcal{C} = \mathcal{X}$ and $\phi(\cdot)$ is the identity. Due to the bilinearity between decision variables in Problems IV.2 and IV.3 for some parameterizations, we describe exact (Sec. 4.3.2) and relaxed (Sec. 4.3.4) formulations for recovering the unknown parameters.

4.3.2 Unions of offset-parameterized constraints

Consider when Problem IV.1 involves avoiding an unsafe set $\mathcal{A}(\theta)$ described by the union and intersection of offset-parameterized half-spaces (i.e., θ does not multiply κ):

$$\mathcal{A}(\theta) = \bigcup_{m=1}^{N_c} \bigcap_{n=1}^{N_c^m} \{\kappa \mid a_{m,n}^\top \kappa < b_{m,n}(\theta)\} \tag{4.10}$$

This parameterization can represent any arbitrarily-shaped unsafe set if N_c is sufficiently large (i.e., as a union of polytopes) *Tao* (2016), though in practice our method may not be efficient for large N_c. We will often use the specific case of unions of axis-aligned N_c^m-dimensional hyper-rectangles,

$$\mathcal{A}(\theta) = \bigcup_{m=1}^{N_c} \bigcap_{n=1}^{N_c^m} \{\kappa \mid \kappa < \overline{\theta}_n^m, -\kappa < -\underline{\theta}_n^m\}, \tag{4.11}$$

where $\overline{\theta}_n^m / \underline{\theta}_n^m$ are the upper/lower extents of dimension n of box m. We now modify the KKT conditions to handle the "or" constraints in (4.10). Primal feasibility (4.5c) changes to

$$\forall \kappa \in \xi_j^{\text{dem}}, \forall m = 1, ..., N_c, \bigvee_{n=1}^{N_c^m} \Big(a_{m,n}^\top \kappa \geq b_{m,n}(\theta) \Big), \tag{4.12}$$

which can be implemented using the big-M formulation *Bertsimas and Tsitsiklis* (1997):

$$\forall \kappa, \forall m, \mathbf{A}_m^\top \kappa \geq \mathbf{b}_m(\theta) - M(\mathbf{1} - \mathbf{z}_m^{j,\kappa}),\ \textstyle\sum_{n=1}^{N_c^m} \mathbf{z}_m^{j,\kappa}(n) \geq 1, \tag{4.13}$$

where M is a large positive number, $\mathbf{A}_m \in \mathbb{R}^{N_c^m \times |\kappa|}$ and $\mathbf{b}_m \in \mathbb{R}^{N_c^m}$ are the vertical concatenation of $a_{m,n}$ and $b_{m,n}$ for all n, $\mathbf{z}_m^{j,\kappa} \in \{0,1\}^{N_c^m}$ are binary variables encoding that at least one half-space constraint must hold, and $\mathbf{z}_m^{j,\kappa}(n)$ is the nth entry of $\mathbf{z}_m^{j,\kappa}$. For demonstration ξ_j^{dem} to be locally-optimal, we know that for each $\kappa \in \xi_j^{\text{dem}}$, the complementary slackness condition, $\lambda_{(m,n),\neg k}^{j,\kappa}(a_{m,n}^\top \kappa - b_{m,n}(\theta)) = 0$, must hold for at least one n and for all m in Eq. (4.12). Furthermore, in the stationarity condition (4.5h), $\lambda_{(m,n),\neg k}^{j,\kappa} \nabla_\kappa g_{(m,n),\neg k}(\xi_j^{\text{loc}}, \theta)$ terms should only be included for (m,n) pairs where the complementary slackness condition is enforced. Thus, we can enforce

that $\lambda^{j,\kappa}_{(m,n),\neg k}(a^\top_{m,n}\kappa - b_{m,n}(\theta)) = 0$ holds for all $\kappa \in \xi^{\text{dem}}_j$, for all $m \in \{1,...,N_c\}$, and for some $n \in \{1,...,N^m_c\}$ by writing:

$$\begin{aligned} \forall \kappa, m, \begin{bmatrix} \boldsymbol{\lambda}^{j,\kappa}_{m,\neg k} \\ \mathbf{A}^\top_m \kappa - \mathbf{b}_m(\theta) \end{bmatrix} \leq M \begin{bmatrix} \mathbf{z}^{j,\kappa}_{m,1} \\ \mathbf{z}^{j,\kappa}_{m,2} \end{bmatrix}, \quad \mathbf{z}^{j,\kappa}_{m,1} + \mathbf{z}^{j,\kappa}_{m,2} \leq 2 - \mathbf{q}^{j,\kappa}_m, \\ \textstyle\sum_{n=1}^{N^m_c} \mathbf{q}^{j,\kappa}_m(n) \geq 1, \quad \mathbf{z}^{j,\kappa}_{m,1},\ \mathbf{z}^{j,\kappa}_{m,2},\ \mathbf{q}^{j,\kappa}_m \in \{0,1\}^{N^m_c} \end{aligned} \tag{4.14}$$

together with (4.5d) and (4.5e), where we use a big-M formulation with binary variables z (encoding the complementary slackness condition) and q (encoding if the complementary slackness condition is being enforced). We have denoted $\boldsymbol{\eta}^{j,\kappa}_m \doteq [\eta^{j,\kappa}_{(m,1)}, \ldots, \eta^{j,\kappa}_{(m,N^m_c)}]^\top$ for $\eta \in \{\lambda, z, q\}$. Next, we modify line 2 of constraint (4.5h) to enforce:

$$\sum_{m=1}^{N_c} \left[(\boldsymbol{\lambda}^{j,\kappa}_{m,\neg k} \odot \mathbf{q}^{j,\kappa}_m)^\top \nabla_\kappa (\mathbf{b}_n(\theta) - \mathbf{A}^\top_m \kappa) \right] \doteq \sum_{m=1}^{N_c} {\mathbf{q}^{j,\kappa}_m}^\top \mathbf{L}^{j,\kappa}_m \tag{4.15}$$

for all $\kappa \in \xi^{\text{dem}}_j$, where the (i,n)-th entry of $\mathbf{L}^{j,\kappa}_m \in \mathbb{R}^{N^m_c \times |\kappa|}$, $\mathbf{L}^{j,\kappa}_m(i,n)$, refers to $\lambda^{j,\kappa}_{(m,n),\neg k} \nabla_{\kappa(i)}(b_{m,n}(\theta) - a^\top_{m,n}\kappa)$. Note that λ (continuous variables) and q (binary variables) are bilinear ($\nabla_\kappa(b_{m,n}(\theta) - a^\top_{m,n}\kappa)$ has no decision variables as θ does not multiply κ). By assuming bounds $\underline{M} \leq \mathbf{L}^{j,\kappa}_m(i,n) \leq \overline{M}$, this can be reformulated **exactly** in a linear fashion (i.e., *linearized*) *Liberti and Pantelides* (2006) by replacing each bilinear product $\mathbf{q}^{j,\kappa}_m(n)\mathbf{L}^{j,\kappa}_m(i,n)$ in (4.15) with slack variables $\mathbf{R}^{j,\kappa}_m(i,n)$ and adding constraints (where $\tilde{\mathbf{q}}^{j,\kappa}_m(n) \doteq (1 - \mathbf{q}^{j,\kappa}_m(n)$ for short):

$$\begin{aligned} \min(0, \underline{M}) \leq \mathbf{R}^{j,\kappa}_m(i,n) \leq \overline{M} \\ \underline{M}\mathbf{q}^{j,\kappa}_m(n) \leq \mathbf{R}^{j,\kappa}_m(i,n) \leq \overline{M}\mathbf{q}^{j,\kappa}_m(n) \\ \mathbf{L}^{j,\kappa}_m(i,n) - \tilde{\mathbf{q}}^{j,\kappa}_m(n)\overline{M} \leq \mathbf{R}^{j,\kappa}_m(i,n) \leq \mathbf{L}^{j,\kappa}_m(i,n) - \tilde{\mathbf{q}}^{j,\kappa}_m(n)\underline{M} \\ \mathbf{R}^{j,\kappa}_m(i,n) \leq \mathbf{L}^{j,\kappa}_m(i,n) + \tilde{\mathbf{q}}^{j,\kappa}_m(n)\overline{M} \end{aligned} \tag{4.16}$$

Finally, let $\mathbf{R}^j_m$ / $\mathbf{L}^j_m$ be the horizontal concatenation of $\mathbf{R}^{j,\kappa}_m$ / $\mathbf{L}^{j,\kappa}_m$, for all $\kappa \in \xi^{\text{dem}}_j$. We can now pose the full problem:

Problem IV.4 (KKT inverse, unions)**.**

$$\begin{aligned} \text{find} \quad & \theta, \boldsymbol{\lambda}^{j,\kappa}_k, \boldsymbol{\lambda}^{j,\kappa}_{\neg k}, \boldsymbol{\nu}^j_k, \mathbf{R}^j_m, \mathbf{L}^j_m, \mathbf{q}^{j,\kappa}_m, \mathbf{z}^{j,\kappa}_{m,1}, \mathbf{z}^{j,\kappa}_{m,2}, \\ & \forall \kappa \in \xi^{\text{dem}}_j, m = 1,...,N_c, j = 1,...,N_s \\ \text{s.t.} \quad & \texttt{-------- Primal feasibility --------} \\ & \text{Equation (4.13)},\ j = 1,...,N_s \\ & \texttt{--- Lagrange mult. nonnegativity ---} \\ & \boldsymbol{\lambda}^{j,\kappa}_k \geq \mathbf{0},\ \forall \kappa \in \xi^{\text{dem}}_j,\ j = 1,\ldots,N_s \\ & \boldsymbol{\lambda}^{j,\kappa}_{m,\neg k} \geq \mathbf{0},\ \forall \kappa \in \xi^{\text{dem}}_j,\ m = 1,...,N_c,\ j = 1,...,N_s \\ & \texttt{------ Complementary slackness -----} \\ & \text{Equation (4.14)},\ j = 1,\ldots,N_s \\ & \texttt{---------- Stationarity ----------} \\ & \nabla_{\xi_{xu}} c_\Pi(\xi^{\text{loc}}_j) + {\boldsymbol{\lambda}^j_{\neg k}}^\top \nabla_{\xi_{xu}} \mathbf{g}_k(\xi^{\text{loc}}_j) + \textstyle\sum_{m=1}^{N_c} \mathbf{1}^\top_{N^m_c} \mathbf{R}^j_m \\ & \quad + {\boldsymbol{\nu}^j_{\neg k}}^\top \nabla_{\xi_{xu}} \mathbf{h}_k(\xi^{\text{loc}}_j) = 0,\ j = 1,...,N_s \\ & \text{Equation (4.16)},\ \forall \kappa \in \xi^{\text{dem}}_j, m = 1,...,N_c, \\ & \quad n = 1,...,N^m_c, j = 1,...,N_s \end{aligned} \tag{4.17}$$

4.3.3 Extraction of safe and unsafe states

Before moving onto affine parameterizations, we first detail how to check guaranteed safeness/unsafeness (as defined in (4.8)-(4.9)). One can check if a constraint state $\kappa \in \mathcal{G}_s$ or $\kappa \in \mathcal{G}_{\neg s}$ by adding a constraint $g(\kappa, \theta) > 0$ or $g(\kappa, \theta) \leq 0$ to Problem IV.2 and checking feasibility of the resulting program:

Problem IV.5 (Query if κ is guaranteed safe **OR** unsafe)**.**

$$\begin{array}{ll} \text{find} & \theta, \boldsymbol{\lambda}_k^j, \boldsymbol{\lambda}_{\neg k}^j, \boldsymbol{\nu}_k^j, \quad j = 1, ..., N_s \\ \text{s.t.} & \{\text{KKT}(\xi_j^{\text{loc}})\}_{j=1}^{N_s}, \quad g(\kappa, \theta) > 0 \textbf{ OR } g(\kappa, \theta) \leq 0 \end{array} \tag{4.18}$$

If Problem IV.5 is infeasible, then $\kappa \in \mathcal{G}_s$ or $\kappa \in \mathcal{G}_{\neg s}$. Solving this problem is akin to querying an oracle about the safety of κ. The oracle can return that κ is guaranteed safe (program infeasible after forcing κ to be unsafe), guaranteed unsafe (program infeasible after forcing κ to be safe), or unsure (program is feasible despite forcing κ to be safe or unsafe).

Since the constraint space is continuous, it is not possible to check via enumeration if each $\kappa \in \mathcal{G}_{\neg s}$ or $\kappa \in \mathcal{G}_s$. To address this, we can check the neighborhood of a constraint state κ_{query} for membership in $\mathcal{G}_{\neg s}$ by solving the following:

Problem IV.6 (Volume extraction)**.**

$$\begin{array}{ll} \underset{\varepsilon, \kappa_{\text{near}}, \theta, \boldsymbol{\lambda}_k^j, \boldsymbol{\lambda}_{\neg k}^j, \boldsymbol{\nu}_k^j}{\text{minimize}} & \varepsilon \\ \text{subject to} & \{\text{KKT}(\xi_j^{\text{loc}})\}_{j=1}^{N_s} \\ & \|\kappa_{\text{near}} - \kappa_{\text{query}}\|_\infty \leq \varepsilon \\ & g(\kappa_{\text{near}}, \theta) > 0 \end{array} \tag{4.19}$$

Intuitively, Problem IV.6 finds the largest box centered at κ_{query} contained within $\mathcal{G}_s$. An analogous problem can also be posed to recover the largest hypercube centered at κ_{query} contained within $\mathcal{G}_{\neg s}$. For some common parameterizations (axis-aligned hyper-rectangles, convex sets), subsets of $\mathcal{G}_s$ and $\mathcal{G}_{\neg s}$ can be even more efficiently recovered by performing line searches or taking convex hulls of guaranteed safe/unsafe states, details of which are in Appendix B of *Chou et al.* (2019). Volumes of safe/unsafe space can thus be produced by repeatedly solving Problem IV.6 for different κ_{query}.

4.3.4 KKT relaxation for unions of affine constraints

Now, consider when Problem IV.1 involves avoiding an unsafe set $\mathcal{A}(\theta)$ described by a union of affine constraints:

$$\mathcal{A}(\theta) = \bigcup_{i=1}^{N_c} \{\kappa \mid g_{i,\neg k}(\kappa, \theta) > 0\} \tag{4.20}$$

where $g_{i,\neg k}(\kappa, \theta)$ is an affine function of θ for fixed κ. Unlike in Sec. 4.3.2, formulating the recovery problem like Problem IV.4 yields trilinearity in the stationarity

condition between continuous variables θ and λ and binary variables q, since for the affine case, $\nabla_{\xi_{xu}} g_{i,\neg k}(\cdot, \theta)$ remains a function of θ. As the product of two continuous decision variables cannot be linearized exactly, one must solve a MINLP to recover θ in this case, which can be inefficient. However, a relaxation which enables querying of guaranteed safeness/unsafeness via Problem IV.5 can be formulated as a MILP. For legibility, we present the $N_c = 1$ case, where there is only one affine constraint (and hence the binary variables q seen in Problem IV.4 are all set to 1 and can thus be dropped). Each bilinear term $\lambda^{j,\kappa}_{1,\neg k} \nabla_\kappa g_{1,\neg k}(\kappa, \theta)$ is replaced with $l^{j,\kappa}_1 z^{j,\kappa}_{1,1}$, where $l^{j,\kappa}_1$ is a variable which represents the bilinear term and $z^{j,\kappa}_{1,1}$ is an indicator variable encoding that if $z^{j,\kappa}_{1,1}$ is 0, then $\lambda^{j,\kappa}_{1,\neg k}$ must be 0. Hence, by linearizing the bilinear term as such, there is no relaxation gap when the Lagrange multipliers are zero; the only loss is when the Lagrange multipliers are non-zero (i.e., when the demonstration touches the constraint boundaries). In this case, coupling between λ and θ is lost by introducing the $l^{j,\kappa}_1$ variables. We further linearize $l^{j,\kappa}_1 z^{j,\kappa}_{1,1}$ (product of continuous, binary variables) with the same procedure in Sec. 4.3.2 by again introducing slack variables $r^{j,\kappa}_1$ and constraining them accordingly with (4.16), where the $q^{j,\kappa}_{m,n}$ are replaced with $z^{j,\kappa}_{1,1}$. Putting things together, we can write the following relaxed constraint recovery problem for $N_c = 1$:

Problem IV.7 (KKT relaxation, affine)**.**

$$
\begin{aligned}
\textit{find} \quad & \theta, \boldsymbol{\lambda}^{j,\kappa}_k, \boldsymbol{\lambda}^{j,\kappa}_{\neg k}, \boldsymbol{\nu}^j_k, \mathbf{r}^j_1, \boldsymbol{\ell}^j_1, z^{j,\kappa}_{1,1}, z^{j,\kappa}_{1,2}, \ \forall \kappa \in \xi^{dem}_j, \ j = 1, ..., N_s \\
\textit{s.t.} \quad & \texttt{-------- Primal feasibility --------} \\
& g_{1,\neg k}(\kappa, \theta) \leq 0, \ \forall \kappa \in \xi^{dem}_j, \ j = 1, \ldots, N_s \\
& \texttt{--- Lagrange mult. nonnegativity ---} \\
& \boldsymbol{\lambda}^{j,\kappa}_k \geq \mathbf{0}, \ \forall \kappa \in \xi^{dem}_j, \ j = 1, \ldots, N_s \\
& \lambda^{j,\kappa}_{1,\neg k} \geq 0, \ \forall \kappa \in \xi^{dem}_j, j = 1, \ldots, N_s \\
& \texttt{------ Complementary slackness -----} \\
& \begin{bmatrix} \lambda^{j,\kappa}_{1,\neg k} \\ -g_{1,\neg k}(\kappa, \theta) \end{bmatrix} \leq M \begin{bmatrix} z^{j,\kappa}_{1,1} \\ z^{j,\kappa}_{1,2} \end{bmatrix}, \ z^{j,\kappa}_{1,1} + z^{j,\kappa}_{1,2} \leq 1, \\
& \quad \forall \kappa \in \xi^{dem}_j, j = 1, \ldots, N_s \\
& \texttt{---------- Stationarity ----------} \\
& \nabla_{\xi_{xu}} c_\Pi(\xi^{loc}_j) + \boldsymbol{\lambda}^j_k \nabla_{\xi_{xu}} \mathbf{g}_k(\xi^{loc}_j) + \mathbf{r}^j_1 \\
& \quad + \boldsymbol{\nu}^j_k \nabla_{\xi_{xu}} \mathbf{h}_k(\xi^{loc}_j) = 0, \ j = 1, \ldots, N_s \\
& min(0, \underline{M})\mathbf{1} \leq \mathbf{r}^j_1 \leq \overline{M}\mathbf{1}, \quad \underline{M}\mathbf{z}^j_{1,1} \leq \mathbf{r}^j_1 \leq \overline{M}\mathbf{z}^j_{1,1}, \\
& \quad \boldsymbol{\ell}^j_1 - (\mathbf{1} - \mathbf{z}^j_{1,1})\overline{M} \leq \mathbf{r}^j_1 \leq \boldsymbol{\ell}^j_1 - (\mathbf{1} - \mathbf{z}^j_{1,1})\underline{M}, \\
& \quad \mathbf{r}^j_1 \leq \boldsymbol{\ell}^j_1 + (\mathbf{1} - \mathbf{z}^j_{1,1})\overline{M}, \quad j = 1, \ldots, N_s
\end{aligned}
\tag{4.21}
$$

where $\mathbf{r}^j_1$, $\mathbf{z}^j_{1,1}$, $\boldsymbol{\ell}^j_1$ denote horizontal concatenation of $r^{j,\kappa}_1$, $z^{j,\kappa}_{1,1}$, $l^{j,\kappa}_1$ for all $\kappa \in \xi^{\text{dem}}_j$. The case if the constraint is a union of affine constraints yields quadrilinearity and can be handled similarly, requiring one extra step to linearize the products of binary variables $q^{j,\kappa}_m$ and $z^{j,\kappa}_{1,1}$, which can be done exactly.

While Problem IV.7 cannot recover the constraint parameter θ directly, one can still check if a constraint state is guaranteed safe/unsafe using Problem IV.5 (see Theorem IV.12 for reasoning).

4.3.5 Unknown constraint parameterization

In many applications, we may not know a constraint parameterization a priori. However, complex unsafe/safe sets can often be approximated as the union of many simple unsafe/safe sets. Thus, we adapt the method in Chapter III for incrementally growing a parameterization based on the complexity of the provided demonstrations. More precisely, suppose the true parameterization $g(\kappa, \theta)$ of the unsafe set $\mathcal{A}(\theta) = \{\kappa \mid g(\kappa, \theta) > 0\}$ is unknown but can be exactly/approximately written as the union of N^* simple sets $\mathcal{A}(\theta) \approxeq \bigcup_{i=1}^{N^*} \{\kappa \mid g_s(\kappa, \theta_i) > 0\} \doteq \bigcup_{i=1}^{N^*} \mathcal{A}(\theta_i)$. Each simple set $\mathcal{A}(\theta_i)$ has a known parameterization $g_s(\cdot, \cdot)$ but N^*, the minimum number of simple sets needed to reconstruct $\mathcal{A}$, is unknown. We can estimate a lower bound on N^*, $\underline{N}$, by incrementally adding simple sets until Problem IV.2 is feasible (i.e., there exists a sufficiently complex constraint which can satisfy the demonstrations' KKT conditions). Issues with conservativeness of the recovered constraint when $\underline{N} < N^*$ are discussed in III and also hold here, which we omit for brevity.

4.3.6 Handling cost function uncertainty

We now extend the KKT conditions presented in (4.5) and Problems IV.2 and IV.3 to learn constraints with parametric uncertainty in the cost function (i.e., if γ in Problem IV.1 is unknown). To address this, the first term in the stationarity condition (4.5h) must be changed to $\nabla_{\xi_{xu}} c_\Pi(\xi_j^{\text{loc}}, \gamma)$. Then, if $c_\Pi(\cdot, \gamma)$ is affine in γ, γ can be found using a MILP.

Querying/volume extraction holds just as before; the only difference is that γ is now a decision variable in Problem IV.5/IV.6. Note we are extracting constraint states that are guaranteed safe/unsafe for all possible cost parameters; that is, we are extracting safe/unsafe sets that are robust to cost uncertainty.

We summarize what we can solve for when using various parameterizations. For the exact cases, we can solve for θ/γ, but when relaxing, we can only solve for $\mathcal{S}/\mathcal{A}$ via queries. Note the constraint/cost can be nonlinear in κ without inducing relaxation, though it precludes usage of Problem IV.6 (as κ is a decision variable in the latter, but not the former):

Constraint param.	Cost param.	Recover θ, γ?	$\mathcal{G}_s/\mathcal{G}_{\neg s}$
θ: form of (4.10); κ: affine	γ: affine; κ: nonlin.	Yes: Prob. IV.4	Prob. IV.5/IV.6
θ: form of (4.10); κ: nonlin.	γ: affine; κ: nonlin.	Yes: Prob. IV.4	Prob. IV.5
θ: form of (4.20); κ: nonlin.	γ: affine; κ: nonlin.	No	Prob. IV.5

This only describes what we can solve for; the actual accuracy of the recovered θ/γ and the size of the recovered $\mathcal{G}_s/\mathcal{G}_{\neg s}$ depends on how informative the demonstrations are, i.e., the demonstrations should interact with the constraint.

4.3.7 Applications to safe planning

As learned constraints can be reused for novel tasks with the same safety requirements, we end this section by describing how our method can be used within a planner to guarantee the safety of trajectories planned for such tasks. Recall that Problems

IV.5 and IV.6 can be used to query if a constraint state κ or a region around κ is guaranteed safe/unsafe. The planner can use this information by either:

- Extracting an *explicit* representation of the constraint by repeatedly solving Problem IV.6 for different κ_{query} to cover $\mathcal{S}$ and $\mathcal{A}$. Denote these extracted sets as $\hat{\mathcal{S}} \subseteq \mathcal{S}$ and $\hat{\mathcal{A}} \subseteq \mathcal{A}$ (the conservativeness of our method is proved in Sec. 4.4.1). Then, $\hat{\mathcal{S}}$ can be passed to a planner and quickly used for constraint/collision checking via set-containment checks. A planned trajectory is *guaranteed safe* if each state on it lies in $\hat{\mathcal{S}}$, since $\hat{\mathcal{S}}$ is contained in true safe set $\mathcal{S}$. If $\hat{\mathcal{S}}$ is small and the planner cannot find a feasible trajectory, we can at least guarantee that a trajectory is *not definitely unsafe* it it does not intersect with $\hat{\mathcal{A}}$, as $\hat{\mathcal{A}}$ is contained in the true unsafe set $\mathcal{A}$.
- Extracting an *implicit* representation of the constraint by solving Problem IV.5 as needed by the planner. This may be less computationally efficient than the explicit case, but we demonstrate in Sec. 4.5.3 that we still achieve reasonable planning times for a 7-DOF arm.

4.4 Theoretical Analysis

In this section, we prove that our method provides a conservative estimate of the guaranteed learned safe/unsafe sets $\mathcal{G}_s, \mathcal{G}_{\neg s}$ (Sec. 4.4.1) and prove learnability results using locally-optimal demonstrations (Sec. 4.4.2).

4.4.1 Conservativeness

Definition IV.8 (Implied unsafe/safe set)**.** For some set $\mathcal{B} \subseteq \Theta$, let $I_{\neg s}(\mathcal{B}) \doteq \bigcap_{\theta \in \mathcal{B}} \{x \mid g(x, \theta) > 0\}$ be the set of states implied unsafe by restricting the parameter set to $\mathcal{B}$, i.e., $I_{\neg s}(\mathcal{B})$ is the set of states that all $\theta \in \mathcal{B}$ mark as unsafe. Similarly, let $I_s(\mathcal{B}) \doteq \bigcap_{\theta \in \mathcal{B}} \{x \mid g(x, \theta) \leq 0\}$ be the set of states implied safe by restricting the parameter set to $\mathcal{B}$.

We further introduce the following lemma:

Lemma IV.9 (Lemma C.1 in *Chou et al.* (2019))**.** *Suppose $\mathcal{B} \subseteq \hat{\mathcal{B}}$, for some other set $\hat{\mathcal{B}}$. Then, $I_{\neg s}(\hat{\mathcal{B}}) \subseteq I_{\neg s}(\mathcal{B})$ and $I_s(\hat{\mathcal{B}}) \subseteq I_s(\mathcal{B})$.*

Theorem IV.10 (Conservativeness of Problem IV.2)**.** *Suppose the constraint parameterization $g(x, \theta)$ is known exactly. Then, extracting $\mathcal{G}_s$ and $\mathcal{G}_{\neg s}$ (as defined in* (4.8) *and* (4.9)*, respectively) from the feasible set of Problem IV.2 projected onto Θ (denoted as $\mathcal{F}$) returns $\mathcal{G}_{\neg s} \subseteq \mathcal{A}$ and $\mathcal{G}_s \subseteq \mathcal{S}$.*

Proof. We first prove that $\mathcal{G}_{\neg s} \subseteq \mathcal{A}$. Suppose that there exists $x \in \mathcal{G}_{\neg s}$ such that $x \notin \mathcal{A}$. Then by definition, for all $\theta \in \mathcal{F}$, $g(x, \theta) > 0$. However, we know that all locally-optimal demonstrations satisfy the KKT conditions with respect to the true parameter θ^*; hence, $\theta^* \in \mathcal{F}$. Then, $x \in \mathcal{A}(\theta^*)$. Contradiction. Similar logic holds for proving that $\mathcal{G}_s \subseteq \mathcal{S}$. Suppose that there exists $x \in \mathcal{G}_s$ such that $x \notin \mathcal{S}$. Then

by definition, for all $\theta \in \mathcal{F}$, $g(x, \theta) \leq 0$. However, we know that all locally-optimal demonstrations satisfy the KKT conditions with respect to the true parameter θ^*; hence, $\theta^* \in \mathcal{F}$. Then, $x \in \mathcal{S}(\theta^*)$. Contradiction. □

Remark IV.11. *Unfortunately, it is difficult to guarantee conservativeness when using suboptimal demonstrations (solving Problem IV.3), as the relationship between cost suboptimality and KKT violation is generally unknown. However, we note in practice that the $\mathcal{G}_s, \mathcal{G}_{\neg s}$ recovered using suboptimal demonstrations still tend to be conservative (see Sec. 4.5.2).*

Theorem IV.12 (Conservativeness of Problem IV.7)**.** *Suppose the constraint parameterization $g(x, \theta)$ is known exactly. Then, extracting $\mathcal{G}_s$ and $\mathcal{G}_{\neg s}$ (as defined in* (4.8) *and* (4.9)*, respectively) from the feasible set of Problem IV.7 (denoted as $\mathcal{F}$) returns $\mathcal{G}_{\neg s} \subseteq \mathcal{A}$ and $\mathcal{G}_s \subseteq \mathcal{S}$.*

Proof. Denote the Θ-projected feasible set of the original unrelaxed problem (i.e., variables r_i are not introduced and the bilinear terms between θ and λ remain) as $\mathcal{F}_{\text{MINLP}}$ and the Θ-projected feasible set of Problem IV.7 as $\mathcal{F}$. Using the logic in Theorem IV.10, extracting $\mathcal{G}_s$ and $\mathcal{G}_{\neg s}$ from $\mathcal{F}_{\text{MINLP}}$ yields $\mathcal{G}_s \subseteq \mathcal{S}$ and $\mathcal{G}_{\neg s} \subseteq \mathcal{A}$ (since $\mathcal{F}_{\text{MINLP}}$ is the true feasible set, like assumed in Theorem IV.10). Furthermore, $\mathcal{F}_{\text{MINLP}} \subseteq \mathcal{F}$, since relaxing the bilinear terms to linear terms in Problem IV.7 expands the feasible set compared to the unrelaxed problem. By definition, $\mathcal{G}_s = I_s(\mathcal{F}_{\text{MINLP}})$ and $\mathcal{G}_{\neg s} = I_{\neg s}(\mathcal{F}_{\text{MINLP}})$, and via Lemma IV.9, $I_s(\mathcal{F}) \subseteq I_s(\mathcal{F}_{\text{MINLP}})$ and $I_{\neg s}(\mathcal{F}) \subseteq I_{\neg s}(\mathcal{F}_{\text{MINLP}})$. Hence, $I_s(\mathcal{F}) \subseteq \mathcal{S}$ and $I_{\neg s}(\mathcal{F}) \subseteq \mathcal{A}$. □

Remark IV.13. *For brevity, we omit conditions on $M, \underline{M}, \overline{M}$ for conservativeness; it is well-known that this is achieved by choosing the big-M constants to be sufficiently large Bertsimas and Tsitsiklis (1997).*

4.4.2 Global vs local learnability

Definition IV.14 (Local learnability)**.** A state $x \in \mathcal{A}$ is locally learnable if there exists **any** set of N_s locally-optimal demonstrations, where N_s may be infinite, such that $x \in \mathcal{I}_{\neg s}(\mathcal{F})$, where $\mathcal{F}$ is the Θ-projected feasible set of Problem IV.2. We also define the *locally learnable* set of unsafe states $\mathcal{G}_{\neg s}^{\text{loc},*}$ as the union of all locally learnable states.

Definition IV.15 (Global learnability)**.** A state $x \in \mathcal{A}$ is globally learnable if there exists **any** set of N_s globally-optimal demonstrations and $N_{\neg s}$ sampled strictly lower-cost (and hence unsafe) trajectories, where N_s and $N_{\neg s}$ may be infinite, such that $x \in \mathcal{I}(\mathcal{F}_{\text{glo}})$, where $\mathcal{F}_{\text{glo}}$ is the feasible set of Problem 2 in *Chou et al.* (2019) (which recovers a constraint consistent with the demonstrations and sampled unsafe trajectories). Accordingly, we define the *globally learnable* set of unsafe states $\mathcal{G}_{\neg s}^{\text{glo},*}$ as the union of all globally learnable states.

Note that a safe state $x_s \in \mathcal{S}$ can always be learned guaranteed safe, as there always exists a safe globally-optimal or locally-optimal demonstration passing through x_s. Armed with these definitions, we show the following:

Theorem IV.16 (Global vs local)**.** *Suppose the initial constraint parameter set* Θ *is identical for both the local and global problems. Then,* $\mathcal{G}_{\neg s}^{loc,*} \subseteq \mathcal{G}_{\neg s}^{glo,*}$.

Proof. Any globally-optimal demonstration must also satisfy the KKT conditions, as it is also locally-optimal. Further conditions (in the form of lower-cost trajectories being infeasible) must be imposed on a constraint parameter for it to be globally-optimal. Hence, $\mathcal{F}_{\text{glo}} \subseteq \mathcal{F}$. By Lemma IV.9, $\mathcal{I}(\mathcal{F}) \subseteq \mathcal{I}(\mathcal{F}_{\text{glo}})$, and thus $\mathcal{G}_{\neg s}^{\text{loc},*} \subseteq \mathcal{G}_{\neg s}^{\text{glo},*}$. □

Note that Theorem IV.16 holds in the limit of having sampled **all** unsafe trajectories. In practice, the sampling is nowhere near complete, especially for nonlinear dynamics. We see in these cases (Sec. 4.5.3) that our KKT-based method learns more compared to sampling-based techniques. Finally, we note that cost function uncertainty can only decrease learnability, as it enlarges the feasible set of Problem IV.2.

4.5 Results

We show our method, first on 2D examples (Sec. 4.5.1) for intuition, and then on high-dimensional 7-DOF arm (Sec. 4.5.2) and quadrotor (Sec. 4.5.3) constraint-learning problems (see the accompanying video for experiment visualizations). All computation times are recorded on a laptop with a 3.1 GHz Intel Core i7 processor and 16 GB RAM.

4.5.1 2D examples

Global vs. local: Assuming global demonstration optimality can enlarge $\mathcal{G}_s/\mathcal{G}_{\neg s}$ compared to assuming local optimality (Theorem IV.16). In this example, we show some common differences in the learned constraints when assuming global/local optimality. Consider a 2D kinematic system $\chi_{t+1} = \chi_t + u_t$, $\chi = [x, y]^\top$, $\|u_t\| \leq 1$ avoiding the pink obstacle in Fig. 4.1. We use an axis-aligned box constraint parameterization. In Fig. 4.1 (left), by assuming the demonstrations (cyan, green) are globally-optimal and sampling lower-cost trajectories (the middle state on each trajectory is plotted in red), the hatched area is implied guaranteed unsafe, as any axis-aligned box containing the sampled unsafe states (in red) must also contain the hatched area. In contrast, assuming local optimality gives us zero volume learned guaranteed safe/unsafe, as a measure-zero horizontal line obstacle (orange dashed) can make the demonstrations locally-optimal: as the line supports the middle state on each demonstration, the cost cannot be locally improved. In Fig. 4.1, center, we show a case where there is no gap in learnability: without assuming a parameterization, the demonstrations can be explained by two horizontal line obstacles, but together with the box parameterization, we recover $\mathcal{G}_s = \mathcal{S}$ and $\mathcal{G}_{\neg s} = \mathcal{A}$. Fig. 4.1 (right) shows that assuming global optimality may result in non-conservative constraint recovery (e.g., if the dotted red line were a sampled unsafe trajectory), while a horizontal line obstacle (orange dashed line) can explain local optimality of the demonstration, yielding conservative constraint recovery.

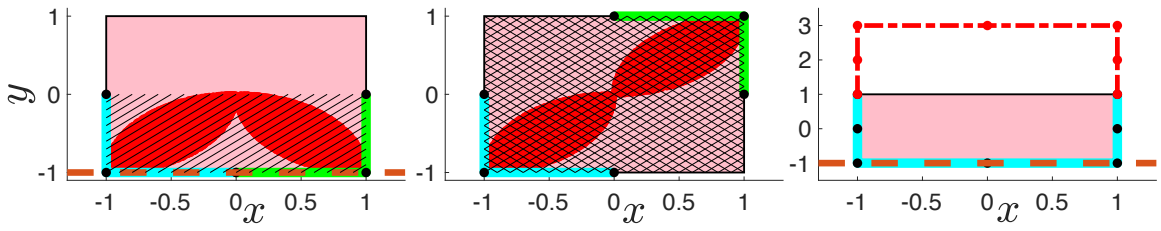


Figure 4.1: Left: local learns less than global. Center: local learns the same as global. Right: global recovers non-conservative solution. Red: sampled unsafe trajectories. Pink: true constraint. Green/cyan: demonstrations.

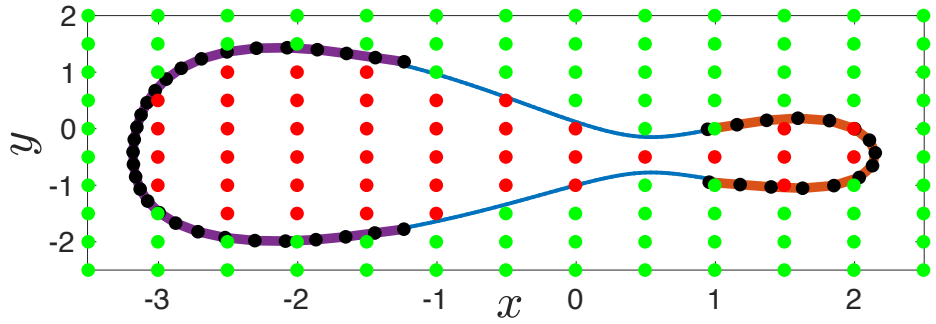


Figure 4.2: Nonlinear constraint. Blue: true constraint boundary. Red/green states: learned in $\mathcal{G}_{\neg s}/\mathcal{G}_s$. Purple/orange: two demonstrations.

Effects of cost uncertainty: We show that learnability under cost uncertainty is more related to the possible behaviors that a cost uncertainty set can represent, rather than the actual size of the cost parameter space. For the demonstrations/constraint in the center plot of Fig. 4.1, consider the following cost uncertainty sets: A) $c(\xi) = \sum_{t=1}^{T-1} \gamma_1(x_{t+1} - x_t)^2 + \gamma_2(y_{t+1} - y_t)^2$, where $\gamma_i \in [-5, 5], i = 1, 2$ and B) $c(\xi) = \sum_{k=1}^{10} \sum_{t=1}^{T-1} \left(\gamma_{1,k}(x_{t+1} - x_t)^{2k} + \gamma_{2,k}(y_{t+1} - y_t)^{2k}\right)$, where $\gamma_{i,k} \in [0.001, 5]$ for all i, k. While Set A has a much smaller parameter space compared to Set B (2 vs 20 parameters), allowing γ_1, γ_2 to take negative values enables the case where the demonstrator wants to maximize path length (i.e., set $\gamma_1 = \gamma_2 = -1$). For fixed start/goal states and control constraints, the observed demonstrations are actually locally-optimal with respect to a cost function which maximizes path length in an environment with no box state space constraint. Hence, for Set A, our method returns $\mathcal{G}_s = \mathcal{G}_{\neg s} = \emptyset$. In contrast, while Set B has a much larger parameter space, the range of allowable behaviors is small (all cost terms must penalize path length). Thus, despite the large cost parameter space, $\mathcal{G}_s = \mathcal{S}$ and $\mathcal{G}_{\neg s} = \mathcal{A}$.

Nonlinear constraint: We emphasize that while our method requires an affine parameterization in the *constraint parameters*, constraints that are nonlinear in the *state* can still be learned. Consider a parameterization $g_{1,\neg k}(\kappa, \theta) = 2(x^4 + y^4) - 5(x^3 + y^3) + 5(x-1)^3 + 5(y+1)^3 - \theta$, which yields a highly nonlinear state space constraint. With two demonstrations for $\theta = 2$ (see Fig. 4.2), $\mathcal{G}_s = \mathcal{S}$ and $\mathcal{G}_{\neg s} = \mathcal{A}$.

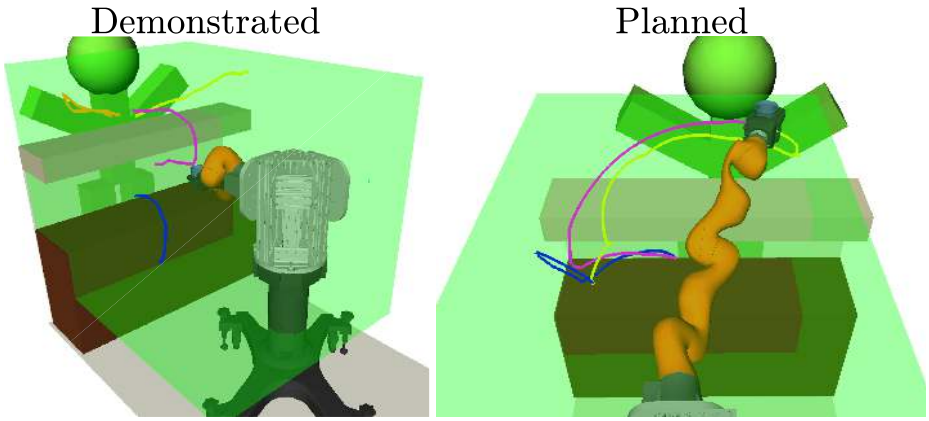


Figure 4.3: Left: demonstrations for bartender example. Right: novel trajectories planned with learned constraint.

4.5.2 7-DOF arm

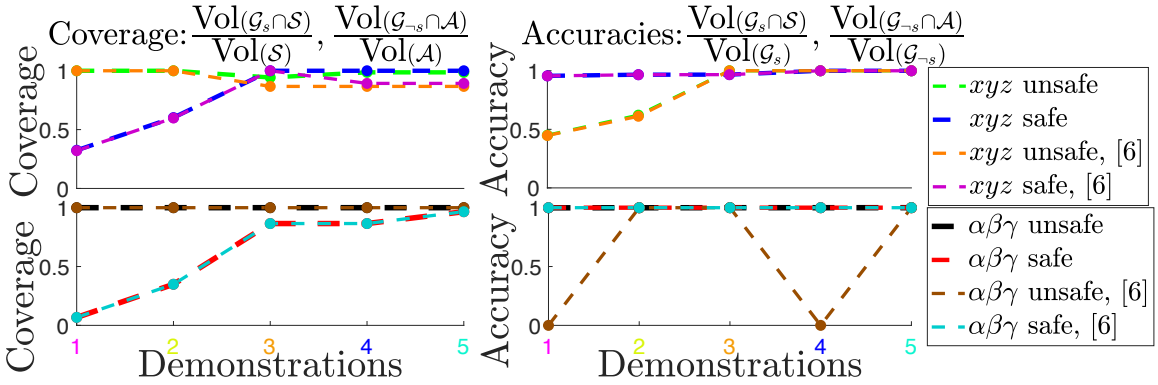


Figure 4.4: Arm bartender statistics; x-axis color-coded with demos in Fig. 4.3.

Robot bartender: Consider a 7-DOF Kuka iiwa robot bartender which must deliver a drink from the bar cabinet (Fig. 4.3, brown box) or from another bartender to a customer at the counter (Fig. 4.3, gray box). To do this, the arm must satisfy an end-effector pose constraint to avoid spilling the drink, and the swept volume of the arm must not collide with the bar furniture while satisfying proxemics constraints (Fig. 4.3, green box) with respect to the customer. We use a kinematic model of the arm, $j^i_{t+1} = j^i_t + u^i_t, i = 1, ..., 7$, where $\|u_t\|_2^2 \leq 0.8$ for all t. Five suboptimal human demonstrations, optimizing joint-space path length $c(\xi) = \sum_{i=1}^{T-1} \|j_{t+1} - j_t\|_2^2$, are captured in a virtual reality environment using an HTC Vive. The proxemics constraints encoded in the demonstrations (Fig. 4.3, left) disallow the arm from getting too close to the customer and from making large sweeping motions from the left, right, and particularly the top, as the customer can perceive such motions as aggressive. We aim to learn these 15 constraint parameters: $[\underline{x}^{\text{ctr}}, \underline{z}^{\text{ctr}}, \overline{z}^{\text{ctr}}]$ (unknown extents of the bar top), $[\underline{x}^{\text{cab}}, \overline{z}^{\text{cab}}]$ (unknown extents of the bar cabinet), $[\underline{\alpha}, \overline{\alpha}, \underline{\beta}, \overline{\beta}, \underline{\gamma}, \overline{\gamma}]$ (unknown pose constraint), and $[\overline{x}^{\text{prox}}, \underline{y}^{\text{prox}}, \overline{y}^{\text{prox}}, \overline{z}^{\text{prox}}]$ (box proxemics constraint).

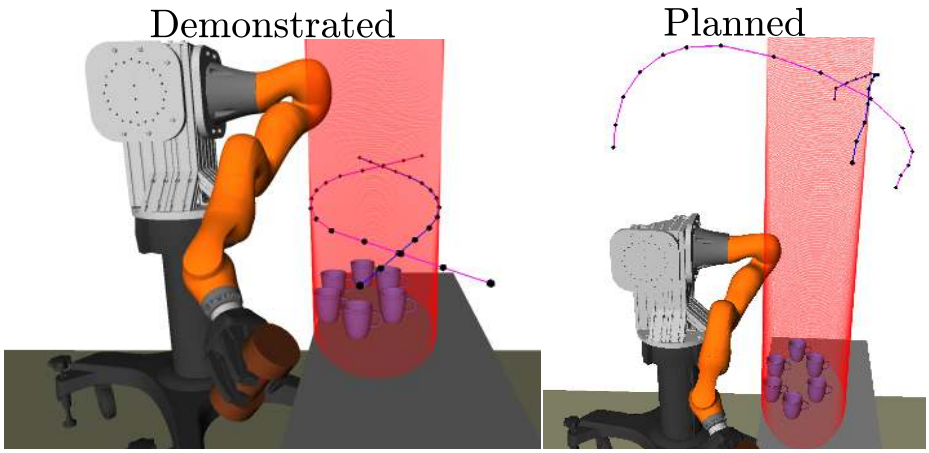


Figure 4.5: Left: arm demos for ellipse example. Right: novel trajectories planned using learned constraint.

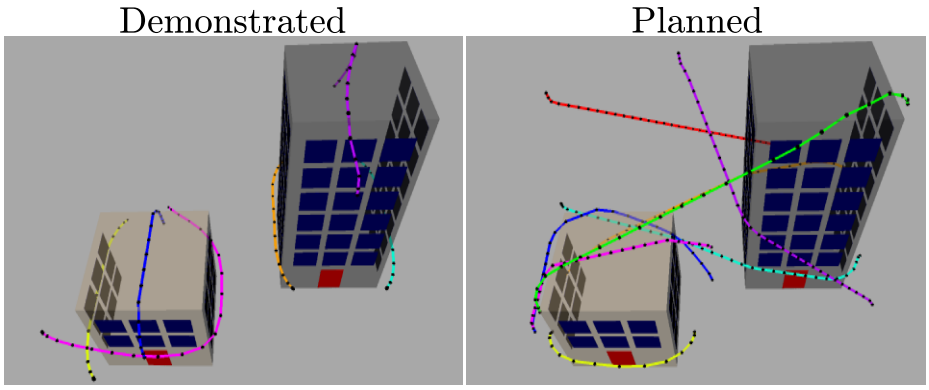


Figure 4.6: Left: quadrotor demonstrations. Right: novel planned trajectories.

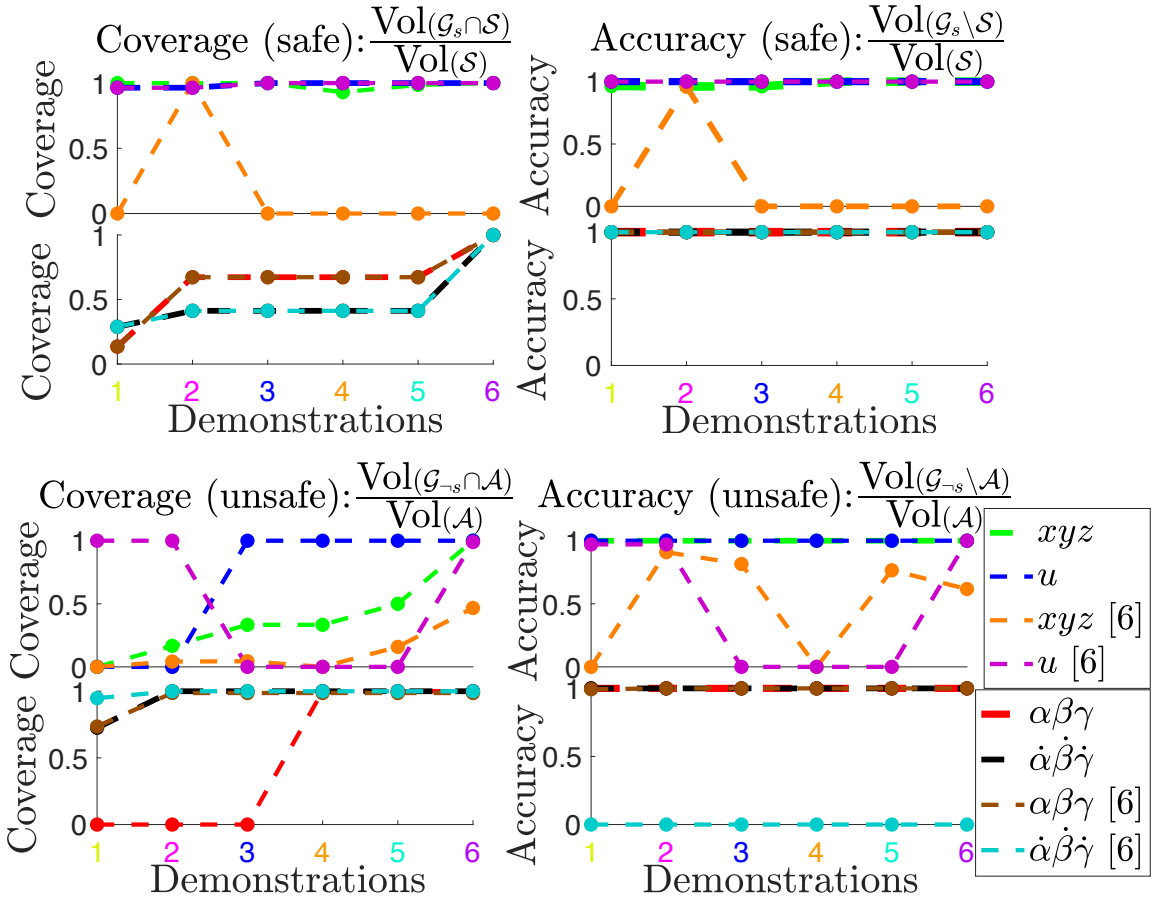


Figure 4.7: Quadrotor statistics: coverage and accuracy for $\mathcal{G}_s$, $\mathcal{G}_{\neg s}$. Demonstration axis is color coded with the demonstrations shown on the left in Fig. 4.6.

The constraint parameters are recovered using the suboptimal analogue of Problem IV.4 (i.e., using the objective function of Problem IV.3), taking 17.2 seconds to solve when using all demonstrations. For tractability, we approximate the swept volume constraint by sampling 18 points on the volume of the arm, mapping them through the arm's forward kinematics, and ensuring that the resulting points are consistent with the obstacle avoidance/proxemics constraint parameters. We solve Problems IV.5/IV.6 to extract the learned guaranteed safe/unsafe sets $\mathcal{G}_s/\mathcal{G}_{\neg s}$, where each query takes 16.4/12.1 seconds on average over 10 queries. Fig. 4.4 shows the coverage of $\mathcal{G}_s/\mathcal{G}_{\neg s}$ compared to the true safe/unsafe sets $\mathcal{S}/\mathcal{A}$, as well as the accuracy of the claimed safe/unsafe sets (Fig. 4.4). We use the sampling-based approach described in Chapter III with Problem 3 of *Chou et al.* (2019) as a baseline. We note that *Chou et al.* (2019) will have difficulty with this example, since the swept volume constraint scales the number of decision variables in Problem 3 of *Chou et al.* (2019) by a factor of 18; this limits the number of trajectory samples which can be tractably used in the constraint-recovery problem. Despite this, the pose constraint is learned fairly well by both the baseline and our method, though the baseline experiences accuracy dips due to suboptimality causing some safe lower-cost trajectories. However, the baseline performs poorly on the position constraints, as it does not learn that the bar top or the bar cabinet are unsafe and does not fully learn the safe set, due to insufficient trajectory samples. In contrast, our KKT-based approach recovers $\mathcal{G}_s = \mathcal{S}$ and $\mathcal{G}_{\neg s} = \mathcal{A}$. Finally, we extract volumes of guaranteed safe space using the procedure in Sec. 4.3.3 and provide the extracted constraint to the CBiRRT planner *Berenson et al.* (2011), generating the novel safe trajectories in Fig. 4.3 (right). This experiment suggests that when Problem IV.1 has many constraints (in this case, due to the swept volume), sampling trajectories leads to worse scalability and worse constraint-recovery performance compared to our KKT-based approach.

Elliptical end-effector constraint: This example is meant to demonstrate the efficacy of using Problem IV.5 in the planning loop. Suppose the arm is manipulating a heavy object near some glassware. For safety, the end effector's center of mass is constrained to lie outside an elliptical cylinder containing the glassware: $\chi_t^\top A(\theta)\chi_t - 2b(\theta)^\top \chi_t + c(\theta) > 0$, where $\chi_t = [x_t, y_t]^\top, A(\theta) = \text{diag}([0.5, 2]), b(\theta) = [0, 1.1]^\top$, and $c(\theta) = 0.505$. We modify Problem IV.5 to use the affine-relaxed KKT conditions, and solving this problem using two demonstrations (Fig. 4.5, left) is enough to recover $\mathcal{G}_s = \mathcal{S}$, $\mathcal{G}_{\neg s} = \mathcal{A}$ via queries. To plan novel constraint-satisfying trajectories, we use STOMP *Kalakrishnan et al.* (2011), where the usual collision/constraint checker is replaced with Problem IV.5. We show two planned trajectories (Fig. 4.5, right), where the planning times were 2 and 6 minutes. Averaged over 10 different queries, solving Problem IV.5 takes 0.073 seconds. We note that this can be sped up by warm-starting Problem IV.5 with the results of previous queries (since like many trajectory optimizers, STOMP samples points near previous iterates).

4.5.3 Quadrotor

We consider the scenario of a quadrotor carrying a delicate payload in an urban environment (see Fig. 4.6). Accordingly, the quadrotor is constrained to not collide

with surrounding buildings (i.e., $(x, y, z) \notin ([\underline{x}_1, \overline{x}_1] \times [\underline{y}_1, \overline{y}_1] \times [0, \overline{z}_1]) \vee ([\underline{x}_2, \overline{x}_2] \times [\underline{y}_2, \overline{y}_2] \times [0, \overline{z}_2])$), satisfy control constraints $\|u_t\| \leq \overline{U}$, pose constraints $\alpha \in [\underline{\alpha}, \overline{\alpha}], \beta \in [\underline{\beta}, \overline{\beta}], \gamma \in [\underline{\gamma}, \overline{\gamma}]$, and angular velocity constraints $\dot{\alpha} \in [\underline{\dot{\alpha}}, \overline{\dot{\alpha}}], \dot{\beta} \in [\underline{\dot{\beta}}, \overline{\dot{\beta}}], \dot{\gamma} \in [\underline{\dot{\gamma}}, \overline{\dot{\gamma}}]$. In this problem, we aim to recover all of these constraints (23 unknown constraint parameters total) using the six demonstrations in Fig. 4.6 (left) by solving Problem IV.4, which takes 19.4 seconds when using all demonstrations. We start from a single box parameterization for each constraint and detect from infeasibility that another box should be added to the position constraint parameterization (see Sec. 4.3.5). The demonstrations are synthetically generated by solving trajectory optimization problems for the cost function $c(\xi) = \sum_{r \in R} \sum_{t=1}^{T-1} \gamma_r (r_{t+1} - r_t)^2$, where $R = \{x, y, z, \dot{\alpha}, \dot{\beta}, \dot{\gamma}\}$ and $\gamma_r = 1$. Our algorithm assumes parametric cost uncertainty of $\gamma_r \in [0.01, 3]$, and we assume the cost function is known exactly for the baseline *Chou et al.* (2019). This problem is especially challenging for the baseline, since having many unknown constraint parameters can lead to non-identifiability of the constraint from the sampled trajectories. Furthermore, the nonlinearity of the quadrotor dynamics (we use the dynamics in *Chou et al.* (2019)) makes sampling difficult. We compute $\mathcal{G}_s / \mathcal{G}_{\neg s}$ via Problems IV.5/IV.6, taking 26.6/43.0 seconds on average over 10 different queries. Fig. 4.7 compares the coverage and accuracy of $\mathcal{G}_s$ and $\mathcal{G}_{\neg s}$ between our approach and the baseline *Chou et al.* (2019) for each of the constraint spaces (position, pose, velocity, and control). The baseline and our approach perform comparably for some of the "simpler" convex constraints (e.g., the angular velocity/control safe sets). However, the baseline struggles to learn the unsafe sets (due to the simultaneous identification of so many constraints from poor trajectory samples) and position constraints (as the quadrotor has second order dynamics, it is difficult to sample combinations of trajectories which uniquely imply a single state is unsafe). We also note that the baseline accuracies fluctuate greatly due to imperfect trajectory sampling and the difficulty of distinguishing between multiple constraints: different data may cause the optimization to switch unsafeness assignments from one constraint to another active constraint). By extracting volumes of $\mathcal{G}_s$ using the method in Sec. 4.3.3, we pass $\mathcal{G}_s$ to a trajectory optimizer *Andersson et al.* (2018) to generate novel safe trajectories (Fig. 4.6, right). This experiment suggests that by avoiding trajectory sampling, our KKT-based approach performs better on high-dimensional nonlinear systems.

4.6 Discussion and Conclusion

We present an algorithm which uses the KKT optimality conditions to determine constraints that make observed demonstrations appear locally-optimal with respect to an uncertain cost function. As the KKT conditions are an *implicit* condition on the set of constraints that can possibly explain the demonstrations, we sidestep the shortcomings of previous methods (*Chou et al.* (2018a, 2019), Chapter III) which rely on sampling lower-cost trajectories as explicit certificates of unsafeness. In future work, we aim to address two shortcomings of our method: first, we require the dynamics to be known in closed form, while the methods in Chapter III just need a simulator; second, the number of decision variables in our method scales linearly

with the number of demonstrations, making it important that the demonstrations are informative with respect to the unknown constraint. To address these issues, we plan to extend our method to handle uncertain dynamics and develop an active learning method to obtain informative demonstrations.

CHAPTER V

Gaussian Process Constraint Learning for Chance-Constrained Planning from Demonstrations

In this chapter, we propose a method for learning constraints represented as Gaussian processes (GPs) from locally-optimal demonstrations. Our approach uses the Karush-Kuhn-Tucker (KKT) optimality conditions to determine where on the demonstrations the constraint is tight, and the gradient of the constraint at those states. We then train a GP representation of the constraint which is consistent with and which generalizes this information. We further show that the GP uncertainty can be used within a kinodynamic RRT to plan probabilistically-safe trajectories, and that we can exploit the GP structure within the planner to exactly achieve a specified safety probability. We demonstrate our method can learn complex, nonlinear constraints demonstrated on a 5D nonholonomic car, a 12D quadrotor, and a 3-link planar arm, all while requiring minimal prior information on the constraint. Our results suggest the learned GP constraint is accurate, outperforming previous constraint learning methods that require more *a priori* knowledge. This chapter is based on the paper *Chou et al.* (2022b).

5.1 Introduction

To address safety in LfD, recent work has represented tasks as *constrained* optimization problems, and learns the unknown cost function and constraints from demonstrations *Chou et al.* (2018a, 2019, 2020b); *Englert et al.* (2017); *Menner et al.* (2019) via the Karush-Kuhn-Tucker (KKT) optimality conditions, enabling constraint learning for complex manipulation and mobile robotics tasks. However, these methods require that the unknown constraints can be described by an *a priori* known representation or parameterization (e.g., as a union of axis-aligned boxes *Chou et al.* (2020a,b)), restricting these methods to the learning of highly-structured constraints. Moreover, such representations can be highly inefficient (e.g., many boxes may be required to approximate complex constraints), leading to a computational burden that makes it challenging to scale these methods up to learn realistic constraints. Consider a demonstrator steering a quadrotor to avoid collisions with a tree (Fig. 5.1). On the

one hand, we are unlikely to obtain an efficient constraint representation for learning trees *a priori* unless we can learn one (e.g., via deep learning) using an enormous number of demonstrations, and on the other hand, a prohibitive number of boxes is needed to represent the tree.

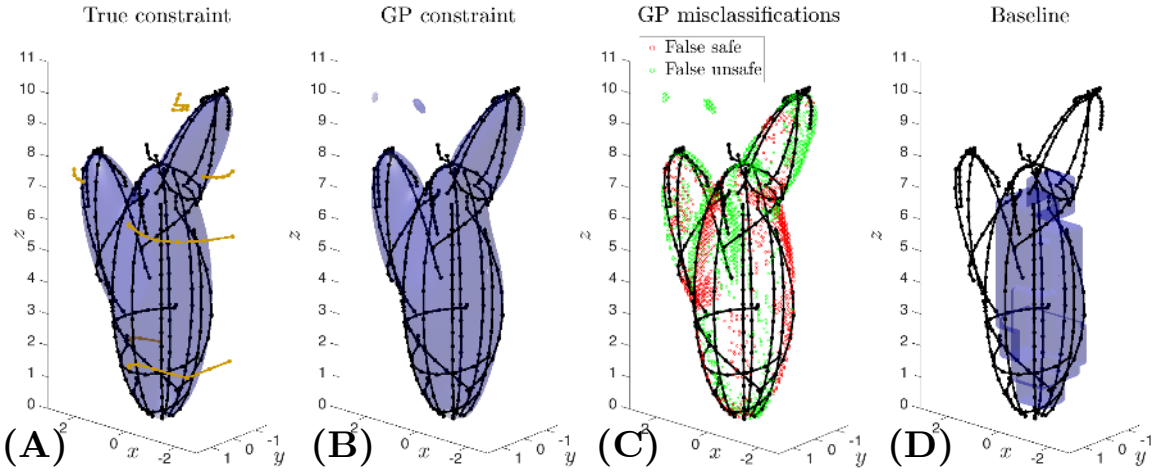


Figure 5.1: Demonstrations (black) avoiding a tree-like obstacle on a 12D quadrotor. **(A)** True constraint (blue); plans using the GP constraint (gold). **(B)** Posterior mean of the GP constraint (blue). **(C)** Errors of GP posterior mean w.r.t. the true constraint. **(D)** Constraint learned via *Chou et al.* (2020b) using 6 boxes.

We address these issues via the insight that the demonstrations' Karush-Kuhn-Tucker (KKT) optimality conditions provide information on 1) where the unknown constraint is tight on the demonstrations, and 2) the gradient of the constraint (i.e., the surface normal on the constraint boundary) at those points. Crucially, we show that this information can be extracted in a way that is *agnostic* to the chosen constraint representation. This is in sharp contrast to prior work *Chou et al.* (2020b), which uses the KKT conditions to directly determine a set of constraint parameters, for a fixed constraint parameterization, which make the demonstrations satisfy their KKT conditions. This representation-agnostic constraint information can be incorporated into a flexible non-parametric Gaussian process (GP) function approximator, which enables constraint learning while requiring minimal *a priori* knowledge on the underlying constraint structure. Our contributions are:

- We show how to use the demonstrations' KKT conditions to extract information on the values and gradients of the unknown constraint, how to ensure it is robust to the ill-posedness of the constraint learning problem, and how it can be jointly incorporated into a GP.
- We show how the uncertainty of the learned GP constraint can be used to plan chance-constrained trajectories which satisfy the unknown constraint with a specified probability, and how the Gaussian structure of the uncertainty can be exploited in the planner to exactly compute trajectory safety probabilities.
- We evaluate our method on complex nonlinear constraint learning problems

demonstrated on a 5D nonholonomic car, a 12D quadrotor and a three-link planar arm, showing that our method outperforms baselines.

5.2 Related Work

Our method is related to prior work in IOC *Englert et al.* (2017); *Johnson et al.* (2013); *Keshavarz et al.* (2011) that uses the KKT conditions to learn an unknown cost function, assuming the constraints are known. Other IOC methods *Finn et al.* (2016); *Levine et al.* (2011); *Wulfmeier et al.* (2017) use flexible function approximators to learn unknown cost functions without using known features, again assuming known constraints. Our approach is complementary to these methods, as it seeks to learn the constraints.

Our work also builds upon constraint learning methods *Chou et al.* (2018a, 2019, 2020b); *Pérez-D'Arpino and Shah* (2017), which often assume a known constraint parameterization to simplify the inverse problem which recovers the unknown constraint *Chou et al.* (2020a,b). When this assumption is removed *Chou et al.* (2019), the unsafe set defined by the unknown constraint is assumed to be well-approximated by a finite union of simple unsafe sets, e.g., axis-aligned boxes *Chou et al.* (2019). However, the inverse problem scales exponentially with the number of simple sets, as each set adds binary decision variables to the optimization. This renders complex constraints infeasible to learn unless the true constraint representation is known, restricting previous methods (e.g., *Chou et al.* (2020a,b)) to learn simple constraints, e.g., unions of a few boxes *Chou et al.* (2020a,b), or to know the parameterization (*Chou et al.*, 2018a, Fig. 6) (*Chou et al.*, 2020b, Fig. 2). Our work is also related to methods that plan using the uncertainty in the learned constraint, e.g., *Chou et al.* (2020a). However *Chou et al.* (2020a) scales exponentially in the number of simple unsafe sets; in contrast, we use the uncertainty of the GP constraint to scale more gracefully.

Finally, our work relates to planning under uncertainty, where the uncertainty may arise from sensing *Burns and Brock* (2007), state estimation *Bry and Roy* (2011), motion *Aoude et al.* (2013), and the environment/obstacles; our work relates most closely to this final case. *Blackmore et al.* (2006) plans with uncertain obstacles via chance-constrained optimization and requires polytopic obstacles and linear Gaussian dynamics. Under similar assumptions, *Luders et al.* (2010) embeds chance constraints in a Rapidly-Exploring Random Tree (RRT) *LaValle and James J. Kuffner* (2001). In contrast, we assume deterministic dynamics but can handle GP-representable constraints and nonlinear dynamics.

5.3 Preliminaries and Problem Statement

5.3.1 Demonstrator's problem and KKT optimality conditions

We represent a length T demonstration of a task Π performed on a deterministic discrete-time nonlinear system $x_{t+1} = f(x_t, u_t, t)$, $x \in \mathcal{X} \subseteq \mathbb{R}^{n_x}$, $u \in \mathcal{U} \subseteq$

$\mathbb{R}^{n_u}$ as a constrained optimization over state/control trajectories $\xi_{xu} \doteq (\xi_x, \xi_u) \doteq [x_1, \dots, x_T, u_1, \dots, u_{T-1}]$:

Problem V.1 (Forward (demonstrator's) problem / task Π)**.**

$$\begin{aligned} \underset{\xi_{xu}}{\text{minimize}} \quad & c_\Pi(\xi_{xu}) \\ \text{subject to} \quad & \phi(\xi_{xu}) \in \mathcal{S} \subseteq \mathcal{C} \Leftrightarrow \quad \mathbf{g}^*_{\neg k}(\phi(\xi_{xu})) \leq \mathbf{0} \\ & \bar{\phi}(\xi_{xu}) \in \bar{\mathcal{S}} \subseteq \bar{\mathcal{C}}, \quad \phi_\Pi(\xi_{xu}) \in \mathcal{S}_\Pi \subseteq \mathcal{C}_\Pi \\ & \quad \Leftrightarrow \quad \mathbf{h}_k(\xi_{xu}) = \mathbf{0}, \quad \mathbf{g}_k(\xi_{xu}) \leq \mathbf{0} \end{aligned}$$

where $c_\Pi(\cdot)$ is a known, possibly task-dependent cost function, and $\phi(\cdot)$ maps state/control trajectories to *constraint states* κ in *constraint space* $\mathcal{C}$ (i.e., $\kappa \in \mathcal{C}$), where the constraint is evaluated. For example, for an obstacle constraint, $\phi(\cdot)$ would select the position components of the states. The safe set $\mathcal{S} \subseteq \mathcal{C} \subseteq \mathbb{R}^{n_c}$ is defined by the unknown inequality constraint $\mathbf{g}^*_{\neg k}(\phi(\xi_{xu})) \leq \mathbf{0}$ and is unknown to the learner. $\bar{\phi}(\cdot)$ and $\phi_\Pi(\cdot)$ map to spaces $\bar{\mathcal{C}}$ and $\mathcal{C}_\Pi$, containing a known task-shared safe set $\bar{\mathcal{S}}$ and task-dependent safe set $\mathcal{S}_\Pi$, defined by known equality and inequality constraints $\mathbf{h}_k(\xi_{xu}) = \mathbf{0}$, $\mathbf{g}_k(\xi_{xu}) \leq \mathbf{0}$. We embed the dynamics in $\bar{\mathcal{S}}$ and the start/goal constraints in $\mathcal{S}_\Pi$. We focus on unknown scalar, state-dependent, time-separable inequality constraints

$$\mathbf{g}^*_{\neg k}(\phi(\xi_{xu})) \leq \mathbf{0} \;\Leftrightarrow\; g^*_{\neg k}(\phi_{\text{sep}}(x_t)) \leq 0, \quad \forall t = 1, ..., T, \tag{5.1}$$

where $\phi_{\text{sep}} : \mathcal{X} \mapsto \mathcal{C}$ is the time-separable counterpart of $\phi(\cdot)$, $g^*_{\neg k} : \mathcal{C} \mapsto \mathbb{R}$, and $\kappa_t = \phi_{\text{sep}}(x_t)$. We note that extending to control-dependent constraints is straightforward. Moreover, we can learn the (un)safe set for an M-dimensional vector-valued constraint by learning $g^*_{\neg k}(\cdot) = \max_{i=1,\dots,M} g^*_{i,\neg k}(\cdot)$. We assume each demonstration ξ_{loc} solves Prob. V.1 to local optimality, satisfying Prob. V.1's KKT conditions *Boyd and Vandenberghe* (2004). With Lagrange multipliers $\boldsymbol{\lambda}$, $\boldsymbol{\nu}$, the relevant KKT conditions for the jth demonstration ξ^{dem}_j, denoted KKT(ξ^{dem}_j), are:

$$\begin{aligned} \text{Primal feasibility:} \quad & \mathbf{g}^*_{\neg k}(\phi(\xi^{\text{dem}}_j)) \leq \mathbf{0}, && \text{(5.2a)} \\ \text{Lagrange mult.} \quad & \boldsymbol{\lambda}^j_k \geq \mathbf{0} && \text{(5.2b)} \\ \text{nonnegativity:} \quad & \lambda^{j,t}_{\neg k} \geq 0, \; t = 1, ..., T^j \;\Leftrightarrow\; \boldsymbol{\lambda}^j_{\neg k} \geq \mathbf{0} && \text{(5.2c)} \\ \text{Complementary} \quad & \boldsymbol{\lambda}^j_k \odot \mathbf{g}_k(\xi^{\text{dem}}_j) = \mathbf{0} && \text{(5.2d)} \\ \text{slackness:} \quad & \boldsymbol{\lambda}^j_{\neg k} \odot \mathbf{g}^*_{\neg k}(\phi(\xi^{\text{dem}}_j)) = \mathbf{0} && \text{(5.2e)} \\ \text{Stationarity:} \quad & \nabla_{\xi_{xu}} c_\Pi(\xi^{\text{dem}}_j) + \boldsymbol{\lambda}^{j\top}_k \nabla_{\xi_{xu}} \mathbf{g}_k(\xi^{\text{dem}}_j) \\ & \quad + \boldsymbol{\lambda}^{j\top}_{\neg k} \nabla_{\xi_{xu}} \mathbf{g}^*_{\neg k}(\phi(\xi^{\text{dem}}_j)) && \text{(5.2f)} \\ & \quad + \boldsymbol{\nu}^{j\top}_k \nabla_{\xi_{xu}} \mathbf{h}_k(\xi^{\text{dem}}_j) = \mathbf{0} \end{aligned}$$

where $\odot$ denotes element-wise multiplication. Here, $\boldsymbol{\lambda}^j_k \in \mathbb{R}^{N^j_{\text{ineq}}}$, $\boldsymbol{\nu}^j_k \in \mathbb{R}^{N^j_{\text{eq}}}$, and $\boldsymbol{\lambda}^j_{\neg k} \in \mathbb{R}^{T^j}$ are vectorized Lagrange multipliers for the known inequality, known equality, and unknown inequality constraints for ξ^{dem}_j, i.e., $\boldsymbol{\lambda}^j_{\neg k} = [\boldsymbol{\lambda}^{j,1}_{\neg k}, \dots \boldsymbol{\lambda}^{j,T^j}_{\neg k}]$. The blue quantities are unknown to the learner. Intuitively, (5.2a) enforces that ξ^{dem}_j is feasible for Prob. V.1 (it lies in $\mathcal{S}$ and satisfies the known constraints), that a multiplier is

zero unless its associated constraint is tight (5.2b)-(5.2e), and that its cost cannot be locally improved (5.2f).

In previous work *Chou et al.* (2020a,b), the unknown constraint is modeled as $g^*_{\neg k}(z, \theta)$, where θ are parameters for a known representation of $g^*_{\neg k}$ with a low-order dependence on θ, e.g., linear $g^*_{\neg k}(z, \theta) = \theta^\top g(z)$, where $g(z)$ are known features; the constraint learning problem then reduces to finding θ. In contrast, we do not require a known parameterization for $g^*_{\neg k}(\cdot)$, instead approximating $g^*_{\neg k}(\cdot)$ as a GP to be learned.

5.3.2 Overview of Gaussian processes

A GP is a set of (potentially infinitely many) random variables, any finite number of which have a joint Gaussian distribution *Rasmussen and Williams* (2005). It is defined by a mean function $m(x)$ and a covariance function $k(x, x')$. In regression, GPs are often used as the prior distribution for an unknown function $f(x)$ of interest, i.e., $f \sim \mathcal{GP}(m, k)$. Given an input-output dataset $\mathcal{D} = \{(x_n, y_n)\}_{n=1}^{N_\mathrm{d}}$, and assuming a noisy observation model $y_n \sim \mathcal{N}(f(x_n), \sigma^2)$, the predictive conditional posterior $\tilde{f}|\mathcal{D}$ is also a Gaussian if a GP is used as the prior. In performing inference at a set of points $\{z_m\}_{m=1}^{N_\mathrm{q}}$, the posterior mean and covariance on these points are given by

$$\mathbb{E}[\tilde{f}(\mathbf{Z})|\mathcal{D}] = k(\mathbf{Z}, \mathbf{X})(k(\mathbf{X}, \mathbf{X}) + \sigma^2 I)^{-1}\mathbf{Y}, \tag{5.3}$$

$$\mathrm{cov}(\tilde{f}(\mathbf{Z})|\mathcal{D}) = k(\mathbf{Z}, \mathbf{Z}) - k(\mathbf{Z}, \mathbf{X})(k(\mathbf{X}, \mathbf{X}) + \sigma^2 I)^{-1}k(\mathbf{X}, \mathbf{Z}). \tag{5.4}$$

where $\mathbf{Z}$, $\mathbf{X}$, and $\mathbf{Y}$ are vectors containing all elements in $\{z_m\}_{m=1}^{N_\mathrm{q}}$, $\{x_n\}_{n=1}^{N_\mathrm{d}}$, and $\{y_n\}_{n=1}^{N_\mathrm{d}}$, respectively *Rasmussen and Williams* (2005).

5.3.3 Problem statement

Given locally-optimal demonstrations $\{\xi_j^{\mathrm{dem}}\}_{j=1}^{N_s}$, we want an estimate $g_{\neg k}(\cdot)$ of the unknown constraint $g^*_{\neg k}(\cdot)$, defining

$$\mathcal{S} = \{\phi_{\mathrm{sep}}(x) \mid g_{\neg k}(\phi_{\mathrm{sep}}(x)) \leq 0\} = \{\kappa \mid g_{\neg k}(\kappa) \leq 0\} \tag{5.5}$$

as a safe set that is consistent with the demonstrations' KKT conditions, where the true constraint $g^*_{\neg k}(\cdot)$ is assumed to be a sample from $\mathcal{GP}(m, k)$. Moreover, we wish to use the learned constraint to plan trajectories ξ_{xu}^{plan} that connect novel start/goal states while satisfying $g^*_{\neg k}(\cdot)$ with at least some specified probability $1 - \delta$, i.e., $\Pr(\mathbf{g}^*_{\neg k}(\phi(\xi_{xu}^{\mathrm{plan}})) \leq \mathbf{0}) \geq 1 - \delta$.

5.4 Method

Our method determines where the unknown constraint is tight and its gradient at those points from the KKT conditions (Sec. 5.4.1), uses this information to train a GP representation of the constraint (Sec. 5.4.2), and plans novel probabilistically-safe trajectories using the learned constraint (Sec. 5.4.3). We overview the flow of our method in Fig. 5.2.

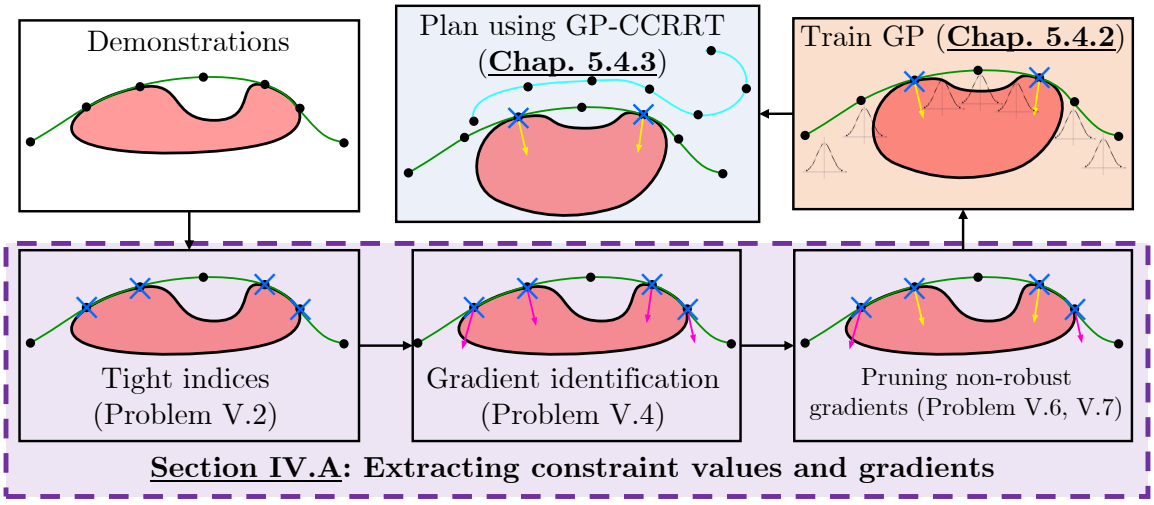


Figure 5.2: Method flow. Given a set of locally-optimal demonstrations, we first find consistent constraint values and gradients (Sec. 5.4.1), then use this data to train a consistent GP constraint representation (Sec. 5.4.2), and then finally plan probabilistically-safe trajectories using the learned GP constraint (Sec. 5.4.3).

5.4.1 Obtaining constraint value and gradient information

For a locally-optimal demonstration, the KKT conditions (5.2) provide information on the following:

1. If/when the unknown constraint $g^*_{\neg k}(\cdot)$ is *tight* (i.e., at which time steps of the demonstration $g^*_{\neg k}(\phi_{\text{sep}}(x_t)) = 0$) via complementary slackness (5.2e) and stationarity (5.2f).

2. How the constraint changes locally around these tight demonstration points, in the form of the constraint gradient at that point, $\nabla_{x_t} g^*_{\neg k}(\phi_{\text{sep}}(x_t))$, via stationarity (5.2f).

Combining both sources of information is crucial in recovering an accurate constraint that is KKT-consistent.

5.4.1.1 Constraint value information

We first describe a method for inferring when the unknown constraint $g^*_{\neg k}(\cdot)$ is tight. As shorthand, let the stationarity residual

$$\text{stat}^j(\boldsymbol{\lambda}^j_k, \boldsymbol{\lambda}^j_{\neg k}, \boldsymbol{\nu}^j_k) = \begin{bmatrix} \text{stat}^j_{x_1}(\boldsymbol{\lambda}^j_k, \boldsymbol{\lambda}^j_{\neg k}, \boldsymbol{\nu}^j_k), \\ \vdots \\ \text{stat}^j_{x_T}(\boldsymbol{\lambda}^j_k, \boldsymbol{\lambda}^j_{\neg k}, \boldsymbol{\nu}^j_k) \\ \text{stat}^j_{u_1}(\boldsymbol{\lambda}^j_k \boldsymbol{\lambda}^j_{\neg k}, \boldsymbol{\nu}^j_k) \\ \vdots \\ \text{stat}^j_{u_t}(\boldsymbol{\lambda}^j_k \boldsymbol{\lambda}^j_{\neg k}, \boldsymbol{\nu}^j_k) \end{bmatrix} \in \mathbb{R}^{|\xi_{xu}|}$$

be the LHS of the stationarity condition (5.2f) for the jth demonstration ξ_j^{dem}, where $\text{stat}^j_{x_t/u_t}(\boldsymbol{\lambda}_k^j, \boldsymbol{\lambda}_{\neg k}^j, \boldsymbol{\nu}_k^j) \in \mathbb{R}^{n_x}/\mathbb{R}^{n_u}$ is the sub-vector containing the residual terms for x_t / u_t. Recall that complementary slackness (5.2e) enforces that at each timestep, the Lagrange multiplier for the the unknown constraint is zero unless the constraint is tight. Moreover, as any locally-optimal trajectory ξ_{loc} must satisfy the stationarity condition (5.2f), we can determine that the unknown constraint $g^*_{\neg k}(\cdot)$ must be tight on ξ_j^{dem} at timestep t if we cannot force the norm of the stationarity residual at that timestep to be zero, i.e., $\|\text{stat}^j_{x_t}\| > 0$, while also enforcing that $g^*_{\neg k}(\phi_{\text{sep}}(x_t))$ is not tight (cf. Fig. 5.3.A for intuition) and that the KKT conditions for the known constraints are satisfied. This is achieved by solving Prob. V.2 – a rapidly-solvable linear program (LP):

Problem V.2 (Tightness check at time t on demonstration j)**.**

$$\begin{aligned} \underset{\boldsymbol{\lambda}_k^j, \boldsymbol{\nu}_k^j}{\text{minimize}} \quad & \left\|\text{stat}^j_{x_t}(\boldsymbol{\lambda}_k^j, \mathbf{0}, \boldsymbol{\nu}_k^j)\right\|_1 \\ \text{subject to} \quad & (5.2\text{b}), (5.2\text{d}), \end{aligned}$$

where the effect of the unknown inequality constraint on the residual is erased by zeroing out its corresponding Lagrange multipliers $\boldsymbol{\lambda}^j_{\neg k}$. Then, the following result holds:

Corollary V.3. If the optimal value of Prob. V.2, denoted $p_2^{t,j,*}$, is greater than 0, then the true constraint is tight: $g^*_{\neg k}(\kappa_t^j) = 0$.

Proof. Suppose for contradiction that $g^*_{\neg k}(\kappa_t^j) < 0$. Then, since ξ_j^{dem} satisfies (5.2), $g^*_{\neg k}(\kappa_t^j) < 0$ implies via (5.2e)-(5.2f) that there exists $\boldsymbol{\lambda}_k^j$, $\boldsymbol{\lambda}^j_{\neg k} = \mathbf{0}$, $\boldsymbol{\nu}_k^j$ such that $p_2^{t,j,*} = 0$. However, by the theorem statement, $p_2^{t,j,*} > 0$. Contradiction. □

By solving Prob. V.2 and checking if $p_2^{t,j,*} > 0$ for $t = 1, \ldots, T^j$, we can find a set of timesteps where $g^*_{\neg k}(\kappa_t^j) = 0$; call these identified tight timesteps $\mathbf{t}^j_{\text{tight}}$. Intuitively, Prob. V.2 checks if we can ensure $g^*_{\neg k}(\kappa_t^j) = 0$ despite the known constraints, e.g., dynamics, control constraints, which may be simultaneously tight. By solving Prob. V.2 $\sum_{j=1}^{N_{\text{dem}}} T^j$ times (once for each timestep), we can check tightness over the entire dataset. We close with two important remarks. First, complementary slackness and stationarity do not provide information on $g_{\neg k}(\kappa_t^j)$ for timesteps $\mathbf{t}^j_{\neg\text{tight}} \doteq \{1, \ldots, T^j\} \setminus \mathbf{t}^j_{\text{tight}}$; we can only deduce using primal feasibility that $g^*_{\neg k}(\kappa_t^j) \leq 0$ for $t \in \mathbf{t}^j_{\neg\text{tight}}$. Second, the estimated set of tight timesteps $\mathbf{t}_{\text{tight}}$ may only be a subset of the true set of tight timesteps, i.e., $\mathbf{t}^j_{\text{tight}} \subseteq \{t \mid g^*_{\neg k}(\kappa_t^j) = 0\}$; this is because the system may lie on the constraint boundary but the cost cannot be improved by crossing it (Fig. 5.3.C), i.e., there are multipliers such that $\|\text{stat}^j_{x_t}(\boldsymbol{\lambda}_k^j, \mathbf{0}, \boldsymbol{\nu}_k^j)\|_1 = 0$ despite $\boldsymbol{\lambda}^{j,t}_{\neg k} = 0$; KKT cannot guarantee that these points are tight.

5.4.1.2 Gradient value information

Next, we obtain a set of KKT-consistent gradients of the unknown constraint at each identified tight timestep $t \in \mathbf{t}^j_{\text{tight}}$. In Prob. V.4, we set the Lagrange mutipliers

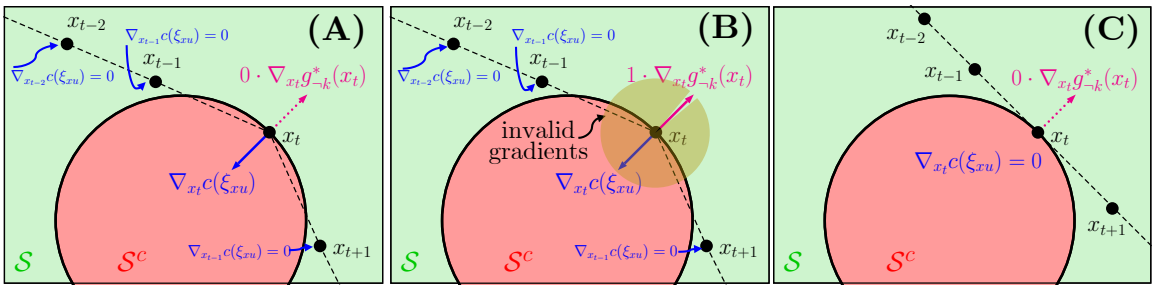


Figure 5.3: Consider a demonstrator minimizing path length on a kinematic system; $\phi_{\text{sep}}(x_t) = x_t \in \mathbb{R}^2$. In this simplified setting, we can interpret (5.2f) as balancing between vectors ∇c and $\lambda \nabla g_{\neg k}$; if they cancel to $\mathbf{0}$, stationarity holds. We visualize this for Prob. V.2-V.4. **(A)** Prob. V.2: $\|\text{stat}^{x_t}\|$ can only go to zero if $\boldsymbol{\lambda}^t_{\neg k} > 0$; thus, we detect $g^*_{\neg k}(x_t) = 0$. **(B)** Prob. V.4: only a scaling of the magenta constraint normal can make $\|\text{stat}^{x_t}\| = 0$; all gradients in gold are are not anti-parallel to ∇c and cannot cancel it. **(C)**: sometimes if $g^*_{\neg k}(x_t) = 0$, it is still possible for $\|\text{stat}^{x_t}\| = 0$ with $\boldsymbol{\lambda}^t_{\neg k} = 0$.

$\boldsymbol{\lambda}^{j,t}_{\neg k} = 1$, for all $t \in \mathbf{t}^j_{\text{tight}}$, and set the non-tight Lagrange multipliers as $\boldsymbol{\lambda}^{j,t}_{\neg k} = 0$, for all $t \in \mathbf{t}^j_{\neg\text{tight}}$; denote the concatenation of the multipliers as $\mathbf{1}_{\text{tight}}(\xi^{\text{dem}}_j)$. We then solve for gradients $\nabla_{x_t} g_{\neg k}(\phi_{\text{sep}}(x^j_t))$, for all $t \in \mathbf{t}^j_{\text{tight}}$, which make the demonstration KKT-consistent along with the Lagrange multipliers of the known constraints:

Problem V.4 (Gradient identification on demonstration j)**.**

$$\begin{aligned} \text{find} \quad & \boldsymbol{\lambda}^j_k, \boldsymbol{\nu}^j_k, \nabla_{x_t} g_{\neg k}(\phi_{\text{sep}}(x^j_t)), \forall t \in \mathbf{t}^j_{\text{tight}} \\ \text{subject to} \quad & (5.2\text{a}), (5.2\text{b}), (5.2\text{d}) \\ & \text{stat}^j(\boldsymbol{\lambda}^j_k, \mathbf{1}_{\text{tight}}(\xi^{\text{dem}}_j), \boldsymbol{\nu}^j_k) = \mathbf{0}. \end{aligned}$$

Prob. V.4 remains an LP as we fix $\boldsymbol{\lambda}^j_{\neg k}$ to avoid bilinearity. To show this does not overly restrict the set of KKT-consistent gradients, we show that while the true gradient may be not be feasible for Prob. V.4, a *positive scaling* of it will be. A scaled gradient is acceptable for two reasons. First, it can be impossible to uniquely identify an unscaled gradient via KKT alone: by letting the tight multipliers $\boldsymbol{\lambda}^{j,t}_{\neg k}$, $t \in \mathbf{t}^j_{\text{tight}}$ take positive non-unit values, they can scale their values to satisfy KKT for different scalings of $\nabla_{x_t} g_{\neg k}(\phi_{\text{sep}}(x^j_t))$. Second, while a scaled gradient affects how quickly the constraint changes away from the tight point, it does not affect the shape of the constraint (i.e., it does not rotate the unit surface normal vector at the boundary of the unsafe set). Let $\mathcal{F}$ be the feasible set of Prob. V.4 and $\text{proj}_{\nabla g_{\neg k}}(\mathcal{F}) \doteq \{\nabla g_{\neg k} \mid \exists(\boldsymbol{\lambda}_k, \boldsymbol{\nu}_k, \nabla g_{\neg k}) \in \mathcal{F}\}$. Then, we have the following result:

Theorem V.5. A positive scaling of the true constraint gradient $\alpha^j_t \nabla_{x_t} g^*_{\neg k}(\phi_{\text{sep}}(x^j_t))$, for $\alpha^j_t > 0$, for all $t \in \mathbf{t}^j_{\text{tight}}$, is contained in $\text{proj}_{\nabla g_{\neg k}}(\mathcal{F})$.

Proof. Since ξ^{dem}_j is locally-optimal, it satisfies its KKT conditions; i.e., there exists $\boldsymbol{\lambda}^j_k \geq \mathbf{0}$, $\boldsymbol{\nu}^j_k$, and $\boldsymbol{\lambda}^j_{\neg k} \geq \mathbf{0}$, where $\boldsymbol{\lambda}^{j,t}_{\neg k} = 0$, for all $t \in \mathbf{t}^j_{\neg\text{tight}}$. This is via Prob.

V.2: if $t \in \mathbf{t}^j_{\neg\text{tight}}$, the KKT conditions can be satisfied if $\boldsymbol{\lambda}^{j,t}_{\neg k} = 0$. Denote one such KKT-consistent set of multipliers as $\boldsymbol{\lambda}^{j,*}_k$, $\boldsymbol{\nu}^{j,*}_k$, and $\boldsymbol{\lambda}^{j,*}_{\neg k}$. As $g^*_{\neg k}(\cdot)$ is state-dependent and time-separable, we can write $\boldsymbol{\lambda}^{j\top}_{\neg k}\nabla_{\xi_{xu}}\mathbf{g}^*_{\neg k}(\phi(\xi^{\text{dem}}_j)) = [\boldsymbol{\lambda}^{j,1}_{\neg k}\nabla_{x_1}g^*_{\neg k}(\phi_{\text{sep}}(x^j_1)), \ldots, \boldsymbol{\lambda}^{j,T}_{\neg k}\nabla_{x_T}g^*_{\neg k}(\phi_{\text{sep}}(x^j_T)), \mathbf{0}_{1\times n_u(T^j-1)}]$. Then, a feasible solution for Prob. V.4 is $\boldsymbol{\lambda}^j_k = \boldsymbol{\lambda}^{j,*}_k$, $\boldsymbol{\nu}^j_k = \boldsymbol{\nu}^{j,*}_k$, and $\nabla_{x_t}g_{\neg k}(\phi_{\text{sep}}(x^j_t)) = \boldsymbol{\lambda}^{j,*,t}_{\neg k}\nabla_{x_t}g^*_{\neg k}(\phi_{\text{sep}}(x^j_t))$, for all $t \in \mathbf{t}^j_{\text{tight}}$. Thus, the theorem holds by setting $\alpha^j_t = \boldsymbol{\lambda}^{j,*,t}_{\neg k}$. □

While fixing $\boldsymbol{\lambda}^j_{\neg k}$ restricts $\text{proj}_{\nabla g_{\neg k}}(\mathcal{F})$ and reduces scaling ambiguity, due to other active constraints, these recovered KKT-consistent gradients may still not be unique. While scaled gradients are tolerable, a rotation of the true gradient can also lie in $\text{proj}_{\nabla g_{\neg k}}(\mathcal{F})$, complicating the learning as: 1) the unsafe set shape becomes uncertain, 2) modeling gradient uncertainty is challenging, as determining the set of all consistent gradient vectors is computationally intensive *Chou et al.* (2020a), and 3) the gradient uncertainty cannot be well-modeled by a Gaussian distribution, as required by our GP representation.

Though quantifying the uncertainty in the constraint gradients is challenging, we can efficiently check if a given KKT-consistent normal vector is unique, modulo a positive scaling. This can be done by checking that there does not exist another KKT-consistent normal vector that either A) lies in the orthogonal complement of the given normal vector or B) points in directly the opposite direction (see Fig. 5.4). Let $\nabla_{x_t}\tilde{g}^j_{\neg k}$ be the gradient returned by Prob. V.4 for timestep t on ξ^{dem}_j and $\nabla_{x_t}\tilde{g}^{j,\perp}_{\neg k} \in \mathbb{R}^{n_c\times(n_c-1)}$ as a basis for its orthogonal complement. Then, condition A) can be checked by solving:

Problem V.6 (Orthogonal check at time t on demonstration j)**.**

$$\begin{aligned} \underset{\boldsymbol{\lambda}^j_k, \boldsymbol{\nu}^j_k, \nabla_{x_t}g_{\neg k}(\phi_{\text{sep}}(x^j_t))}{\text{maximize}} \quad & \left\|\nabla_{x_t}g_{\neg k}(\phi_{\text{sep}}(x^j_t))^\top \nabla_{x_t}\tilde{g}^{j,\perp}_{\neg k}\right\|_1 \\ \text{subject to} \quad & (5.2\text{a}), (5.2\text{b}), (5.2\text{d}) \\ & \text{stat}^j(\boldsymbol{\lambda}^j_k, \mathbf{1}_{\text{tight}}(\xi^{\text{dem}}_j), \boldsymbol{\nu}^j_k) = \mathbf{0}. \end{aligned}$$

Intuitively, Prob. V.6 searches for an alternate gradient in the orthogonal complement of the gradient obtained via Prob. V.4 such that some assignment of multipliers also exists to satisfy the KKT conditions. Due to the non-convex objective, Prob. V.6 can be modeled as a mixed integer linear program (MILP) with only a small number of binary variables, thus remaining rapidly-solvable. Next, condition B) can be checked via:

Problem V.7 (Anti-parallel check; time t on demonstration j)**.**

$$\begin{aligned} \underset{\boldsymbol{\lambda}^j_k, \boldsymbol{\nu}^j_k, \nabla_{x_t}g_{\neg k}(\phi_{\text{sep}}(x^j_t))}{\text{minimize}} \quad & \nabla_{x_t}g_{\neg k}(\phi_{\text{sep}}(x^j_t))^\top \nabla_{x_t}\tilde{g}^j_{\neg k} \\ \text{subject to} \quad & (5.2\text{a}), (5.2\text{b}), (5.2\text{d}) \\ & \text{stat}^j(\boldsymbol{\lambda}^j_k, \mathbf{1}_{\text{tight}}(\xi^{\text{dem}}_j), \boldsymbol{\nu}^j_k) = \mathbf{0}. \end{aligned}$$

Prob. V.7, an LP, searches for a KKT-consistent gradient minimizing the dot product with $\nabla_{x_t}\tilde{g}^j_{\neg k}$, i.e., pointing as anti-parallel to the original gradient as possible. Thm.

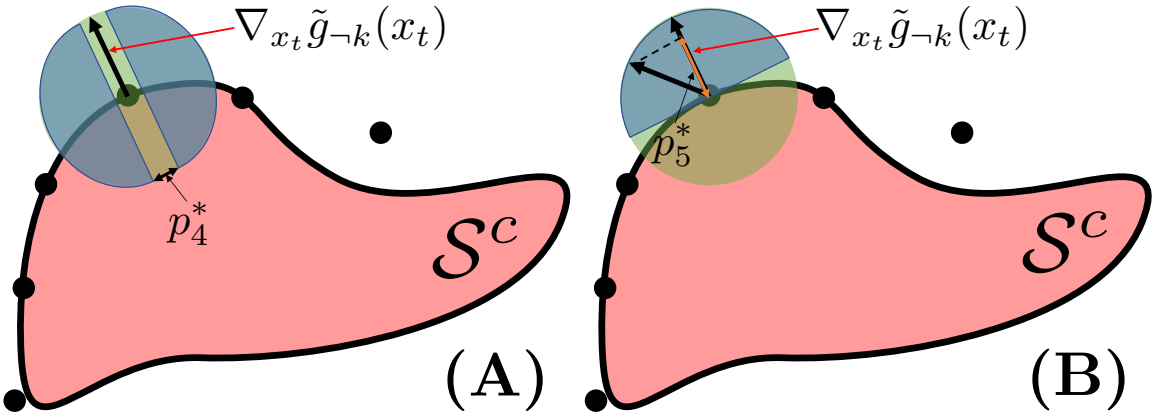


Figure 5.4: Prob. V.6 and V.7 intuition. **(A)**: Prob. V.6 searches for a new gradient orthogonal to the original gradient by maximizing the distance from the origin as measured in the coordinates of $\nabla_{x_t}\tilde{g}^{\perp}_{\neg k}$. If $p_4^* = 0$ (i.e., the gradient remains in the gap between the blue areas as the gap $\to 0$), the new gradient must remain in the span of the original gradient. **(B)**: Prob. V.7 searches for a new gradient with minimal dot product w.r.t. the original gradient; if the result remains in the blue semicircle (i.e., $p_5^* > 0$) and $p_4^* = 0$, the gradient from Prob. V.4 is unique up to a scaling.

V.8 shows how Probs. V.6-V.7 can check gradient uniqueness. Denote the optimal values of Prob. V.6 and V.7 as p_4^* and p_5^*. We have:

Theorem V.8. If $p_4^* = 0$ and $p_5^* > 0$, the true gradient $\nabla_{x_t} g^*_{\neg k}(\phi_{\text{sep}}(x_t^j))$ is a positive scaling of the recovered gradient $\nabla_{x_t}\tilde{g}_{\neg k}(\phi_{\text{sep}}(x_t^j))$, i.e., there exists $\alpha > 0$ such that $\nabla_{x_t} g^*_{\neg k}(\phi_{\text{sep}}(x_t^j)) = \alpha \nabla_{x_t}\tilde{g}_{\neg k}(\phi_{\text{sep}}(x_t^j))$.

Proof. First, $p_4^* = 0$ iff all feasible $\nabla_{x_t} g_{\neg k}(\phi_{\text{sep}}(x_t^j))$ lie in $\text{span}(\nabla_{x_t}\tilde{g}_{\neg k})$, as the objective of Prob. V.6 is just the norm of the coordinates of $\nabla_{x_t} g_{\neg k}(\phi_{\text{sep}}(x_t^j))$ in the basis of the orthogonal complement, i.e., there exists $\beta \in \mathbb{R}$ such that $\nabla_{x_t} g^j_{\neg k} = \beta \nabla_{x_t}\tilde{g}^j_{\neg k}$. Second, if $p_5^* > 0$, then $\nabla_{x_t} g^{j,\top}_{\neg k} \nabla_{x_t}\tilde{g}^j_{\neg k} > 0$, for all $\nabla_{x_t} g^j_{\neg k} \in \text{proj}_{\nabla_{g_{\neg k}}}(\mathcal{F})$. Third, by combining these two results, we have that $\beta \nabla_{x_t}\tilde{g}^{j,\top}_{\neg k} \nabla_{x_t}\tilde{g}^j_{\neg k} > 0$, implying $\beta > 0$, as $\|\nabla_{x_t}\tilde{g}^j_{\neg k}\| > 0$ in order for $t \in \mathbf{t}^j_{\text{tight}}$. Finally, from Thm. V.5, we know $\nabla_{x_t} g^*_{\neg k} = \gamma \nabla_{x_t} g_{\neg k}$, for some $\gamma > 0$ and $\nabla_{x_t} g_{\neg k} \in \text{proj}_{\nabla_{g_{\neg k}}}(\mathcal{F})$; we recover the theorem statement by setting $\alpha = \beta\gamma$. □

Our approach is to use the tight points with a unique KKT-consistent unit normal vector to train our GP constraint (see Sec. 5.4.2); we call these gradients *robustly-identified* and their timesteps as $\mathbf{t}^j_{\text{rob}} \subseteq \mathbf{t}^j_{\text{tight}}$, for all $j = 1, \ldots, N_{\text{dem}}$.

5.4.2 Embedding KKT-based information in a Gaussian process

Let the number of robustly-identified points over all demonstrations be N_{robust}. We collect the constraint states corresponding to the robustly-identified gradients and denote it as $\mathcal{D}_\kappa \doteq \{\phi_{\text{sep}}(x_t^j) \mid t \in \mathbf{t}^j_{\text{rob}}, j \in \{1, \ldots, N_{\text{dem}}\}\} \in \mathbb{R}^{N_{\text{robust}} \times n_c}$. We also collect the robustly-identified gradients $\mathcal{D}_\nabla \doteq \{\nabla_{x_t} g_{\neg k}(\phi_{\text{sep}}(x_t^j)) \mid t \in \mathbf{t}^j_{\text{rob}}, j \in$

$\{1, \ldots, N_{\text{dem}}\}\} \in \mathbb{R}^{N_{\text{robust}} \times n_c}$. Moreover, as the value of the unknown constraint equals zero at all robustly-identified points, we can define a third set $\mathcal{D}_g \doteq \mathbf{0}_{N_{\text{robust}}}$, i.e., the zero vector of size N_{robust}. We wish to learn a GP which is consistent with both the constraint values $\mathcal{D}_g$ as well as the constraint gradients $\mathcal{D}_\nabla$. Note that derivative of a GP is a GP, and the joint distribution of a GP and its derivative is also a GP *Solak et al.* (2002); forming this joint GP provides us an avenue for incorporating both the constraint value and gradient information. Like the derivation of the GP posterior without derivative observations (Sec. 5.3.2), one can derive the posterior distribution conditioned on the training inputs, their derivatives, and the outputs. For brevity, please refer to *Wu et al.* (2018) for detailed derivations. For this joint GP, we can define the training inputs and outputs as $\mathbf{X} = \mathcal{D}_\kappa$ and $\mathbf{Y} = [\mathcal{D}_g, \mathcal{D}_\nabla]$, comprising the dataset $\mathcal{D} = (\mathbf{X}, \mathbf{Y})$, and use the negative marginal log likelihood $-\mathcal{L}_{\text{MLL}}$ (*Rasmussen and Williams*, 2005, Eq. 2.30) to optimize the GP hyperparameters.

A key subtlety is that as the learned constraint is a GP, its constraint value at any given query point is not deterministic; rather, it is sampled from a Gaussian distribution whose mean and variance is determined by the training data and the location of the query point (i.e., $g_{\neg k}(\kappa_m) \sim \mathcal{N}(\mu_m, \sigma_m^2 \mid \mathcal{D}, \kappa_m)$). Moreover, while the demonstrations are guaranteed safe by assumption (i.e., $g_{\neg k}(\kappa_t) \leq 0$ for all t), the stochasticity of the GP values prevents us from enforcing that the demonstrations are safe with probability 1, as the Gaussian has infinite support. Instead, we select a standard deviation threshold ρ for which we want the demonstrations to be safe and add a hinge loss on its violation, where $R = \sum_{j=1}^{N_{\text{dem}}} T^j$:

$$\mathcal{L}_{\text{feas}} = (1/R) \sum_{n=1}^{R} \max(\mu(x_n \mid \mathcal{D}) + \rho\sigma(x_n \mid \mathcal{D}), 0). \tag{5.6}$$

Then the full training loss is $\mathcal{L} = -\mathcal{L}_{\text{MLL}} + \mathcal{L}_{\text{feas}}$.

5.4.3 Planning with the learned constraint

We describe a method for planning with the learned GP constraint. As the GP is probabilistic, so is the boundary of the learned safe set $\mathcal{S}$ (5.5); thus, our planner provides probabilistic, rather than deterministic, safety guarantees. As the dynamics are assumed known, we only consider the uncertainty of the GP constraint in planning. Recall that we wish to connect a start and goal state with a dynamically-feasible trajectory ξ_{xu}^{plan} that satisfies the true constraint $\mathbf{g}_{\neg k}^*(\phi(\xi_{xu}^{\text{plan}}))$ with probability $1 - \delta$. From the assumption (Sec. 5.3.3) that $g_{\neg k}^*$ is drawn from the GP, we achieve this by satisfying the learned constraint $\mathbf{g}_{\neg k}(\phi(\xi_{xu}^{\text{plan}}))$ with probability $1 - \delta$. Unlike previous work in planning under uncertainty *Blackmore et al.* (2006); *Luders et al.* (2010, 2013), we make no structural assumptions on the dynamics or the shape of the constraints.

We modify a constrained kinodynamic RRT *LaValle* (2006) to plan with the learned constraint, though our method can be adapted to other sampling or optimization-based planners. Our planner, which we refer to as Gaussian Process-Chance Constrained RRT (GP-CCRRT), is presented in Alg. V.1. The main novelty of the proposed planner is its GP constraint-checker: when a new node x_q is sampled, instead of checking if x_q satisfies the timestep-independent chance constraint $\Pr(g_{\neg k}(\phi_{\text{sep}}(x_q)) \leq$

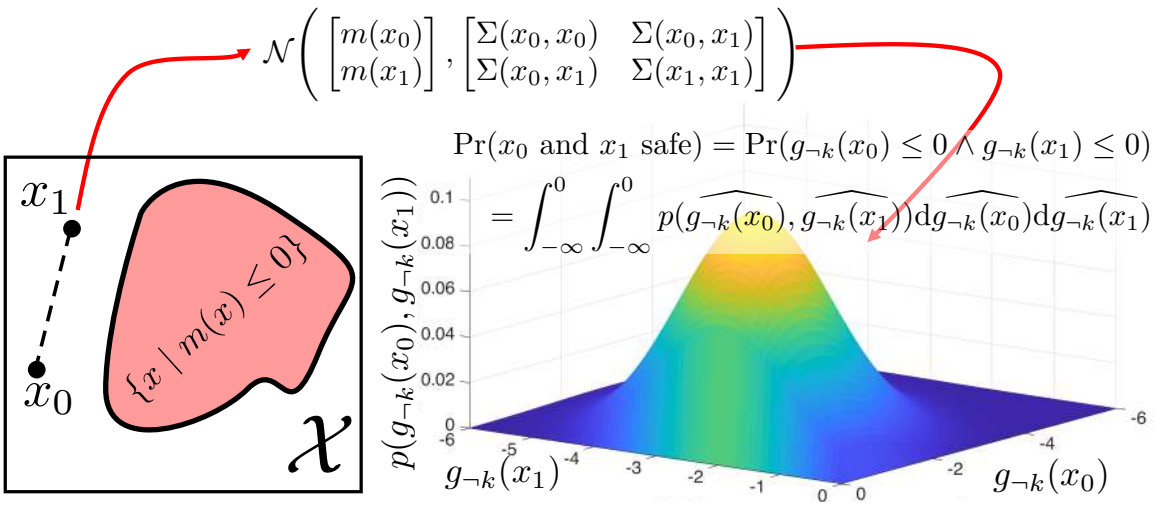


Figure 5.5: Illustration of GP-CCRRT. A candidate length 2 trajectory from the root of the RRT induces a bivariate Gaussian; its safety probability can then be calculated by calculating the CDF of the induced Gaussian.

$0) \geq 1-\delta$, we check if we can connect x_q to the tree by exactly evaluating the joint probability of safety for the full trajectory from the root to the candidate node x_q (line 7-8). Our ability to efficiently compute this probability relies on the Gaussian structure of our learned GP constraint representation. Let the full trajectory from the root to x_q, denoted ξ_q, be length K. Evaluating the learned GP $g_{\neg k}(\cdot)$ at those K points returns the mean and covariance matrix of the predictive posterior distribution, which is a K-variate Gaussian (line 7). Then, the trajectory safety probability, $p^q_{\text{safe}} \doteq \Pr(\bigwedge_{n=1}^{K}(g_{\neg k}(\kappa_n) \leq 0))$, is obtained by integrating the density of this $|K|$-variate Gaussian from $-\infty$ to 0 in each dimension (i.e., the cumulative distribution function, or `cdf`), evaluated at $\mathbf{0}_K$ (line 8). Highly-optimized implementations of the multivariate Gaussian CDF *Genz and Trinh* (2014) enable fast CDF evaluation at planning time. Finally, node x_q is accepted if ξ_q is safe with at least probability $1-\delta$ (line 8). We visualize the GP constraint check in Fig. 5.5.

Algorithm V.1: GP-CCRRT

Input: x_I, x_G, ϵ, goal_bias
1 $\mathcal{T}_x \leftarrow \{x_I\}, \mathcal{T}_u \leftarrow \{\}$
2 **while** True **do**
3 $\quad x_{\text{desired}} \leftarrow$ `SampleState`(goal_bias)
4 $\quad x_{\text{near}} \leftarrow$ `NearestNeighbor`($\mathcal{T}_x$, x_{desired})
5 $\quad \xi_{\text{prev}} \leftarrow$ `PathFromRoot`(x_{near})
6 $\quad x_q, u_q \rightarrow$ `ShootToDesired`($x_{\text{near}}, x_{\text{desired}}$)
7 $\quad \mu, \Sigma \leftarrow g_{\neg k}(\phi(\xi_{\text{prev}} \cup x_q))$
8 $\quad$ **if** `cdf`$(\mathcal{N}(\mu, \Sigma), \mathbf{0}) \geq 1-\delta$ **then** $(\mathcal{T}_x, \mathcal{T}_u) \leftarrow (\mathcal{T}_x, \mathcal{T}_u) \cup (x_q, u_q)$
9 $\quad$ **if** $\|x_q - x_G\| \leq \epsilon$ **then**
10 $\quad\quad$ return $\xi^{\text{plan}}_{xu} \leftarrow$ `ConstructPath`($\mathcal{T}_x$, $\mathcal{T}_u$, x_q)

5.5 Results

We evaluate our method on learning complex, nonlinear constraints demonstrated on a point robot, nonholonomic car, quadrotor, and arm. Please see the video for visualizations. We train all GPs using GPyTorch with an RBF kernel using the Adam optimizer. We obtain demonstrations by solving Prob. V.1 using IPOPT *Wächter and Biegler* (2006). We compare with two baselines. The first, (*Chou et al.*, 2020b, Prob. 4), approximates the unknown constraint as a union of B axis-aligned boxes (as in (*Chou et al.*, 2019, Sec. 4.4)). In the second, we use a neural network (NN) instead of a GP to fit the constraint using the same data; in all examples, the NN has 5 hidden layers of size 256, 512, 1024, 512, and 128 and is trained for 200 epochs with learning rate 5×10^{-5}. To train the NN, we use MSE losses on the target tight constraint values and gradients with a hinge loss that encourages all points to be feasible. We also compute (Table 5.1) how many states are falsely claimed safe ("false safe (FS)") or unsafe ("false unsafe (FU)") by setting $\mathcal{S} = \{\kappa \mid \mu(\kappa \mid \mathcal{D}) + \tau\sigma(\kappa \mid \mathcal{D}) \leq 0\}$ for standard deviations $\tau \in \{0, 1, 2, 2.33\}$. While $\mathcal{S}$ is not used in GP-CCRRT (it uses joint instead of individual safety probabilities), it is a good surrogate for constraint accuracy. Finally, Probs. V.2-V.7 are all solved in 0.5 seconds.

		Our method				Baselines	
		$0\sigma_p$	$1\sigma_p$	$2\sigma_p$	$2.33\sigma_p$	*Chou et al.* (2020b)	NN
Cup	FS (%)	0.004	0.000	0.000	0.000	22.616	52.706
	FU (%)	0.022	1.294	3.532	4.684	5.284	0.000
Car	FS (%)	1.741	0.319	0.071	0.042	5.947	15.555
	FU (%)	0.424	58.761	64.807	66.305	0.117	0.000
Box	FS (%)	3.230	0.462	0.189	0.138	0.000	10.859
	FU (%)	1.648	81.146	86.641	87.190	0.000	0.000
Tree	FS (%)	0.593	0.057	0.004	0.000	14.867	23.427
	FU (%)	0.729	11.108	31.412	37.773	0.160	0.000
Arm	FS (%)	1.403	0.163	0.012	0.003	17.179	15.029
	FU (%)	0.658	57.294	70.644	73.490	0.808	0.151

Table 5.1: GP classification errors (False Safe (FS); False Unsafe (FU)).

2D cup constraint: The purpose of our first example is to demonstrate that our approach can learn complicated unsafe sets with hollow interiors. Consider demonstrations wiping the interior and exterior of a cup. The cup is centered at the origin and has inner and outer radii $\underline{r}$ and $\bar{r}$, respectively. One way to represent the wiping task is to minimize the cumulative distance from the center of the rim over time, i.e., $c(\xi) = \sum_{t=1}^{T} \| \|\chi_t\| - \frac{\bar{r}+\underline{r}}{2} \|_2^2$, subject to point-robot dynamics, control constraints, and nonpenetration with the cup, where $\chi_t = [x_t, y_t]$. In this example, we aim to learn the shape of the cup from the demonstrations (i.e., the unsafe set between the inner and outer cup radii, see Fig. 5.6.A). Given four demonstrations (Fig. 5.6.A), and by training the GP for 500 epochs at learning rate 0.05, we are able to recover the cup shape (Fig. 5.6.B) with very high accuracy (see Table 5.1). In contrast, neither of the baselines can accurately recover the constraint (Table 5.1); the NN fails to accurately fit the constraint gradients, while *Chou et al.* (2020b) fails to accurately fill the interior of the unsafe set (when allocated 5 boxes in its representation).

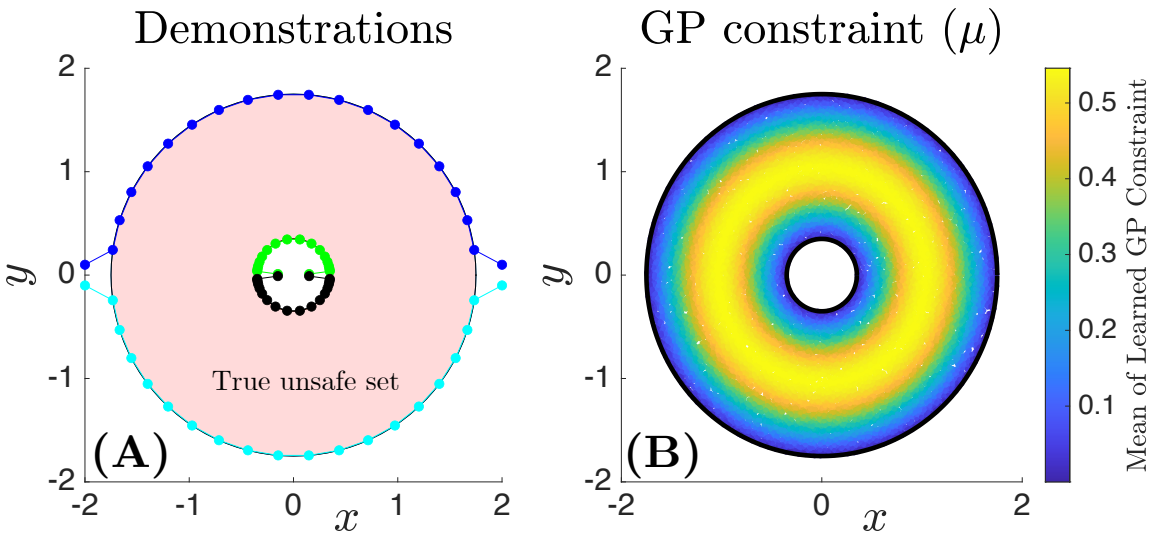


Figure 5.6: 2D hollow cup constraint. (A) Demonstrations and true constraint. (B) Learned GP posterior mean, with true constraint overlaid.

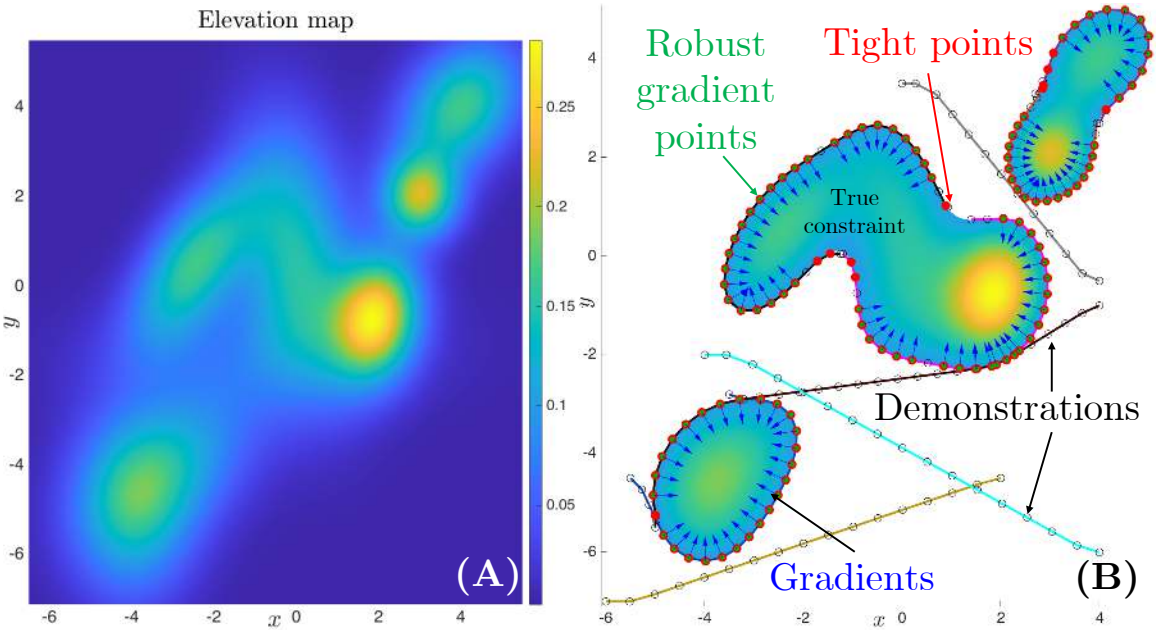


Figure 5.7: 5D car example. **(A)** Hilly terrain map. **(B)** Demonstrations; identified tight points (red); robustly-identified timesteps (green); robustly-identified gradients (blue).

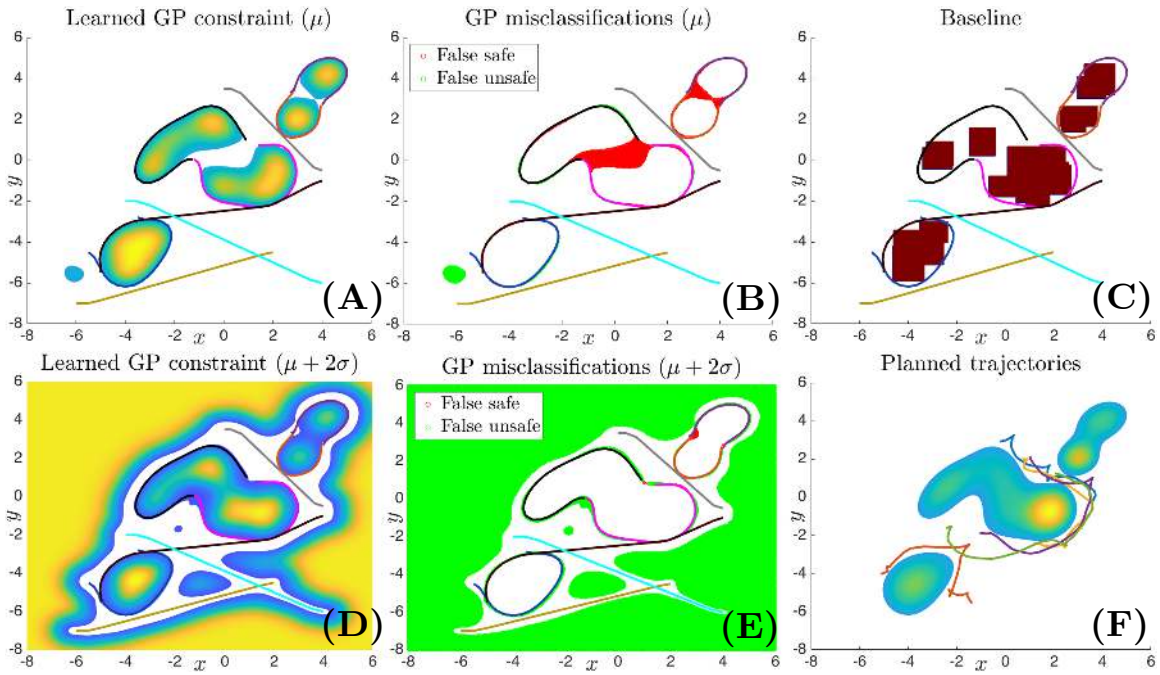


Figure 5.8: 5D car example, learned. **(A)** Learned GP constraint, mean function. **(B)** Mean function misclassifications. **(C)** Constraint learned using baseline *Chou et al.* (2020b). **(D)** Learned GP constraint, buffered by GP uncertainty. **(E)** Buffered misclassifications. **(F)** Plans computed using learned GP constraint.

5D nonholonomic car: We first evaluate our method on a 5D car, showing that can learn a nonlinear, disconnected constraint without prior knowledge. Consider an autonomous vehicle driving on hilly terrain (Fig. 5.7.A) which must stay below a maximum elevation; the corresponding unsafe set (i.e., the subset of the map above the elevation limit) is the filled-in region in Fig. 5.7.B. We use the second-order unicycle dynamics from (*LaValle*, 2006, Eq. 13.46) with a discretization time of $\Delta T = 0.5$. Prior work *Chou et al.* (2018a) studied a similar example; however, in *Chou et al.* (2018a), the map (i.e., the constraint representation) is given, so the only the elevation threshold must be learned. In contrast, we are not given the map, and must learn the representation jointly with the threshold – a much harder problem.

We obtain 9 demonstrations minimizing the xy-space path length, and a control constraint of $\|u_t\|_2^2 \leq 5$ is imposed for all time. Here, ϕ_{sep} maps to the xy-state components. By solving Prob. V.2, we identify tight points (Fig. 5.7.B, red). Next, by solving Probs. V.4-V.7, we find robustly-consistent gradients (Fig. 5.7.B, blue arrows) at a subset of the tight points (Fig. 5.7.B, green). Note the accuracy of Prob. V.2, which identifies that the cyan trajectory is not tight, despite it curving due to dynamical constraints, and correct identification for the black trajectory, which makes and breaks contact with the constraint boundary. The few tight points that are not identified (e.g., near $[-2, 2]$) are where the constraint boundary is flat; thus, the sub-trajectory is optimal despite being on the boundary (Fig. 5.3.C). Note that most tight points also have robustly-identifiable gradients; the exceptions are before/after the system leaves the constraint boundary; this is due to the dynamics, as the car may brake/turn to prevent constraint violation, expanding the set of consistent gradients.

We train the GP for 150 epochs at learning rate 0.08. In Fig. 5.8, we show the GP accuracy and compare with the baseline *Chou et al.* (2020b). Overall, the GP mean faithfully recreates the true unsafe set (Fig. 5.8.A), though it misclassifies (Fig. 5.8.B) the center of the middle and top obstacles; this is as there are few tight points in that area. Still, when buffering the GP with its uncertainty (Fig. 5.8.D-E), the regions which are falsely classified shrink (see Table 5.1), though this is at the cost of conservatively marking much of the map far from the demonstrations as unsafe. This arises from the GP's ability to capture epistemic uncertainty and can actually be desirable as it leads to cautious plans that remain near the data and away from unseen constraints that are inactive on the demonstrations. For the baseline *Chou et al.* (2020b), we use 20 boxes and terminate the optimization after 60 minutes. The result has higher error than the learned GP constraint and fails to capture the constraint shape. The NN baseline is also inaccurate, as it drives the value of most tight points to 0 but fails to fit the gradient data (Table 5.1). Finally, we plan with GP-CCRRT with a safety probability of 0.9; five plans are shown in Fig. 5.8.F, which all satisfy $g^*_{\neg k}(\cdot)$. On average, our planner solves in 3 minutes, with 20 and 50 percent of that time being dedicated to GP posterior and CDF computation, respectively; this can be sped up via lazy checking of the CDF constraint. Overall, this example suggests we can learn nonconvex constraints with minimal *a priori* knowledge.

12D quadrotor: We evaluate our method on two constraint learning tasks on a 12D quadrotor (see (*Chou et al.*, 2021b, Eq. 19) for the dynamics). We first show our method achieves comparable performance with *Chou et al.* (2020b) for learning

constraints that can be represented as a union of boxes. We are given 24 short demonstrations (Fig. 5.9.A) that minimize xyz path length while satisfying a control constraint $\|u_t\|_2^2 \leq 100$, for all t. Moreover, the baseline *Chou et al.* (2020b) is also provided the information that the constraint can be represented as a union of two axis-aligned boxes (thus learning the constraint exactly). We train the GP for 600 epochs at learning rate 0.1, and the learned GP (Fig. 5.9.B) captures the union-of-boxes shape well. Two main inaccuracies are A) the interior of the learned box is hollow (this is expected, as no data can be collected in the obstacle) and B) there are some "ringing effects" (this is caused by the GP, which favors smooth functions, attempting to fit the discontinuous box gradients). Numerically, Table 5.1 shows that the GP misclassifications are low, and moreover, the number of states that are falsely claimed to be safe can be driven near zero by buffering with the GP uncertainty. The GP outperforms the NN baseline, which again struggles to fit the gradient data. Overall, this example suggests our method also performs well where previous methods excel.

Next, we evaluate our approach on learning a complex nonlinear constraint which is well beyond the capability of the baseline *Chou et al.* (2020b). We are given 25 demonstrations (Fig. 5.1.A, black) avoiding collisions with an unknown tree-like obstacle to be learned, which is a union of three ellipsoids (Fig. 5.1.A, blue). Crucially, we lack *a priori* knowledge on the structure of $g^*_{\neg k}(\cdot)$. The dynamics and cost function used are as in *Chou et al.* (2020a) and *Chou et al.* (2020b), respectively, and ϕ_{sep} maps to the xyz-state components. We train the GP for 150 epochs at learning rate 0.08. In comparing with the baseline, we use 6 axis-aligned boxes and time out the optimization after 2 hours.

We visualize our results in Fig. 5.1.B-C. Our learned GP is visually accurate (Fig. 5.1.B), with minor errors (Fig. 5.1.C) where there are no tight points. This is reasonable, as we cannot expect the GP to be accurate far from the data. In contrast, the baseline *Chou et al.* (2020b) is inaccurate (Fig. 5.1.D), failing to cover the upper portion of the obstacle; moreover, the shape is inaccurate due to the limitations of axis-aligned boxes. The NN baseline is also inaccurate, failing to fit the constraint gradients (Table 5.1). Numerical results in Table 5.1 suggest that the GP mean is the most accurate when considering both metrics. As before, the "False Safe" rate can be made smaller at the cost of conservativeness by buffering with the predictive uncertainty. In contrast, the baseline has a high "False Safe" rate, which can lead to constraint violation in planning. We show six plans computed via GP-CCRRT (Fig. 5.1.A, gold), which are safe with probability at least 0.9, and which are safe for the true constraint. On average, planning takes 90 seconds; 55 and 20 percent of this is due to GP posterior and CDF calculations. This example suggests our method scales to complex constraints on high-dimensional systems while requiring minimal prior information.

Planar 3-link manipulator: Finally, we evaluate our method on a kinematic planar 3-link arm. The arm is mounted at the origin and must avoid a blue nonconvex workspace obstacle (Fig. 5.10.D, E). To show that our method can learn complex constraints on articulated robots, we learn the configuration space (C-space) representation of the obstacle (Fig. 5.10.A, blue), which is also nonconvex. We obtain

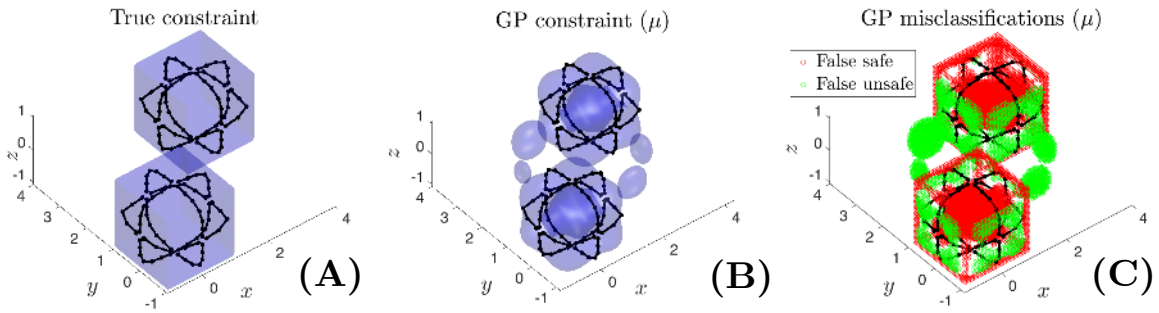


Figure 5.9: 12D quadrotor box example. **(A)** Two box obstacles; demonstrations. **(B)** Learned GP constraint (mean). **(C)** GP misclassifications (mean).

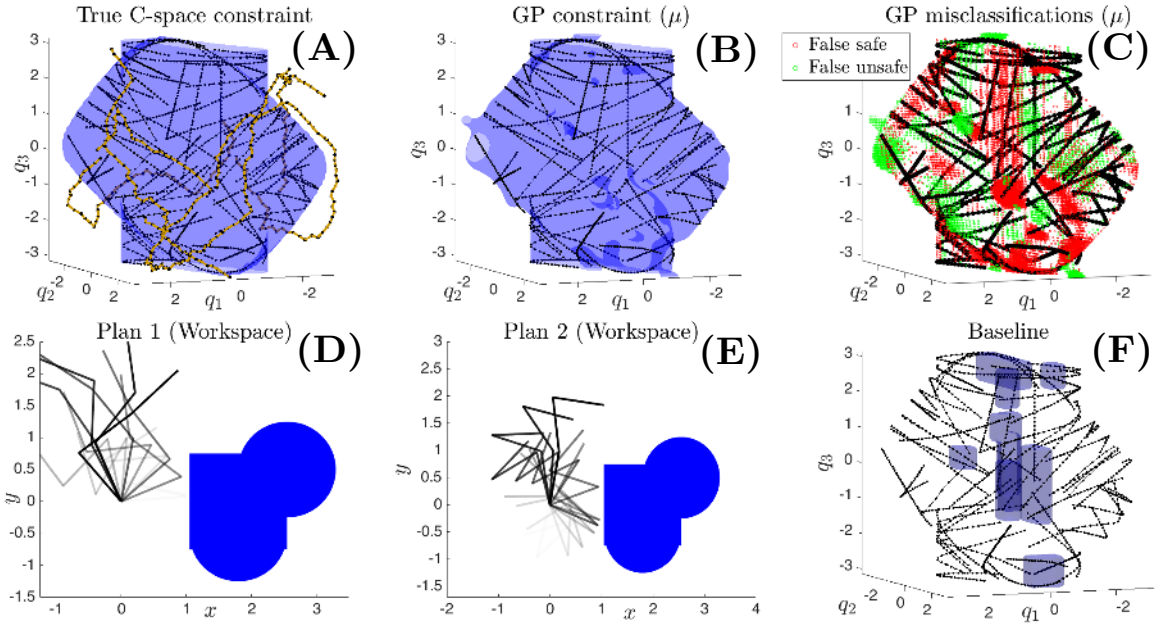


Figure 5.10: 3-link planar arm example, learned. **(A)** True C-space constraint; plans found using GP constraint (gold). **(B)** Learned GP constraint, mean function. **(C)** Mean function misclassifications. **(D-E)** Plans computed using learned GP constraint (workspace). **(F)** Constraint learned with baseline *Chou et al.* (2020b).

50 demonstrations (Fig. 5.10.A, black) which minimize the joint-space path length, i.e., $c(\xi_{xu}) = \sum_{t=1}^{T} \|q_{t+1} - q_t\|^2$ subject to a control constraint $\|q_{t+1} - q_t\| \leq 0.1$, for all t. We train the GP for 150 epochs at learning rate 0.08, obtaining a GP whose posterior mean is visually consistent with the true constraint (Fig. 5.10.B). The posterior mean misclassifications are mostly on the interior of the C-space obstacle (as expected, since no data can be collected there), as well as minor errors on the constraint surface which are further away from the data. We plan via GP-CCRRT, taking two minutes on average, where 40 and 45 percent of the time is due to GP posterior and CDF computations, respectively. We use GP-CCRTT to obtain plans that are safe with probability greater than 0.9; time-lapses of two such plans are shown in Fig. 5.10.D-E. Finally, we evaluate the baseline *Chou et al.* (2020b) with 10 boxes, timing out the optimization after 6 hours. Due to the number of demonstrations and constraint parameters, the baseline struggles (Fig. 5.10.F), returning boxes that do not adequately satisfy the KKT conditions and fail to capture the features of the true constraint. As before, the posterior mean has low "false safe" and "false unsafe" rates, and the "false safe" rate can be reduced via buffering (see Table 5.1). In contrast, the baseline has a much higher "false safe" rate, since it fails to cover most of the unsafe set, though it has a low "false unsafe" rate, since it marks most of the space as safe. The NN baseline also fails to fit the gradient data, leading to low accuracy (Table 5.1). Overall, this example suggests that we can learn non-convex, non-workspace constraints on articulated robots while requiring minimal prior information, which is a necessity for C-space obstacles, which can be unintuitive.

5.6 Discussion and Conclusion

In this chapter, we learn constraints from demonstrations with minimal *a priori* knowledge by finding where the unknown constraint is tight and a scaling of its gradient at those points via the KKT conditions, and then training a GP-represented constraint that is consistent with and generalizes this data. We also show that the Gaussian structure of the GP uncertainty can exploited in an RRT-based planner to compute plans which satisfy the unknown constraint with a specified probability. Our results on a 5D car, 12D quadrotor, and 3-link planar arm show we can learn complex constraints on realistic systems which prior methods cannot handle. We conclude by discussing design choices and future work.

Why use a GP constraint representation? Our learning problem (fitting a function using zero level set data (the tight points) and its gradients at those points) closely relates to fitting manifolds *Sutanto et al.* (2020) and signed distance functions *Park et al.* (2019) to data (though our method differs greatly in how it obtains the data, i.e., the KKT conditions). In both *Sutanto et al.* (2020) and *Park et al.* (2019), neural networks (NN) are used to fit large datasets on the order of $10^3 \sim 10^4$ and 10^5 for *Sutanto et al.* (2020) and *Park et al.* (2019), respectively. In Sec. 5.5, we show that in training the NN with only $\sim 10^2$ tight data points, the NN failed to provide an accurate fit. In contrast, derivative data can be directly embedded via the joint GP, which fits the constraint better with less data (on the order of 10^2). Moreover, GPs

handle constraint uncertainty in a principled way, which is crucial for safe planning. **Limitations and future work**: First, as GPs are non-parametric, dense coverage of the constraint space with tight points is needed to reduce the predictive uncertainty. This results in cautious plans that stay near the training data, and may needlessly restrict the robot. In future work, we will explore semi-parametric models which combine a known, insufficient parameterization with a non-parametric GP to reduce uncertainty. Second, our method assumes demonstrations are precisely locally-optimal, but this is often untrue due to noisy observations or partial observability *Knuth et al.* (2021b); in future work, we will investigate suboptimality models (e.g., *Ziebart et al.* (2008)) that can be used to adjust the confidence in extracted constraint value/gradient data. Third, we wish to extend our method to time-varying (e.g., temporal logic *Chou et al.* (2020c)) constraints. Finally, the scaling of GPs may hamper the learning of high-dimensional constraints; thus, we will explore scalable GP variants (e.g., sparse spectrum GP regression *Rahimi and Recht* (2007)).

CHAPTER VI

Learning Temporal Logic Formulas from Suboptimal Demonstrations

In this chapter, we present a method for learning multi-stage tasks from demonstrations by learning the logical structure and atomic propositions of a consistent linear temporal logic (LTL) formula. The learner is given successful but potentially suboptimal demonstrations, where the demonstrator is optimizing a cost function while satisfying the LTL formula, and the cost function is uncertain to the learner. Our algorithm uses the Karush-Kuhn-Tucker (KKT) optimality conditions of the demonstrations together with a counterexample-guided falsification strategy to learn the atomic proposition parameters and logical structure of the LTL formula, respectively. We provide theoretical guarantees on the conservativeness of the recovered atomic proposition sets, as well as completeness in the search for finding an LTL formula consistent with the demonstrations. We evaluate our method on high-dimensional nonlinear systems by learning LTL formulas explaining multi-stage tasks on a simulated 7-DOF arm and a quadrotor, and show that it outperforms competing methods for learning LTL formulas from positive examples. Finally, we demonstrate that our approach can learn a real-world multi-stage tabletop manipulation task on a physical 7-DOF Kuka iiwa arm. This chapter is based off of the papers *Chou et al.* (2021b) and *Chou et al.* (2020c).

6.1 Introduction

Imagine demonstrating a multi-stage task to a robot arm delivery worker, such as finding and delivering a set of objects from a storage area to some customers (Fig. 6.1). How should the robot understand and generalize the demonstration? One popular method is inverse reinforcement learning (IRL), which assumes a level of optimality on the demonstrations, and aims to learn a reward function that replicates the demonstrator's behavior when optimized *Abbeel and Ng* (2004); *Argall et al.* (2009); *Ng and Russell* (2000); *Ratliff et al.* (2006). Due to this representation, IRL works well on short-horizon goal-directed tasks, but can struggle to scale to multi-stage, constrained tasks *Chou et al.* (2018a); *Krishnan et al.* (2019); *Vazquez-Chanlatte et al.* (2018). Transferring reward functions across environments (e.g., from one storage

Figure 6.1: Multi-stage delivery task: place the soup in an open-top box and deliver it, then deliver the Cheez-Its to a second delivery location. To avoid spills, a pose constraint is enforced while the soup is being delivered in the open-top box.

area to another) can also be difficult, as IRL may overfit to aspects of the training environment. It may instead be fruitful to decouple the high- and low-level task structure, learning a logic-based temporal abstraction of the task that is valid for different environments which can combine low-level, environment-dependent skills. Linear temporal logic (LTL) is well-suited for representing this abstraction, since it can unambiguously specify high-level temporally-extended constraints *Baier and Katoen* (2008) as a function of atomic propositions (APs), which can be used to describe salient low-level state-space regions. To this end, a growing community in controls and anomaly detection has focused on learning linear temporal logic (LTL) formulas to explain trajectory data. However, the vast majority of these methods require both positive and negative examples in order to regularize the learning problem. While this is acceptable in anomaly detection, where one expects to observe formula-violating trajectories, in the context of robotics, it can be unsafe to ask a demonstrator to execute formula-violating behavior, such as dropping a fragile object or crashing into obstacles.

In this chapter, our insight is that by assuming that demonstrators are goal-directed (i.e., that they approximately optimize an objective function that may be uncertain to the learner), we can regularize the LTL learning problem without being provided any formula-violating behavior. In particular, we learn LTL formulas which are parameterized by their high-level logical structure and low-level AP regions, and we show that to do so, it is important to consider demonstration optimality both in terms of the quality of the discrete high-level logical decisions and the continuous low-level control actions. We use the Karush-Kuhn-Tucker (KKT) optimality conditions from continuous optimization to learn the shape of the low-level APs, along with notions of discrete optimality to learn the high-level task structure. We solve a mixed integer linear program (MILP) to jointly recover LTL and cost function parameters which are consistent with the demonstrations. We make the following contributions:

1. We develop a method for time-varying, constrained inverse optimal control, where the demonstrator optimizes a cost function while respecting an LTL formula, where the parameters of the atomic propositions, formula structure, and an uncertain cost function are to be learned. We require only positive demonstrations, can handle demonstration suboptimality, and for fixed formula structure, can extract guaranteed conservative estimates of the AP regions.

2. We develop conditions on demonstrator optimality needed to learn high- and

low-level task structure: AP regions can be learned with discrete feasibility, while logical structure requires various levels of discrete optimality. We develop variants of our method under these different assumptions.

3. We provide theoretical analysis of our method, showing that under mild assumptions, it is guaranteed to return the shortest LTL formula which is consistent with the demonstrations, if one exists. We also prove various results on our method's conservativeness and on formula learnability.

4. We evaluate our method on learning complex LTL formulas demonstrated on nonlinear, high-dimensional systems, show that we can use demonstrations of the same task on different environments to learn shared high-level task structure, and show that we outperform previous approaches.

6.2 Preliminaries and Problem Statement

We consider discrete-time nonlinear systems

$$x_{t+1} = f(x_t, u_t, t),$$

with state $x \in \mathcal{X}$ and control $u \in \mathcal{U}$, where we denote state/control trajectories of the system as $\xi_{xu} \doteq (\xi_x, \xi_u)$.

We use linear temporal logic (LTL) *Baier and Katoen* (2008), which augments standard propositional logic to express properties holding on trajectories over (potentially infinite) periods of time. In this chapter, we will be given finite-length trajectories demonstrating tasks that can be completed in finite time. To ensure that the formulas we learn can be evaluated on finite trajectories, we focus on learning formulas, given in positive normal form, which are described in a parametric temporal logic similar to bounded LTL *Jha et al.* (2009), and which can be written with the grammar

$$\varphi ::= p \mid \neg p \mid \varphi_1 \vee \varphi_2 \mid \varphi_1 \wedge \varphi_2 \mid \Box_{[t_1,t_2]}\varphi \mid \varphi_1 \,\mathcal{U}_{[t_1,t_2]}\, \varphi_2, \tag{6.1}$$

where $p \in \mathcal{P} \doteq \{p_i\}_{i=1}^{N_{\text{AP}}}$ are atomic propositions (APs) and N_{AP} is known to the learner. $t_1 \leq t_2$ are nonnegative integers. Here, $\neg p$ denotes the negation of atomic proposition p, the "or" operator $\varphi_1 \vee \varphi_2$ denotes the disjunction of formulas φ_1 and φ_2, the "and" operator $\varphi_1 \wedge \varphi_2$ denotes the conjunction of formulas φ_1 and φ_2, the "bounded-time always" operator $\Box_{[t_1,t_2]}\varphi$ denotes that φ "always" has to hold over the interval $[t_1, t_2]$, and the "bounded-time until" operator $\varphi_1 \,\mathcal{U}_{[t_1,t_2]}\, \varphi_2$ denotes that φ_2 must eventually hold during the interval $[t_1, t_2]$, and φ_1 must hold for all timesteps prior to that. Due to the positive normal form structure, negation can only appear directly before APs. Let the size of the grammar be $N_{\text{g}} = N_{\text{AP}} + N_{\text{o}}$, where N_{o} is the number of temporal/boolean operators in the grammar. A useful derived operator is "bounded-time eventually" $\Diamond_{[t_1,t_2]}\varphi \doteq \top \,\mathcal{U}_{[t_1,t_2]}\, \varphi$, which denotes that a formula φ eventually has to hold during the interval $[t_1, t_2]$.

In this chapter, we will consider LTL formulas $\varphi(\theta^s, \theta^p)$ that are parameterized by $\theta^s \in \Theta^s$, which encode the logical and temporal structure of the formula, and by $\theta^p \doteq \{\theta_i^p\}_{i=1}^{N_{\text{AP}}}$, where $\theta_i^p \in \Theta_i^p$ defines the shape of the region where p_i holds. Furthermore, we will consider APs of the form: $x \models p_i \Leftrightarrow \mathbf{g}_i(\eta_i(x), \theta_i^p) \leq \mathbf{0}$, where $\eta_i(\cdot) : \mathcal{X} \to \mathcal{C}$ is a known nonlinear function, $\mathbf{g}_i(\cdot,\cdot) \doteq [g_{i,1}(\cdot,\cdot), \ldots, g_{i,N_i^{\text{ineq}}}(\cdot,\cdot)]^\top$ is a vector-valued parametric function, and $\mathcal{C}$ is the space in which the AP constraint is evaluated, elements of which are denoted *constraint states* $\kappa \in \mathcal{C}$.

To show how this notation maps onto a concrete robotics example, consider a 7-DOF arm. We can define the state x as the joint angles, the control u as the joint velocities, the constraint state κ as the end effector pose, and the mapping from the state to constraint state space $\eta : \mathcal{X} \to \mathcal{C} \subseteq \mathbb{R}^3$ as the forward kinematics, mapping from joint space to workspace. One possible atomic proposition is $x \models p \Leftrightarrow \mathbf{g}(\eta(x), \theta^p) = A\eta(x) - \theta^p \leq \mathbf{0}$, where $A = [I_{3\times3}, -I_{3\times3}]^\top$ and $I_{n\times n}$ is the $n \times n$ identity matrix. This atomic proposition p is satisfied if the end effector position is contained within an axis-aligned rectangle in the workspace with extents described by $\theta^p = [\bar{x}, \bar{y}, \bar{z}, -\underline{x}, -\underline{y}, -\underline{z}]$, where $\bar{x}$, $\bar{y}$, and $\bar{z}$ denote the upper extents in the x-, y-, and z-dimensions, and $\underline{x}$, $\underline{y}$, and $\underline{z}$ denote the lower extents in the x-, y-, and z-dimensions. Finally, we can write an LTL formula $\Diamond_{[t_1,t_2]} p$ to enforce that all trajectories must satisfy this workspace constraint at some point between time t_1 and t_2.

We formalize the discussion above by defining the semantics, which describe the satisfaction of an LTL formula φ by a trajectory ξ_{xu}. Specifically, we denote the satisfaction of a formula φ on a finite-duration trajectory ξ_{xu} of total duration T, evaluated at time $t \in \{1, 2, \ldots, T\}$, as $(\xi_{xu}, t) \models \varphi$. Then, the formula satisfaction is defined recursively in the formal semantics (6.2). We will write $\varphi \models \xi_{xu}$ as shorthand for $(\xi_{xu}, 1) \models \varphi$. We emphasize that since we consider discrete-time trajectories, a time interval $[t_1, t_2]$ is evaluated only on integer time instants $\{t_1, t_1 + 1, \ldots, t_2\}$; this is made concrete in (6.2).

We consider tasks that involve optimizing a parametric cost function (encoding efficiency concerns, etc.), while satisfying an LTL formula $\varphi(\theta^s, \theta^p)$ (encoding constraints for task completion):

$$
\begin{aligned}
(\xi_{xu}, t) \models p_i \quad &\Leftrightarrow \quad \mathbf{g}_i(\eta_i(x_t), \theta_i^p) \leq \mathbf{0} \\
(\xi_{xu}, t) \models \neg p_i \quad &\Leftrightarrow \quad \neg((\xi_{xu}, t) \models p_i) \\
(\xi_{xu}, t) \models \varphi_1 \vee \varphi_2 \quad &\Leftrightarrow \quad (\xi_{xu}, t) \models \varphi_1 \vee (\xi_{xu}, t) \models \varphi_2 \\
(\xi_{xu}, t) \models \varphi_1 \wedge \varphi_2 \quad &\Leftrightarrow \quad (\xi_{xu}, t) \models \varphi_1 \wedge (\xi_{xu}, t) \models \varphi_2 \\
(\xi_{xu}, t) \models \Box_{[t_1,t_2]} \varphi \quad &\Leftrightarrow \quad (t + t_1 \leq T) \wedge (\forall \tilde{t} \in [t + t_1, \min(t + t_2, T)], (\xi_{xu}, \tilde{t}) \models \varphi) \\
(\xi_{xu}, t) \models \varphi_1 \mathcal{U}_{[t_1,t_2]} \varphi_2 \quad &\Leftrightarrow \quad (t + t_1 \leq T) \wedge (\exists \tilde{t} \in [t + t_1, \min(t + t_2, T)] \\
&\qquad \text{s.t. } (\xi_{xu}, \tilde{t}) \models \varphi_2) \ \wedge (\forall \check{t} \in [t, \tilde{t} - 1], (\xi_{xu}, \check{t}) \models \varphi_1) \\
(\xi_{xu}, t) \models \Diamond_{[t_1,t_2]} \varphi \quad &\Leftrightarrow \quad (t + t_1 \leq T) \wedge (\exists \tilde{t} \in [t + t_1, \min(t + t_2, T)] \\
&\qquad \text{s.t. } (\xi_{xu}, \tilde{t}) \models \varphi)
\end{aligned}
\tag{6.2}
$$

Problem VI.1 (Demonstrator's forward problem)**.**

$$\begin{array}{ll} \underset{\xi_{xu}}{\text{minimize}} & c(\xi_{xu}, \theta^c) \\ \text{subject to} & \xi_{xu} \models \varphi(\theta^s, \theta^p) \\ & \bar{\eta}(\xi_{xu}) \in \bar{\mathcal{S}} \subseteq \mathcal{C} \end{array}$$

where $c(\cdot, \theta^c)$ is a potentially non-convex cost function, parameterized by $\theta^c \in \Theta^c$. Any *a priori* known constraints are encoded in $\bar{\mathcal{S}}$, where $\bar{\eta}(\cdot)$ is known. In this chapter, we encode in $\bar{\mathcal{S}}$ the system dynamics, start state, and if needed, a goal state separate from the APs.

Next, to ease notation, we will define $G_i(\kappa, \theta_i^p) \doteq \max_{m \in \{1,\dots,N_i^{\text{ineq}}\}} \left(g_{i,m}(\kappa, \theta_i^p)\right)$. Define the subset of $\mathcal{C}$ where p_i holds/does not hold, as

$$\mathcal{S}_i(\theta_i^p) \doteq \{\kappa \mid G_i(\kappa, \theta_i^p) \leq 0\} \tag{6.3}$$

$$\mathcal{A}_i(\theta_i^p) \doteq \text{cl}(\{\kappa \mid G_i(\kappa, \theta_i^p) > 0\}) = \text{cl}(\mathcal{S}_i(\theta_i^p)^c) \tag{6.4}$$

To ensure that Problem VI.1 admits an optimum, we have defined $\mathcal{A}_i(\theta_i^p)$ to be closed; that is, states on the boundary of an AP can be considered either inside or outside. For these boundary states, our learning algorithm can automatically detect if the demonstrator intended to visit or avoid the AP (cf. Sec. 6.3.2).

We are given N_s demonstrations $\{\xi_j^{\text{dem}}\}_{j=1}^{N_s}$ of duration T_j, which approximately solve Prob. VI.1, in that they are feasible (satisfy the LTL formula and known constraints) and achieve a possibly suboptimal cost. Note that Prob. VI.1 can be modeled with continuous (ξ_{xu}) and boolean decision variables (referred to collectively as $\mathbf{Z}$) *Wolff et al.* (2014); the boolean variables determine the high-level plan, constraining the trajectory to obey boolean decisions that satisfy $\varphi(\theta^s, \theta^p)$, while the continuous component synthesizes a low-level trajectory implementing the plan. We will use different assumptions of demonstrator optimality on the continuous/boolean parts of the problem, depending on if θ^p (Sec. 6.3), θ^s (Sec. 6.4), or θ^c (Sec. 6.5) are being learned, discuss extensions and variants of these methods (Sec. 6.6), and discuss how these different degrees of optimality can affect the learnability of LTL formulas (Sec. 6.7).

Our goal is to learn the unknown structure θ^s and AP parameters θ^p of the LTL formula $\varphi(\theta^s, \theta^p)$, as well as unknown cost function parameters θ^c, given demonstrations $\{\xi_j^{\text{dem}}\}_{j=1}^{N_s}$ and the *a priori* known safe set $\bar{\mathcal{S}}$.

6.3 Learning Atomic Proposition Parameters (θ^p)

We develop methods for learning unknown AP parameters θ^p when the cost function parameters θ^c and formula structure θ^s are known. We first review recent results *Chou et al.* (2020b) on learning time-invariant constraints via the KKT conditions (Sec. 6.3.1). Then, we show how the framework can be extended to learn θ^p (Sec. 6.3.2), and develop a method for extracting states which are guaranteed to satisfy or to violate p_i (Sec. 6.3.3). In all of Sec. 6.3, we will assume that demonstrations are

locally-optimal for the continuous component and *feasible for the discrete component.*

6.3.1 Learning time-invariant constraints via KKT

Consider a simplified variant of Prob. VI.1 that only involves always satisfying a single AP; this reduces Prob. VI.1 to a standard trajectory optimization problem:

$$\begin{array}{ll} \underset{\xi_{xu}}{\text{minimize}} & c(\xi_{xu}) \\ \text{subject to} & \mathbf{g}(\eta(x), \theta^p) \leq \mathbf{0}, \quad \forall x \in \xi_{xu} \\ & \bar{\eta}(\xi_{xu}) \in \bar{\mathcal{S}} \subseteq \mathcal{C} \end{array} \tag{6.5}$$

To ease notation, θ^c is assumed known in Sec. 6.3-6.4 and reintroduced in Sec. 6.5. Suppose we rewrite the constraints of (6.5) as $\mathbf{h}^k(\eta(\xi_{xu})) = \mathbf{0}$, $\mathbf{g}^k(\eta(\xi_{xu})) \leq \mathbf{0}$, and $\mathbf{g}^{\neg k}(\eta(\xi_{xu}), \theta^p) \leq \mathbf{0}$, where k and $\neg k$ group together the known and unknown constraints, respectively. Then, with Lagrange multipliers λ and ν, the KKT conditions (first-order necessary conditions for local optimality *Boyd and Vandenberghe* (2004)) of the jth demonstration ξ_j^{dem}, denoted $\text{KKT}(\xi_j^{\text{dem}})$ are as written in (6.6),

---

$\text{KKT}(\xi_j^{\text{dem}})$:

$$\text{Primal feasibility:} \quad \mathbf{h}^k(\eta(x_t^j)) = \mathbf{0}, \quad t = 1, \ldots, T_j \tag{6.6a}$$

$$\mathbf{g}^k(\eta(x_t^j)) \leq \mathbf{0}, \quad t = 1, \ldots, T_j \tag{6.6b}$$

$$\mathbf{g}^{\neg k}(\eta(x_t^j), \theta^p) \leq \mathbf{0}, \quad t = 1, \ldots, T_j \tag{6.6c}$$

$$\text{Lagrange multiplier nonnegativity:} \quad \boldsymbol{\lambda}_t^{j,k} \geq \mathbf{0}, \quad t = 1, \ldots, T_j \tag{6.6d}$$

$$\boldsymbol{\lambda}_t^{j,\neg k} \geq \mathbf{0}, \quad t = 1, \ldots, T_j \tag{6.6e}$$

$$\text{Complementary slackness:} \quad \boldsymbol{\lambda}_t^{j,k} \odot \mathbf{g}^k(\eta(x_t^j)) = \mathbf{0}, \quad t = 1, \ldots, T_j \tag{6.6f}$$

$$\boldsymbol{\lambda}_t^{j,\neg k} \odot \mathbf{g}^{\neg k}(\eta(x_t^j), \theta^p) = \mathbf{0}, \quad t = 1, \ldots, T_j \tag{6.6g}$$

$$\begin{aligned} \text{Stationarity:} \quad & \nabla_{x_t} c(\xi_j^{\text{dem}}) + \boldsymbol{\lambda}_t^{j,k\top} \nabla_{x_t} \mathbf{g}^k(\eta(x_t^j)) \\ & + \boldsymbol{\lambda}_t^{j,\neg k\top} \nabla_{x_t} \mathbf{g}^{\neg k}(\eta(x_t^j), \theta^p) \\ & + \boldsymbol{\nu}_t^{j,k\top} \nabla_{x_t} \mathbf{h}^k(\eta(x_t^j)) = \mathbf{0}, \quad t = 1, \ldots, T_j \end{aligned} \tag{6.6h}$$

where $\odot$ denotes elementwise multiplication. Intuitively, primal feasibility ensures that the demonstrations satisfy the learned constraint, complementary slackness encodes that a Lagrange multiplier for some constraint can only be nonzero if that constraint is active, and stationarity encodes that the cost cannot be locally improved without violating a constraint.

We vectorize the multipliers $\boldsymbol{\lambda}_t^{j,k} \in \mathbb{R}^{N_k^{\text{ineq}}}$, $\boldsymbol{\lambda}_t^{j,\neg k} \in \mathbb{R}^{N_{\neg k}^{\text{ineq}}}$, and $\boldsymbol{\nu}_t^{j,k} \in \mathbb{R}^{N_k^{\text{ineq}}}$, i.e.,

$\boldsymbol{\lambda}_t^{j,k} = [\lambda_{t,1}^{j,k}, \ldots, \lambda_{t,N_{\text{ineq}}^k}^{j,k}]^\top$. We drop (6.6a)-(6.6b), as they involve no decision variables. Then, we can find a constraint which makes the N_s demonstrations locally-optimal by finding a θ^p that satisfies the KKT conditions for each demonstration:

Problem VI.2 (Inverse KKT problem, exact)**.**

$$\begin{aligned} \text{find} \quad & \theta^p, \{\boldsymbol{\lambda}_t^{j,k}, \boldsymbol{\lambda}_t^{j,\neg k}, \boldsymbol{\nu}_t^{j,k}\}_{t=1}^{T_j}, \quad j = 1, ..., N_s \\ \text{subject to} \quad & \{\text{KKT}(\xi_j^{\text{dem}})\}_{j=1}^{N_s} \end{aligned}$$

If the demonstrations are only approximately locally-optimal, Prob. VI.2 may become infeasible. In this case, we can relax stationarity and complementary slackness to cost penalties:

Problem VI.3 (Inverse KKT problem, suboptimal)**.**

$$\begin{aligned} \underset{\theta^p, \boldsymbol{\lambda}_t^{j,k}, \boldsymbol{\lambda}_t^{j,\neg k}, \boldsymbol{\nu}_t^{j,k}}{\text{minimize}} \quad & \sum_{j=1}^{N_s} \left(\|\text{stat}(\xi_j^{\text{dem}})\|_1 + \|\text{comp}(\xi_j^{\text{dem}})\|_1 \right) \\ \text{subject to} \quad & \text{(6.6c)} - \text{(6.6e)}, \ \forall \xi_j^{\text{dem}}, \ j = 1, \ldots, N_s \end{aligned}$$

where $\text{stat}(\xi_j^{\text{dem}})$ denotes the left hand side (LHS) of Eq. (6.6h) and $\text{comp}(\xi_j^{\text{dem}})$ denotes the concatenated LHSs of Eqs. (6.6f) and (6.6g). Please see Sec. 6.6.4 for more discussion on the effect of demonstration suboptimality on learning θ^p. Note that while we have written Prob. VI.2-VI.3 for general constraint parameterizations, not all parameterizations admit computationally-tractable inverse KKT problems. For some constraint parameterizations (e.g., unions of boxes or ellipsoids (*Chou et al.* (2020b), Chapter IV)), Prob. VI.2-VI.3 are MILP-representable[1] and can be efficiently solved; we consider such parameterizations in further detail in Sec. 6.3.2. In the experiments of this chapter, we focus on constraints which are parameterized as axis-aligned boxes in the constraint space $\mathcal{C} \subseteq \mathbb{R}^c$, i.e., $\mathbf{g}(\eta(x), \theta^p) \leq \mathbf{0} \Leftrightarrow A\eta(x) - \theta^p \leq \mathbf{0}$, where $A = [I_{c\times c}, -I_{c\times c}]^\top$ and $\theta^p = [\bar{x}_1, \ldots, \bar{x}_c, \underline{x}_1, \ldots, \underline{x}_c]^\top$ contains the upper extents $\bar{x}_1, \ldots, \bar{x}_c$ and lower extents $\underline{x}_1, \ldots, \underline{x}_c$ of the box in each coordinate.

6.3.2 Modifying KKT for multiple atomic propositions

Having built intuition with the single AP case, we return to Prob. VI.1 and discuss how the KKT conditions change in the multiple-AP setting. We first adjust the primal feasibility condition (6.6c). Recall from Sec. 6.2 that we can solve Prob. VI.1 by finding a continuous trajectory ξ_{xu} and a set of boolean variables $\mathbf{Z}$ enforcing that $\xi_{xu} \models \varphi(\theta^s, \theta^p)$. For each ξ_j^{dem}, let $\mathbf{Z}^j(\theta_i^p) \in \{0,1\}^{N_{\text{AP}} \times T_j}$, and let the (i,t)th index $Z_{i,t}^j(\theta_i^p)$ indicate if on ξ_j^{dem}, constraint state $\kappa_t \models p_i$ for parameters θ_i^p; that is,

[1] This problem can also be represented and solved with satisfiability modulo theories (SMT) solvers.

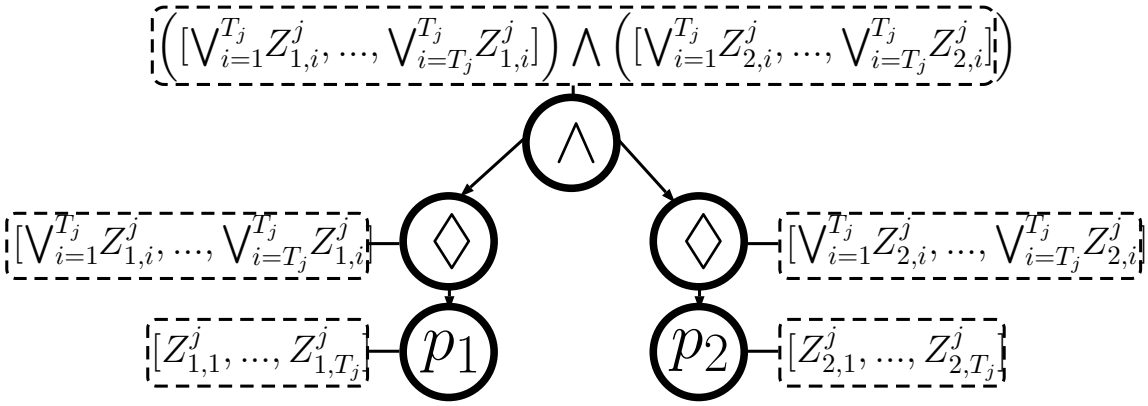


Figure 6.2: A directed acyclic graph (DAG) model of the LTL formula $\varphi = (\Diamond_{[0,T_j-1]} p_1) \wedge (\Diamond_{[0,T_j-1]} p_2)$ (eventually satisfy p_1 and eventually satisfy p_2). The DAG representation can be interpreted as a parse tree for φ (cf. Sec. 6.4.1). The T_j boolean values for each node represent the truth value of the formula associated with the DAG subtree when evaluated on ξ_j^{dem}, starting at times $t = 1, \ldots, T_j$, respectively. Each $\xi_j^{\text{dem}} \models \varphi$ iff the first entry at the root node, $(\bigvee_{i=1}^{T_j} Z_{1,i}^j) \bigwedge (\bigvee_{i=1}^{T_j} Z_{2,i}^j)$, is true.

$$\begin{aligned} Z_{i,t}^j(\theta_i^p) = 1 \Leftrightarrow \kappa_t \in \mathcal{S}_i(\theta_i^p), \\ Z_{i,t}^j(\theta_i^p) = 0 \Leftrightarrow \kappa_t \in \mathcal{A}_i(\theta_i^p). \end{aligned} \tag{6.7}$$

Since LTL operators have equivalent boolean encodings *Wolff et al.* (2014), the truth value of $\varphi(\theta^s, \theta^p)$ can be evaluated as a function of $\mathbf{Z}^j$, θ^p, and θ^s, denoted as $\Phi(\mathbf{Z}^j, \theta^p, \theta^s)$ (we suppress θ^s, as it is assumed known for now). For example, consider the LTL formula $\varphi(\theta^s, \theta^p) = (\Diamond_{[0,T_j-1]} p_1) \wedge (\Diamond_{[0,T_j-1]} p_2)$, which enforces that the system must eventually satisfy p_1 and eventually satisfy p_2. Two trajectories which satisfy this formula are shown in Fig. 6.3. We can evaluate the truth value of $\varphi(\theta^s, \theta^p)$ on ξ_j^{dem} by calculating $\Phi(\mathbf{Z}^j, \theta^p) = (\bigvee_{t=1}^{T_j} Z_{1,t}^j(\theta_1^p)) \wedge (\bigvee_{t=1}^{T_j} Z_{2,t}^j(\theta_2^p))$ (cf. Fig. 6.2). Boolean encodings of common temporal and logical operators can be found in *Biere et al.* (2006). Enforcing that $Z_{i,t}^j(\theta_i^p)$ satisfies (6.7) can be done with a big-M formulation and binary variables $\mathbf{s}_{i,t}^j \in \{0,1\}^{N_i^{\text{ineq}}}$ *Bertsimas and Tsitsiklis* (1997):

$$\begin{aligned} \mathbf{g}_i(\kappa_t^j, \theta_i^p) &\leq M(\mathbf{1}_{N_i^{\text{ineq}}} - \mathbf{s}_{i,t}^j) \\ \mathbf{1}_{N_i^{\text{ineq}}}^\top \mathbf{s}_{i,t}^j - N_i^{\text{ineq}} &\leq M Z_{i,t}^j - M_\epsilon \\ \mathbf{g}_i(\kappa_t^j, \theta_i^p) &\geq -M \mathbf{s}_{i,t}^j \\ \mathbf{1}_{N_i^{\text{ineq}}}^\top \mathbf{s}_{i,t}^j - N_i^{\text{ineq}} &\geq -M(1 - Z_{i,t}^j) \end{aligned} \tag{6.8}$$

where $\mathbf{1}_d$ is a d-dimensional vector of ones, M is a large positive number, and $M_\epsilon \in (0,1)$. In practice, M and M_ϵ can be carefully chosen to improve the solver's performance. Note that $s_{i,m,t}^j$, the mth component of $\mathbf{s}_{i,t}^j$, encodes if κ_t^j satisfies a negated $g_{i,m}(\kappa_t^j, \theta_i^p)$, i.e., if $s_{i,m,t}^j = 1$ or 0, then κ_t^j satisfies $g_{i,m}(\kappa_t^j, \theta_i^p) \leq$ or ≥ 0. We can more compactly rewrite the constraint enforced on the demonstrations as

$\mathbf{g}_i(\kappa_t^j, \theta_i^p) \odot (2\mathbf{s}_{i,t}^j - \mathbf{1}_{N_i^{\text{ineq}}}) \leq \mathbf{0}$ for each i, t; we use this form to adapt the remaining KKT conditions. While enforcing (6.8) is hard in general, if $\mathbf{g}_i(\kappa, \theta_i^p)$ is affine in θ_i^p for fixed κ, (6.8) is MILP-representable; henceforth, we assume $\mathbf{g}_i(\kappa, \theta_i^p)$ is of this form. Note that this can still describe non-convex regions in the constraint space, as the dependency on κ can be nonlinear.

As a concrete example, for the blue trajectory in Fig. 6.3, $\mathbf{Z}_1 = [0, 1, 0, 0, 0]$ and $\mathbf{Z}_2 = [0, 0, 0, 1, 0]$. Consider the first AP p_1. Here, since p_1 is a box in the state space, $g_{1,m}(\kappa_t, \theta_1^p) \leq 0$ can be written as $x_{t,m} - \theta_{1,m}^p \leq 0$, where $\theta_{1,m}^p$ defines the offset for the mth hyperplane that defines the boundary of the box for AP p_1. Then, $s_{1,m,t}$ determines if the polarity of halfspace constraint m is flipped at time t on the blue trajectory.

To modify complementary slackness (6.6g) for the multi-AP case, we note that the elementwise product in (6.6g) is MILP-representable:

$$\begin{gathered}\left[\boldsymbol{\lambda}_{i,t}^{j,\neg k},\ -\mathbf{g}_i(\kappa_t^j, \theta_i^p) \odot (2\mathbf{s}_{i,t}^j - \mathbf{1}_{N_i^{\text{ineq}}})\right] \leq M\mathbf{Q}_{i,t}^j \\ \mathbf{Q}_{i,t}^j \mathbf{1}_2 \leq \mathbf{1}_{N_i^{\text{ineq}}}\end{gathered} \tag{6.9}$$

where $\mathbf{Q}_{i,t}^j \in \{0,1\}^{N_i^{\text{ineq}} \times 2}$. Intuitively, (6.9) enforces that either 1) the Lagrange multiplier is zero and the constraint is inactive, i.e., $g_{i,m}(\kappa, \theta_i^p) \in [-M, 0]$ if $s_{i,m,t}^j = 1$ or $g_{i,m}(\kappa, \theta_i^p) \in [0, M]$ if $s_{i,m,t}^j = 0$, 2) the Lagrange multiplier is nonzero and $g_{i,m}(\kappa_t, \theta_i^p) = 0$, or both; the value of $\mathbf{Q}$ toggles between these options. The stationarity condition (6.6h) must also be modified to consider whether a particular constraint is negated; this can be done by modifying the second line of (6.6h) to terms of the form $\left(\boldsymbol{\lambda}_{i,t}^{j,\neg k\top} \odot (2\mathbf{s}_{i,t}^j - 1)\right) \nabla_{x_t} \mathbf{g}_i^{\neg k}(\eta(x_t), \theta^p)$. The KKT conditions for the multi-AP case, denoted $\text{KKT}_{\text{LTL}}(\xi_j^{\text{dem}})$, then can be written as in (6.10).

As mentioned in Sec. 6.2, if κ_t^j lies on the boundary of AP i, the KKT conditions will automatically determine if $\kappa_t^j \in \mathcal{S}_i(\theta_i^p)$ or $\kappa_t^j \in \mathcal{A}_i(\theta_i^p)$ based on whichever option enables $\mathbf{s}_{i,t}^j$ to take values that satisfy (6.10). To summarize, our approach is to (A) find $\mathbf{Z}^j$, which determines the feasibility of ξ_j^{dem} for $\varphi(\theta^s, \theta^p)$, (B) find $s_{i,m,t}^j$, which link the value of $\mathbf{Z}^j$ from the AP-containment level (i.e., $\kappa_t^j \in \mathcal{S}_i(\theta_i^p)$) to the single-constraint level (i.e., $g_{i,m}(\kappa_t^j, \theta_i^p) \leq 0$), and (C) enforce that ξ_j^{dem} satisfies the KKT conditions for the continuous optimization problem defined by θ^p and fixed values of $\mathbf{s}_{i,t}^j$. Finally, we can write the problem of recovering θ^p for a fixed θ^s as:

Problem VI.4 (Learning θ^p, for fixed template)**.**

$$\begin{aligned}\text{find} \quad & \theta^p, \boldsymbol{\lambda}_t^{j,k}, \boldsymbol{\lambda}_{i,t}^{j,\neg k}, \boldsymbol{\nu}_t^{j,k}, \mathbf{s}_{i,t}^j, \mathbf{Q}_{i,t}^j, \mathbf{Z}^j,\ \forall i, j, t \\ \text{subject to} \quad & \{\text{KKT}_{\text{LTL}}(\xi_j^{\text{dem}})\}_{j=1}^{N_s}\end{aligned}$$

We can also encode prior knowledge in Prob. VI.4, e.g., known AP labels or a prior on θ_i^p, which we discuss in Sec. 6.6.1.

$\text{KKT}_{\text{LTL}}(\xi_j^{\text{dem}})$:

$$\text{Primal feasibility:} \quad \text{Equations (6.6a)} - \text{(6.6b)}, \quad t = 1, \dots, T_j \tag{6.10a}$$
$$\text{Equation (6.8)}, \quad i = 1, \dots, N_{\text{AP}}, \ t = 1, \dots, T_j \tag{6.10b}$$

$$\text{Lagrange multiplier nonnegativity:} \quad \text{Equation (6.6d)}, \quad t = 1, \dots, T_j \tag{6.10c}$$
$$\boldsymbol{\lambda}_{i,t}^{j,\neg k} \geq \mathbf{0}, \quad i = 1, \dots, N_{\text{AP}}, \ t = 1, \dots, T_j \tag{6.10d}$$

$$\text{Complementary slackness:} \quad \text{Equation (6.6f)}, \quad t = 1, \dots, T_j \tag{6.10e}$$
$$\text{Equation (6.9)}, \quad i = 1, \dots, N_{\text{AP}}, \ t = 1, \dots, T_j \tag{6.10f}$$

$$\begin{aligned} \text{Stationarity:} \quad & \nabla_{x_t} c(\xi_j^{\text{dem}}) + \boldsymbol{\lambda}_t^{j,k\top} \nabla_{x_t} \mathbf{g}^k(\eta(x_t^j)) \\ & + \sum_{i=1}^{N_{\text{ineq}}} \left[(\boldsymbol{\lambda}_{i,t}^{j,\neg k\top} \odot (2\mathbf{s}_{i,t}^j - 1)) \nabla_{x_t} \mathbf{g}_i^{\neg k}(\eta(x_t^j), \theta_i^p) \right] \\ & + \boldsymbol{\nu}_t^{j,k\top} \nabla_{x_t} \mathbf{h}^k(\eta(x_t^j)) = \mathbf{0}, \quad t = 1, \dots, T_j \end{aligned} \tag{6.10g}$$

6.3.3 Extraction of guaranteed learned AP

As with the constraint learning problem, the LTL learning problem is also ill-posed: there can be many θ^p which explain the demonstrations. Despite this, we can measure our confidence in the learned APs by checking if a constraint state κ is guaranteed to satisfy/not satisfy p_i for a given AP parameterization. This check is particularly useful when planning trajectories which satisfy the learned LTL formula, as we discuss shortly. Denote $\mathcal{F}_i$ as the feasible set of Prob. VI.4, projected onto Θ_i^p (feasible set of θ_i^p). Then, we say κ is learned to be guaranteed contained in $\mathcal{S}_i(\theta_i^p)$ if for all $\theta_i^p \in \mathcal{F}_i$, $G_i(\kappa) \leq 0$ (i.e., $\kappa \models p_i$, for all feasible θ_i^p). Similarly, we say κ is learned to be guaranteed excluded from $\mathcal{S}_i(\theta_i^p)$ if for all $\theta_i^p \in \mathcal{F}_i$, $G_i(\kappa) \geq 0$. Denote by:

$$\mathcal{G}_s^i \doteq \bigcap_{\theta \in \mathcal{F}_i} \{\kappa \mid G_i(\kappa, \theta) \leq 0\} \tag{6.11}$$

$$\mathcal{G}_{\neg s}^i \doteq \bigcap_{\theta \in \mathcal{F}_i} \{\kappa \mid G_i(\kappa, \theta) \geq 0\} \tag{6.12}$$

as the sets of κ which are guaranteed to satisfy/not satisfy p_i. Having the ability to check if a constraint state lies within $\mathcal{G}_s^i$ or $\mathcal{G}_{\neg s}^i$ is useful when planning with the learned LTL formula, as we can design our plans to be robust to any uncertainty in the learned APs. For instance, if some constraint state κ on a candidate plan must satisfy/not satisfy AP i for the plan to satisfy the learned LTL formula, we can instead force κ to be contained in $\mathcal{G}_s^i$ or $\mathcal{G}_{\neg s}^i$, respectively. Then, plans generated in this

fashion are guaranteed to satisfy the LTL formulas corresponding to any consistent θ^p.

Concretely, to query if κ is guaranteed to satisfy/not satisfy p_i, we can check the feasibility of the following problem:

Problem VI.5 (Query containment of κ in/outside of $\mathcal{S}_i(\theta_i^p)$)**.**

$$\begin{aligned} \text{find} \quad & \theta^p, \boldsymbol{\lambda}_t^{j,k}, \boldsymbol{\lambda}_{i,t}^{j,\neg k}, \boldsymbol{\nu}_t^{j,k}, \mathbf{s}_{i,t}^j, \mathbf{Q}_{i,t}^j, \mathbf{Z}^j, \ \forall i, j, t \\ \text{subject to} \quad & \{\mathrm{KKT}_{\mathrm{LTL}}(\xi_j^{\mathrm{dem}})\}_{j=1}^{N_s} \\ & G_i(\kappa, \theta_i^p) \geq 0 \ \mathbf{OR} \ G_i(\kappa, \theta_i^p) \leq 0 \end{aligned}$$

If forcing κ to (not) satisfy p_i renders Prob. VI.5 infeasible, we can deduce that to be consistent with the KKT conditions, κ must (not) satisfy p_i. Similarly, continuous volumes of κ which must (not) satisfy p_i can be extracted by solving:

Problem VI.6 (AP volume extraction)**.**

$$\begin{aligned} \underset{\substack{\varepsilon, \kappa_{\mathrm{near}}, \theta^p, \boldsymbol{\lambda}_t^{j,k}, \boldsymbol{\lambda}_{i,t}^{j,\neg k}, \\ \boldsymbol{\nu}_t^{j,k}, \mathbf{s}_{i,t}^j, \mathbf{Q}_{i,t}^j, \mathbf{Z}^j}}{\text{minimize}} \quad & \varepsilon \\ \text{subject to} \quad & \{\mathrm{KKT}_{\mathrm{LTL}}(\xi_j^{\mathrm{dem}})\}_{j=1}^{N_s} \\ & \|\kappa_{\mathrm{near}} - \kappa_{\mathrm{query}}\|_\infty \leq \varepsilon \\ & G_i(\kappa_{\mathrm{near}}, \theta_i^p) > 0 \ \mathbf{OR} \ G_i(\kappa_{\mathrm{near}}, \theta_i^p) \leq 0 \end{aligned}$$

Prob. VI.6 searches for the largest box centered around κ_{query} contained in $\mathcal{G}_s^i/\mathcal{G}_{\neg s}^i$. An explicit approximation of $\mathcal{G}_s^i/\mathcal{G}_{\neg s}^i$ can then be obtained by solving Prob. VI.6 for many different κ_{query}.

Finally, we note that another avenue to handle the ambiguity in the learned θ^p is to directly recover the set of all θ^p which are consistent with the demonstration, and planning to satisfy the LTL formulas associated with as many consistent θ^p as possible. This method is described in detail in Chapter VII for time-invariant constraints, and a detailed investigation in applying this approach to temporal logic constraints is the subject of future work.

6.4 Learning Temporal Logic Structure (θ^p, θ^s)

We will discuss how to frame the search over LTL structures θ^s (Sec. 6.4.1), the learnability of θ^s based on demonstration optimality (Sec. 6.4.2), and how we combine notions of discrete and continuous optimality to learn θ^s and θ^p (Sec. 6.4.3).

6.4.1 Representing LTL structure

We adapt *Neider and Gavran* (2018) to search for a directed acyclic graph (DAG), $\mathcal{D}$, that encodes the structure of a parametric LTL formula and is equivalent to its parse tree, with identical subtrees merged. Hence, each node still has at most two

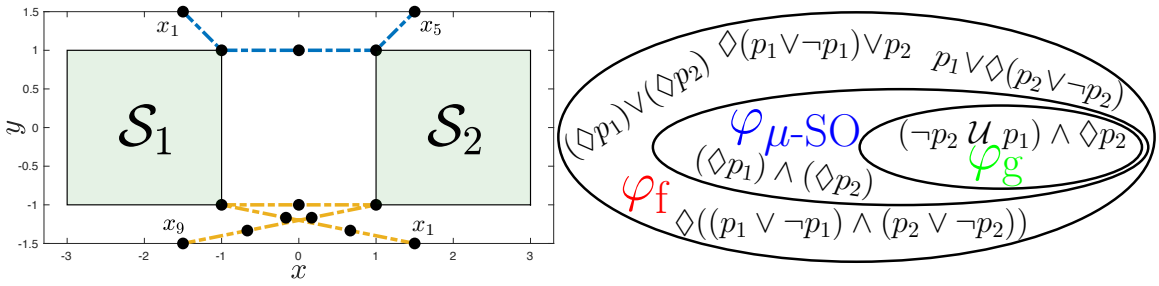


Figure 6.3: **Left**: Two demonstrations which satisfy the LTL formula $\varphi = (\neg p_2\ \mathcal{U}_{[0,T_j-1]}\ p_1) \wedge \Diamond_{[0,T_j-1]} p_2$ (first satisfy p_1, then satisfy p_2). The demonstrations satisfy kinematic constraints and are minimizing path length while satisfying input constraints and start/goal constraints. The blue and yellow demonstrations begin at the corresponding x_1 states and end at x_5 and x_9, respectively. **Right**: Some example formulas that are consistent with φ, for various levels of discrete optimality (φ_f: discrete feasibility, φ_s: spec-optimality, φ_g: discrete global optimality).

children, but can have multiple parents. This framework enables both a complete search over length-bounded LTL formulas and encoding of specific formula templates through constraints on $\mathcal{D}$ *Neider and Gavran* (2018).

Each node in $\mathcal{D}$ is labeled with an AP or operator from (6.1) and has at most two children; binary operators like $\wedge$ and $\vee$ have two, unary operators like $\Diamond_{[t_1,t_2]}$ have one, and APs have none (see Fig. 6.2). Formally, a DAG with N_{DAG} nodes, $\mathcal{D} = (\mathbf{X}, \mathbf{L}, \mathbf{R})$, can be represented as: $\mathbf{X} \in \{0,1\}^{N_{\text{DAG}} \times N_{\text{g}}}$, where $\mathbf{X}_{u,v} = 1$ if node u is labeled with element v of the grammar and 0 else, and $\mathbf{L}, \mathbf{R} \in \{0,1\}^{N_{\text{DAG}} \times N_{\text{DAG}}}$, where $\mathbf{L}_{u,v} = 1$ / $\mathbf{R}_{u,v} = 1$ if node v is the left/right child of node u and 0 else. The DAG is enforced to be well-formed (i.e., there is one root node, no isolated nodes, etc.) with further constraints; see *Neider and Gavran* (2018) for more details. Since $\mathcal{D}$ defines a parametric LTL formula, we set $\theta^s = \mathcal{D}$.

As a concrete example, consider the DAG in Fig. 6.2. Let the grammar be $\varphi ::= p_1 \mid p_2 \mid \varphi_1 \vee \varphi_2 \mid \varphi_1 \wedge \varphi_2 \mid \Box\varphi \mid \Diamond\varphi$, with DAG nodes labeled by $\{p_1, p_2, \vee, \wedge, \Box, \Diamond\}$. We refer to element 1 of the grammar as p_1, element 2 as p_2, element 3 as $\vee$, and so on. The DAG in Fig. 6.2, encoding $(\Diamond p_1) \wedge (\Diamond p_2)$, can be represented with:

$$\mathbf{X} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}, \ \mathbf{L} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \end{bmatrix}, \ \mathbf{R} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

where p_1, p_2, $\wedge$, the left $\Diamond$, and the right $\Diamond$, are labeled as nodes 1, 2, 3, 4, and 5 respectively. As convention, the unary operators are defined to have only left children.

To ensure that demonstration j satisfies the LTL formula encoded by $\mathcal{D}$, we introduce a satisfaction matrix $\mathbf{S}_j^{\text{dem}} \in \{0,1\}^{N_{\text{DAG}} \times T_j}$, where $\mathbf{S}_{j,(u,t)}^{\text{dem}}$ encodes the truth value of the subformula for the subgraph with root node u at time t (i.e., $\mathbf{S}_{j,(u,t)}^{\text{dem}} = 1$ iff the suffix of ξ_j^{dem} starting at time t satisfies the subformula). This can be encoded

with constraints:

$$|\mathbf{S}^{\text{dem}}_{j,(u,t)} - \Phi^t_{uv}| \leq M(1 - \mathbf{X}_{u,v}) \tag{6.13}$$

where Φ^t_{uv} is the truth value of the subformula for the subgraph rooted at u if labeled with v, evaluated on the suffix of ξ^{dem}_j starting at time t. The truth values are recursively generated, and the leaf nodes, each labeled with some AP i, have truth values set to $\mathbf{Z}^j_i(\theta^p_i)$. Next, we can enforce that the demonstrations satisfy the formula encoded in $\mathcal{D}$ by enforcing:

$$\mathbf{S}^{\text{dem}}_{j,(\text{root},1)} = 1,\ j = 1, \ldots, N_s \tag{6.14}$$

Continuing our example, consider again the blue trajectory in Fig. 6.3, which satisfies the aforementioned LTL formula $(\Diamond p_1) \wedge (\Diamond p_2)$. For this trajectory, $\mathbf{S}^{\text{dem}}$ is:

$$\mathbf{S}^{\text{dem}} = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 0 \end{bmatrix}$$

Note that $\mathbf{S}^{\text{dem}}_{(\text{root}=3,1)} = 1$, which reflects that the trajectory satisfies the formula. Furthermore, our method will also use synthetically-generated invalid trajectories $\{\xi^{\neg s}\}_{j=1}^{N_{\neg s}}$ (Sec. 6.4.3). To ensure $\{\xi^{\neg s}\}_{j=1}^{N_{\neg s}}$ do not satisfy the formula, we add more satisfaction matrices $\mathbf{S}^{\neg s}_j$ and enforce:

$$\mathbf{S}^{\neg s}_{j,(\text{root},1)} = 0,\ j = 1, \ldots, N_{\neg s}. \tag{6.15}$$

After discussing learnability, we will show how $\mathcal{D}$ can be integrated into the KKT-based learning framework in Sec. 6.4.3.

6.4.2 A detour on learnability

When learning only the AP parameters θ^p (Sec. 6.3), we assumed that the demonstrator chooses any *feasible* assignment of $\mathbf{Z}$ consistent with the specification, then finds a locally-optimal trajectory for those fixed $\mathbf{Z}$. Feasibility is enough if the structure θ^s of $\varphi(\theta^s, \theta^p)$ is known: to recover θ^p, we just need to find some $\mathbf{Z}$ which is feasible with respect to the known θ^s (i.e., $\Phi(\mathbf{Z}^j, \theta^p, \theta^s) = 1$) and makes ξ^{dem}_j locally-optimal; that is, the demonstrator can choose an arbitrarily suboptimal high-level plan as long as its low-level plan is locally-optimal for the chosen high-level plan. However, if θ^s is also unknown, only using boolean feasibility is not enough to recover meaningful logical structure, as this makes any formula φ for which $\Phi(\mathbf{Z}^j, \theta^p, \theta^s) = 1$ consistent with the demonstration, including trivially feasible formulas always evaluating to $\top$. Formally, we will refer to the set of formulas for which the demonstrations are feasible in the discrete variables and locally-optimal in the continuous variables as φ_f.

On the other end of the spectrum, we can assume the demonstrator is *globally-optimal* in solving Prob. VI.1, i.e., there does not exist any trajectory with lower cost than the demonstration which satisfies both the specification and the known constraints. Let the set of all formulas which make the demonstrations globally-optimal be denoted φ_g. Assuming global optimality invalidates many structures in φ_f, as any formula which accepts a trajectory with a lower cost than the demonstration cannot belong in φ_g.

To make things concrete, consider again the example in Fig. 6.3. Assume for now that θ_1^p, θ_2^p are known. Assuming boolean feasibility, we cannot distinguish between formulas in φ_f, a subset of which are written in the Venn diagram in Fig. 6.3. φ_f contains trivial formulas like $\top$ or $\varphi = (\Diamond_{[0,T_j-1]} p_1) \vee (\Diamond_{[0,T_j-1]} p_2)$, Assuming global optimality, on the other hand, invalidates many structures in φ_f, i.e., the blue trajectory should not visit both $\mathcal{S}_1$ and $\mathcal{S}_2$ if $\varphi = (\Diamond_{[0,T_j-1]} p_1) \vee (\Diamond_{[0,T_j-1]} p_2)$; we achieve a lower cost by only visiting one. Using global optimality, we can distinguish between all but the formulas with globally-optimal trajectories of equal cost (formulas in φ_g), i.e., we cannot learn the ordering constraint $(\neg p_2 \; \mathcal{U}_{[0,T_j-1]} \; p_1)$ from only the blue trajectory, as it coincides with the globally-optimal trajectory for $\varphi = (\Diamond_{[0,T_j-1]} p_1) \wedge (\Diamond_{[0,T_j-1]} p_2)$; we need the yellow trajectory to distinguish the two.

From this discussion, we see that imposing global optimality of the demonstrations in the learning problem can be quite powerful for reducing the set of consistent LTL formulas (provided that the demonstrations are actually globally-optimal). Unfortunately, enforcing global optimality of the demonstrations in the learning problem is challenging, as it requires an exhaustive verification that there are no feasible trajectories with lower cost than the demonstrations. To overcome this challenge, we define an optimality condition that is more restrictive than feasibility and less restrictive than global optimality, and which crucially is easier to impose in learning:

Definition VI.7 (Spec-optimality)**.** A demonstration ξ_j^{dem} is *μ-spec-optimal (μ-SO)*, where $\mu \in \mathbb{Z}_+$, if for every index set $\iota \doteq \{(i_1, t_1), ..., (i_\mu, t_\mu)\}$ in $\mathcal{I} \doteq \{\iota \mid i_m \in \{1, ..., N_{\text{AP}}\}, t_m \in \{1, ..., T_j\}, m = 1, ..., \mu\}$, at least one of the following holds:

- ξ_j^{dem} is locally-optimal after removing the constraints associated with p_{i_m} on $\kappa_{t_m}^j$, for all $(i_m, t_m) \in \iota$.
- For each index $(i_m, t_m) \in \iota$, the formula is not satisfied for a perturbed $\mathbf{Z}$, denoted $\hat{\mathbf{Z}}$, where $\hat{Z}_{i_m,t_m}(\theta_{i_m}^p) = \neg Z_{i_m,t_m}(\theta_{i_m}^p)$, for all $m = 1, \ldots, \mu$, and $\hat{Z}_{i',t'}(\theta_{i'}^p) = Z_{i',t'}(\theta_{i'}^p)$ for all $(i', t') \notin \iota$.
- ξ_j^{dem} is infeasible with respect to $\hat{\mathbf{Z}}$.

Spec-optimality enforces a level of logical optimality, evaluated *locally* around a demonstration: if a state κ_t^j on demonstration ξ_j^{dem} lies inside/outside of AP i (i.e., $G_i(\kappa_t^j, \theta_i^p) \leq 0 \; / \; \geq 0$), and the cost $c(\xi_j^{\text{dem}})$ can be lowered if that AP constraint is relaxed, then the constraint must hold to satisfy the specification. Intuitively, this means that the demonstrator does not visit/avoid APs which will needlessly increase the cost and are not needed to complete the task. Note that the conditions

in Def. VI.7 are essentially checking how the local optimality of a demonstration changes as a result of local perturbations to the assignments of the discrete variables $\mathbf{Z}$. The three conditions in Def. VI.7 capture the three possibilities upon perturbing $\mathbf{Z}$: the demonstration could become infeasible if $\mathbf{Z}$ is perturbed (this is what the third condition checks), the demonstration could remain feasible but local optimality may not change (this is what the first condition checks), or the demonstration could remain feasible and no longer be locally-optimal (this is what the second condition checks). By enforcing that a demonstration is spec-optimal with respect to the formula being satisfied, we enforce that this last possibility (feasible but not locally-optimal) never occurs. We would want to enforce this, for instance, if the demonstration is assumed to be globally-optimal for the true LTL formula, because there should be no alternative assignment of $\mathbf{Z}$ which admits a feasible direction in which the demonstration cost can be improved.

Returning to the discussion on the example in Fig. 6.3, we will show how spec-optimality can be used to distinguish between $\varphi = (\neg p_2 \,\mathcal{U}_{[0,T_j-1]}\, p_1) \wedge \Diamond_{[0,T_j-1]} p_2$ and $\hat{\varphi} = \Diamond_{[0,T_j-1]} p_1 \vee \Diamond_{[0,T_j-1]} p_2$ using only the blue demonstration. Specifically, we show the demonstration is 1-SO with respect to φ but not for $\hat{\varphi}$. For both formulas φ and $\hat{\varphi}$, we can see that $\mathcal{I} = \{(1,1), \ldots, (1,5), (2,1), \ldots, (2,5)\}$. Let's consider φ first. In this case, for values of $\iota \in \{(1,1), (1,3), (1,4), (1,5), (2,1), (2,2), (2,3), (2,5)\}$, the third condition in Def. VI.7 will hold, since for these time-AP pairs, the demonstration is not on the boundary of the paired AP. For $\iota \in \{(1,2), (2,4)\}$, the second condition in Def. VI.7 will hold, since perturbing $\mathbf{Z}$ at either of these time-AP pairs (from $Z_{1,2}(\theta_1^p) = 1$ to 0 or from $Z_{2,4}(\theta_2^p) = 1$ to 0) will cause φ to be not satisfied. Thus, the demonstration is spec-optimal with respect to φ. On the other hand, for $\hat{\varphi}$, again for values of $\iota \in \{(1,1), (1,3), (1,4), (1,5), (2,1), (2,2), (2,3), (2,5)\}$, the third condition in Def. VI.7 will hold. However, none of the three conditions will hold for $\iota \in \{(1,2), (2,4)\}$, since the demonstration will not be locally-optimal upon relaxing the constraints for either p_1 or p_2, and since $\hat{\varphi}$ only enforces that either one of $\mathcal{S}_1$ or $\mathcal{S}_2$ are visited, $\hat{\varphi}$ is still satisfied if either $Z_{1,2}(\theta_1^p)$ or $Z_{2,4}(\theta_2^p)$ is flipped to 0. Hence, the demonstration is not spec-optimal with respect to $\hat{\varphi}$.

In contrast, we can show that it is not possible to use spec-optimality to distinguish between the formulas $\varphi = (\neg p_2 \,\mathcal{U}_{[0,T_j-1]}\, p_1) \wedge \Diamond_{[0,T_j-1]} p_2$ and $\hat{\varphi} = \Diamond_{[0,T_j-1]} p_1 \wedge \Diamond_{[0,T_j-1]} p_2$ using the yellow demonstration in Fig. 6.3. This follows from noting that perturbing any combination of $Z_{1,4}(\theta_1^p)$, $Z_{2,6}(\theta_2^p)$ from their values of 1 to 0 will cause both φ and $\hat{\varphi}$ to be not satisfied. Hence, the yellow demonstration is spec-optimal with respect to both φ and $\hat{\varphi}$; however, it is not globally-optimal for $\hat{\varphi}$, as the demonstrator can achieve a lower cost by first satisfying p_2 and then satisfying p_1.

We will conclude this subsection with some theoretical results which motivate how demonstration spec-optimality can be used to help the learning of LTL formulas. We first show that all globally-optimal demonstrations must also be μ-spec-optimal for the true specification, for any positive integer μ.

Lemma VI.8. *All globally-optimal trajectories are μ-SO.*

Proof. We show that it is not possible for a demonstration ξ_j^{dem} to be globally-optimal while failing to satisfy (a), (b), and (c). If the constraints corresponding to p_{i_m} at

$\kappa^j_{t_m}$ are relaxed, for some $\{(i_m, t_m)\}_{m=1}^{\mu}$, then ξ_j^{dem} can either remain locally-optimal (which means (a) is satisfied, and happens if all the constraints are inactive or redundant) or become not locally-optimal. If ξ_j^{dem} becomes not locally-optimal for the relaxed problem (i.e., (a) is not satisfied), then at least one of the original constraints is active, implying $\bigvee_{m=1}^{\mu} \big(G_{i_m}(\kappa^j_{t_m}) = 0\big)$. In this case, one of the following holds: either (1) each $\kappa^j_{t_m}$ lies on its constraint boundary: $\bigwedge_{m=1}^{\mu} \big(G_{i_m}(\kappa^j_{t_m}) = 0\big)$, or (2) at least one κ_{t_m} does not lie on its constraint boundary. If (2) holds, then ξ_j^{dem} must be infeasible for $\hat{\mathbf{Z}}$, so (c) must be satisfied. If (1) holds, then ξ_j^{dem} is both feasible for $\hat{\mathbf{Z}}$ and not locally-optimal with respect to the relaxed constraints. Then, there exists some trajectory $\hat{\xi}_{xu}$ such that $c(\hat{\xi}_{xu}) < c(\xi_j^{\text{dem}})$, and for at least one m in $1, \ldots, \mu$, $G_{i_m}(\hat{\kappa}^j_{t_m}) > 0$, where $\hat{\kappa}^j_{t_m}$ is the constraint state at time t_m on $\hat{\xi}_{xu}$. $\hat{\xi}_{xu}$ cannot be feasible with respect to the true specification, since it makes ξ_j^{dem} not globally-optimal, so in this case (b) must hold. □ □

Given this result, we can use spec-optimality to vastly reduce the search space when searching for formulas which make the demonstrations globally-optimal (Sec. 6.4.3). To formalize this search space reduction, we prove that the set of consistent formulas shrinks as μ increases, approaching φ_f with lower values of μ and approaching φ_g with higher values of μ.

Theorem VI.9 (Distinguishability)**.** *For the consistent formula sets defined in Sec. 6.4.2, we have $\varphi_g \subseteq \varphi_{\tilde{\mu}\text{-SO}} \subseteq \varphi_{\hat{\mu}\text{-SO}} \subseteq \varphi_f$, for $\tilde{\mu} > \hat{\mu}$.*

Proof. $\varphi_g \subseteq \varphi_{\tilde{\mu}\text{-SO}}$, since per Lemma VI.8, all globally-optimal trajectories are $\tilde{\mu}$-SO. Thus, restricting Prob. VI.11 to enforce global optimality requires more constraints than restricting Prob. VI.11 to enforce $\tilde{\mu}$-SO. With more constraints, the feasible set of consistent formulas cannot be larger for global optimality. Similarly, as enforcing $\tilde{\mu}$-SO requires more constraints than enforcing $\hat{\mu}$-SO, the feasible set of consistent formulas cannot be larger for $\tilde{\mu}$-SO than for $\hat{\mu}$-SO. $\varphi_{\mu\text{-SO}} \subseteq \varphi_f$, since enforcing μ-SO also enforces feasibility. Thus, restricting Prob. VI.11 to enforce μ-SO requires more constraints than the standard Prob. VI.11. With more constraints, the feasible set of consistent formulas cannot be larger for μ-SO. □ □

6.4.3 Counterexample-guided framework

In this section, we will assume that the demonstrator returns a solution to Prob. VI.1 which is *boundedly-suboptimal with respect to the globally optimal solution*, in that $c(\xi_j^{\text{dem}}) \leq (1+\delta)c(\xi_j^*)$, for a known suboptimality slack parameter δ, where $c(\xi_j^*)$ is the cost of the optimal solution. This is reasonable as the demonstration should be feasible (completes the task), but may be suboptimal in terms of cost (e.g., path length, etc.), and δ can be estimated from repeated demonstrations. We sketch one way δ can be estimated in Sec. 6.6.4.

Under the bounded-suboptimality assumption, any trajectory ξ_{xu} satisfying the known constraints $\bar{\eta}(\xi_{xu}) \in \bar{\mathcal{S}}$ at a cost lower than the suboptimality bound, i.e., $c(\xi_{xu}) \leq c(\xi_j^{\text{dem}})/(1+\delta)$, must violate $\varphi(\theta^s, \theta^p)$ (Chapter III). We can use this to

Algorithm VI.1: Falsification

Input: $\{\xi_j^{\text{dem}}\}_{j=1}^{N_s}$, $\bar{\mathcal{S}}$, **Output**: $\hat{\theta}^s, \hat{\theta}^p$
$N_{\text{DAG}} \leftarrow 0$, $\{\xi^{\neg s}\} \leftarrow \{\}$
while $\neg$ *consistent* **do**
  $N_{\text{DAG}} \leftarrow N_{\text{DAG}} + 1$
  while *Problem VI.11 is feasible* **do**
    $\hat{\theta}^s$, $\hat{\theta}^p \leftarrow$ Problem VI.11$(\{\xi_j^{\text{dem}}\}_{j=1}^{N_s}, \{\xi^{\neg s}\}, N_{\text{DAG}})$
    for $j = 1$ *to* N_s **do**
      $\xi_{xu}^j \leftarrow$ Problem VI.10(ξ_j^{dem})
      if *Problem VI.10 is feasible* **then** $\{\xi^{\neg s}\} \leftarrow \{\xi^{\neg s}\} \cup \xi_{xu}^j$;
    if *Prob. VI.10 infeasible, for all* $j = 1, \ldots, N_s$ **then**
      consistent $\leftarrow \top$; break

reject candidate structures $\hat{\theta}^s$ and parameters $\hat{\theta}^p$. If we can find a counterexample trajectory that satisfies the candidate LTL formula $\varphi(\hat{\theta}^s, \hat{\theta}^p)$ at a lower cost by solving Prob. VI.10,

Problem VI.10 (Counterexample search)**.**

$$\begin{array}{ll} \text{find} & \xi_{xu} \\ \text{subject to} & \xi_{xu} \models \varphi(\hat{\theta}^s, \hat{\theta}^p) \\ & \bar{\eta}(\xi_{xu}) \in \bar{\mathcal{S}}(\xi_j^{\text{dem}}) \subseteq \mathcal{C} \\ & c(\xi_{xu}) < c(\xi_j^{\text{dem}})/(1+\delta) \end{array}$$

then $\varphi(\hat{\theta}^s, \hat{\theta}^p)$ cannot be consistent with the demonstration. Thus, we can search for a consistent $\hat{\theta}^s$ and $\hat{\theta}^p$ by iteratively proposing candidate $\hat{\theta}^s$ / $\hat{\theta}^p$ by solving Prob. VI.11 (a modified version of Prob. VI.4, which we will discuss shortly) and searching for counterexamples that can prove the parameters are invalid/valid; this is summarized in Alg. VI.1. Heuristics on the falsification loop are discussed in Sec. 6.6.3.

We note that the structure of the falsification loop in Alg. VI.1 is crucial for enforcing that the returned LTL formula makes the demonstrations globally-optimal (or boundedly-suboptimal), since as discussed in Sec. 6.4.2, it is challenging to encode global optimality directly. As a result, we will rely on encoding conditions that are weaker than global optimality but which can be efficiently enforced, proposing LTL formulas which make the demonstration feasible or spec-optimal (see Prob. VI.11). Thus, the loop is needed to reject formulas which make the demonstrations feasible or spec-optimal but not globally-optimal, in order to ensure that the formula that is eventually returned makes the demonstrations globally-optimal. We now discuss in detail the core components of Alg. VI.1: counterexample generation, addressed in Prob. VI.10, and a combined search for θ^p and θ^s, addressed in Prob. VI.11).

Counterexample generation: We propose different methods to solve Prob. VI.10 based on the dynamics. For piecewise affine systems, Prob. VI.10 can be solved directly as a MILP *Wolff et al.* (2014). However, the LTL planning problem for general nonlinear systems is challenging *Fu et al.* (2017); *Li and Fu* (2017). Probabilistically-

complete sampling-based methods *Fu et al.* (2017); *Li and Fu* (2017) or falsification tools *Annpureddy et al.* (2011) can be applied, but can be slow on high-dimensional systems. For simplicity and speed, we solve Prob. VI.10 by finding a trajectory $\hat{\xi}_{xu} \models \varphi(\hat{\theta}^s, \hat{\theta}^p)$ and boolean assignment $\mathbf{Z}$ for a kinematic approximation of the dynamics via solving a MILP, then warm-start the nonlinear optimizer with $\hat{\xi}_{xu}$ and constrain it to be consistent with $\mathbf{Z}$, returning some ξ_{xu}. We use IPOPT *Wächter and Biegler* (2006) and TrajOpt *Schulman et al.* (2014) to solve these nonlinear optimization problems for the simulation and hardware experiments, respectively. If $c(\xi_{xu}) < c(\xi_j^{\text{dem}})/(1+\delta)$, then we return, otherwise, we generate a new $\hat{\xi}_{xu}$. Whether this method returns a valid counterexample depends on if the nonlinear optimizer converges to a feasible solution; hence, this approach is not complete. However, we show that it works well in practice (see Sec. 6.8-6.9); moreover, the optimal sampling-based planning approaches (e.g., *Li and Fu* (2017)) can always be used as a complete alternative, at the expense of higher computation time.

Unifying parameter and structure search: When both θ^p and θ^s are unknown, they must be jointly learned due to their interdependence: learning the structure involves finding an unknown boolean function of θ^p, parameterized by θ^s, while learning the AP parameters θ^p requires knowing which APs were selected or negated, determined by θ^s. This can be done by combining the KKT (6.10) and DAG constraints (6.13)-(6.15) into a single MILP, which can then be integrated into Alg. VI.1:

Problem VI.11 (Combined search for θ^p, θ^s)**.**

$$
\begin{aligned}
\text{find} \quad & \mathcal{D}, \mathbf{S}_j^{\text{dem}}, \mathbf{S}_j^{\neg s}, \theta^p, \boldsymbol{\lambda}_t^{j,k}, \boldsymbol{\lambda}_{i,t}^{j,\neg k}, \boldsymbol{\nu}_t^{j,k}, \mathbf{s}_{i,t}^j, \mathbf{Q}_{i,t}^j, \mathbf{Z}^j, \\
& \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \forall i, j, t \\
\text{s.t.} \quad & \{\text{KKT}_{\text{LTL}}(\xi_j^{\text{dem}})\}_{j=1}^{N_s} \\
& \text{topology constraints (except single root) for } \mathcal{D} \\
& \text{Equation (6.13)}, \ j = 1, \ldots, N_s \\
& \text{Equation (6.14)}, \ j = 1, \ldots, N_s \\
& \text{Equation (6.15)}, \ j = 1, \ldots, N_{\neg s}
\end{aligned}
$$

In Prob. VI.11, since 1) the $\mathbf{Z}_i^j(\theta_i^p)$ at the leaf nodes of $\mathcal{D}$ are constrained via (6.8) to be consistent with θ^p and ξ_j^{dem} and 2) the formula defined by $\mathcal{D}$ is constrained to be satisfied for the $\mathbf{Z}$ via (6.13), the low-level demonstration ξ_j^{dem} must be feasible for the overall LTL formula defined by the DAG, i.e., $\varphi(\theta^s, \theta^p)$, where $\theta^s = \mathcal{D}$. $\text{KKT}_{\text{LTL}}(\xi_j^{\text{dem}})$ then chooses AP parameters θ^p to make ξ_j^{dem} locally-optimal for the continuous optimization induced by a fixed realization of boolean variables. Overall, Prob. VI.11 finds a pair of θ^p and θ^s which makes ξ_j^{dem} locally-optimal for a fixed $\mathbf{Z}^j$ which is *feasible* for $\varphi(\theta^s, \theta^p)$, i.e., $\Phi(\mathbf{Z}^j, \theta^p, \theta^s) = 1$, for all j. To also impose the spec-optimality conditions (Def. VI.7), we can add these constraints to Prob. VI.11:

$$\mathbf{S}^{\mathrm{dem},\hat{Z}_n^j}_{j,(\mathrm{root},1)} \leq b^1_{nj} \tag{6.16a}$$

$$\|\boldsymbol{\lambda}^{j,\neg k\top}_{i_m,t_m} \nabla_{x_t} \mathbf{g}^{\neg k}_{i_m}(\eta(x^j_t), \theta^p_{i_m})\| \leq M(1-b^2_{nj}), \quad m = 1, ..., \mu \tag{6.16b}$$

$$\mathbf{g}^{\neg k}_{i_m}(\eta(x^j_t), \theta^p_{i_m}) \geq -M(1-\mathbf{e}^j_{nm}),\ m = 1, ..., \mu \tag{6.16c}$$

$$\mathbf{1}^\top_{N^{i_m}_{\mathrm{ineq}}} \mathbf{e}^j_{nm} \geq \hat{Z}^j_{i_m t_m}(\theta^p_{i_m}) - b^3_{nj},\ m = 1, ..., \mu \tag{6.16d}$$

$$\mathbf{g}^{\neg k}_{i_m}(\eta(x^j_t), \theta^p_{i_m}) \leq M(\hat{Z}^j_{i_m,t_m} + b^3_{nj}) \tag{6.16e}$$

$$b^1_{nj} + b^2_{nj} + b^3_{nj} \leq 1, \quad \mathbf{b}_{nj} \in \{0,1\}^3, \quad \mathbf{e}^j_{nm} \in \{0,1\}^{N^{i_m}_{\mathrm{ineq}}} \tag{6.16f}$$

for $n = 1, \ldots, |\mathcal{I}|$, where $\mathbf{S}^{\mathrm{dem},\hat{Z}^j_n}_j$ is the satisfaction matrix for ξ^{dem}_j where the leaf nodes are perturbed to take the values of $\hat{Z}^j_n$, where n indexes an $\iota \in \mathcal{I}$. (6.16a) models the case when the formula is not satisfied, (6.16b) models when ξ^{dem}_j remains locally-optimal upon relaxing the constraint (zero stationarity contribution), and (6.16c)-(6.16e) model the infeasible case. Generally, without spec-optimality, the falsification loop in Alg. VI.1 will need to eliminate more formulas on the way to finding a formula which makes the demonstrations globally-optimal. We conclude this section with some remarks on spec-optimality and the falsification loop:

Remark VI.12. *If $\mu = 1$, the infeasibility constraints* (6.16c)-(6.16e) *can be ignored (since together with* (6.16a)*, they are redundant), and we can modify* (6.16f) *to $b^1_{nj} + b^2_{nj} \leq 1$, $\mathbf{b}_{nj} \in \{0,1\}^2$.*

Remark VI.13. *It is only useful to enforce spec-optimality on index pairs $(i_1, t_1), \ldots, (i_\mu, t_\mu)$ where $G_{i_m}(\kappa^j_{t_m}, \theta^p_{i_m}) = 0$ for all $m = 1, ..., \mu$; otherwise the infeasibility case automatically holds. If θ^p is unknown, we won't know a priori when this holds, but if θ^p are (approximately) known, we can pre-process so that spec-optimality is only enforced for salient $\iota \in \mathcal{I}$.*

Remark VI.14. *We can interpret μ as a tuning knob for shifting the computation between the falsification loop and Prob. VI.11; imposing a larger μ can potentially rule out more formulas at the cost of adding additional constraints and decision variables to Problem VI.11.*

Remark VI.15. *Prob. VI.11 with spec-optimality constraints* (6.16) *can be used to directly search for a $\varphi(\hat{\theta}^s, \hat{\theta}^p)$ which can be satisfied by visiting a set of APs in any order (e.g., surveillance-type tasks) without using the loop in Alg. VI.1, since* (6.16) *directly enforces that any AP (1-SO) or a set of APs (μ-SO) which were visited and which prevent the trajectory cost from being lowered must be visited for any candidate $\varphi(\hat{\theta}^s, \hat{\theta}^p)$.*

6.5 Learning Cost Function Parameters (θ^p, θ^s, θ^c)

If θ^c is unknown, it can be learned by modifying KKT_{LTL} to also consider θ^c in the stationarity condition: all terms containing $\nabla_{\xi_{xu}} c(\xi_j^{\text{dem}})$ should be modified to $\nabla_{\xi_{xu}} c(\xi_j^{\text{dem}}, \theta^c)$. When $c(\cdot,\cdot)$ is affine in θ^c for fixed ξ_j^{dem}, the stationarity condition is representable with a MILP constraint. However, the falsification loop in Alg. VI.1 requires a fixed cost function in order to judge if a trajectory is a counterexample. Thus, one valid approach is to first solve Prob. VI.11, searching also for θ^c, then fixing θ^c, and running Alg. VI.1 for the fixed θ^c. Specifically, the approach is the same as Alg. VI.1, apart from an additional outer while loop, where candidate θ^c are selected. We formally write this procedure in Alg. VI.2, where we refer to the Prob. VI.11 variant that searches over θ^c as Prob. VI.11′, and to the Prob. VI.10 variant that takes in θ^c as input as Prob. VI.10′. Upon the failure of a θ^c to yield a consistent θ^p and θ^s, the θ^c is added into a set of cost parameters for Problem VI.11 to avoid, Θ^c_{av}. The avoidance condition can be implemented with integer constraints, i.e., $|\theta^c_i - \hat{\theta}^c_i| \geq \varepsilon_{\text{av}} - (1 - z^i_{\text{av}})$, $\sum_i z^i_{\text{av}} \geq 1$, for $i = 1, \ldots, |\theta_c|$ and for binary variables z^i_{av}. Here, ε_{av} is a hyperparameter that defines the size of an infinity-norm ball around $\hat{\theta}_i$ which should be avoided in future iterations. One can also achieve a similar effect without this hyperparameter by adding an objective function $\max_{\theta^c} \|\theta^c - \hat{\theta}^c_i\|_\infty$ to Prob. VI.11′, which is MILP-representable.

Algorithm VI.2: Falsification, unknown cost function

Input: $\{\xi_j^{\text{dem}}\}_{j=1}^{N_s}$, $\bar{\mathcal{S}}$, **Output**: $\hat{\theta}^s, \hat{\theta}^p, \hat{\theta}^c$
$N_{\text{DAG}} \leftarrow 0$, $\{\xi^{\neg s}\} \leftarrow \{\}$, $\Theta^c_{\text{av}} \leftarrow \{\}$
while *true* **do**
  $\hat{\theta}^s, \hat{\theta}^p, \hat{\theta}^c \leftarrow$ Problem VI.11′$(\{\xi_j^{\text{dem}}\}_{j=1}^{N_s}, \{\xi^{\neg s}\}, N_{\text{DAG}}, \Theta^c_{\text{av}})$
  while ¬ *consistent* **do**
    $N_{\text{DAG}} \leftarrow N_{\text{DAG}} + 1$
    while *Problem VI.11 is feasible* **do**
      $\hat{\theta}^s, \hat{\theta}^p \leftarrow$ Problem VI.11$(\{\xi_j^{\text{dem}}\}_{j=1}^{N_s}, \{\xi^{\neg s}\}, N_{\text{DAG}}, \hat{\theta}^c)$
      for $j = 1$ *to* N_s **do**
        $\xi^j_{xu} \leftarrow$ Problem VI.10′$(\xi_j^{\text{dem}}, \hat{\theta}^c)$
        if *Problem VI.10′ is feasible* **then**
          $\{\xi^{\neg s}\} \leftarrow \{\xi^{\neg s}\} \cup \xi_{xu}$
      if *Prob. VI.10′ infeasible, for all* $j = 1, \ldots, N_s$ **then**
        consistent $\leftarrow \top$; break
  if *consistent* **then** return;
  else $\Theta^c_{\text{av}} \leftarrow \Theta^c_{\text{av}} \cup \hat{\theta}^c$; break;

Note that this procedure either eventually returns an LTL formula consistent with the fixed θ^c, or Alg. VI.1 becomes infeasible, and a new θ^c must be generated and Alg. VI.1 rerun. This is guaranteed to eventually return a set of θ^c, θ^s, and θ^p which make each ξ_j^{dem} globally-optimal with respect to $c(\xi_{xu}, \theta^c)$ under $\varphi(\theta^s, \theta^p)$. However, it may require iterating through an infinite number of candidate θ^c and hence is not

guaranteed to terminate in finite time (Cor. VI.23). Nonetheless, we note that for a certain class of formulas (Rem. VI.15), a consistent set of θ^c, θ^s, and θ^p can be recovered in one shot.

6.6 Method extensions, variants, and discussion

In this section, we discuss some extensions and variants of our approach which can improve learning (Sec. 6.6.1) and computational performance (Sec. 6.6.2, Sec. 6.6.3). Finally, we discuss the effect of suboptimality on the learning procedure and how the suboptimality slack parameter δ can be estimated (Sec. 6.6.4).

6.6.1 Encoding prior knowledge

In some situations, we may have some *a priori* knowledge on the atomic propositions, e.g., which labels correspond to which atomic proposition regions, or a rough estimate of the AP parameters θ^p. We describe how this knowledge can be integrated into our method.

Known labels: We have assumed that the demonstrations only include state/control trajectories and not the AP labels; this can lead to ambiguity as to which $\mathcal{S}$ should be assigned to which proposition p_i. For example, consider the example in Fig. 6.3 (left), where the aim is to recover $\varphi(\theta^p) = \Diamond\mathcal{S}_1(\theta^p_1) \vee \Diamond\mathcal{S}_2(\theta^p_2)$. The KKT conditions will imply that the demonstrator had to visit two boxes and their locations, but not whether the left box should be labeled $\mathcal{S}_1$ or $\mathcal{S}_2$. However, in some settings it may be reasonable that the labels for each AP are provided, e.g., for an AP which requires a robot arm to grasp an object, we might have sensor data determining if the object has been grasped. In this case, we can incorporate this by simply constraining $\mathbf{Z}^j_i(\theta^p_i)$ to be the labels; this then removes the ambiguity mentioned earlier.

Prior knowledge on θ^p: In some settings, we may have a rough idea of θ^p, e.g., as noisy bounding boxes from a vision system. We might then want to avoid deviating from these nominal parameters, denoted θ^p_{nom}, or restrict θ^p to some region around θ^p_{nom}, denoted $\Theta_{i,\text{nom}}$, subject to the KKT conditions holding. This can be done by adding $\sum_{j=1}^{N_{\text{AP}}} \|\theta^p_i - \theta^p_{i,\text{nom}}\|_1$ as an objective or $\theta^p_{i,\text{nom}} \in \Theta_{i,\text{nom}}$ as a constraint to Prob. VI.4 instead of simply solving Prob. VI.4 as a feasibility problem.

6.6.2 Faster reformulations for the falsification loop

A shortcoming of Alg. VI.1 is that it can be computationally intensive. This is primarily due to Prob. VI.11, which is a mixed-integer program that contains many binary decision variables, including the DAG structure variables $(\mathbf{X}, \mathbf{L}, \mathbf{R})$ and variables $\mathbf{Q}$ and $\mathbf{Z}$ which are needed to learn the continuous parameters θ^p. While Prob. VI.11 can still be solved for examples of moderate size (see the results in Sec. 6.8), we observe that its computation time can become unrealistic for examples with very long LTL formulas (i.e., a large search space for $(\mathbf{X}, \mathbf{L}, \mathbf{R})$). Intuitively, increasing the dimensionality of $(\mathbf{X}, \mathbf{L}, \mathbf{R})$ combinatorially increases the number of

possible assignments, which can cause the optimizer to struggle to find a feasible solution in a reasonable timeframe.

To address these computational challenges, we propose a reformulation for Prob. VI.11 which is better suited for large-scale problems. Instead of fixing the number of nodes N_{DAG} in the DAG $\mathcal{D}$ and searching over grammar element types $\mathbf{X}_{uv}$ for which to populate the nodes, we can fix $\mathbf{X}$ to contain a number of instances of each grammar element, and relax the constraint that there is only one root node, and enforce the constraints of Prob. VI.11 on the LTL formula defined by the subgraph of a particular root node; that is, we enforce the constraints on one tree in the forest of an expanded DAG where AP nodes with common labels are not merged. Additionally, instead of incrementing the *total* DAG size N_{DAG} in the outer loop of Alg. VI.1, we should increment the number of instances of *each* grammar element by one. As a concrete example, instead of searching for a DAG with 5 nodes, where each node can be labeled with any element in the grammar $\{p_1, p_2, \wedge, \Diamond, \Box\}$, one possibility under this reformulation would be to fix $\mathbf{X}$ to contain 11 nodes, with one instance each of p_1 and p_2 and three instances each of $\wedge$, $\Diamond$, and $\Box$. The optimizer would then choose a subset of these nodes to include in the candidate LTL formula by choosing a root node and $(\mathbf{L}, \mathbf{R})$ accordingly.

More concretely, this reformulated problem can be written as a modification of Prob. VI.11, where $\mathbf{X}$ is dropped as a decision variable and an additional binary vector $\mathbf{r} \in \{0,1\}^{N_{\text{DAG}}}$ is added. The purpose of $\mathbf{r}$ is to encode that at least one node in the DAG is a root node, and that conditions (6.14), (6.15), and (6.16) hold for each root node; concretely, if $r_i = 1$, node i is a root node, and if $r_i = 0$, then node i is not a root node. This adjustment can be performed by taking constraints (6.14), (6.15), and (6.16) and relaxing them depending on the value of $\mathbf{r}$, using a big-M formulation. As a concrete example, (6.14) would be modified to

$$\begin{aligned} 1 - \mathbf{S}^{\text{dem}}_{j,(r_i,1)} \leq M(1 - r_i),\ i = 1, \ldots, N_{\text{DAG}}, \\ j = 1, \ldots, N_s. \end{aligned} \tag{6.17}$$

By holding $\mathbf{X}$ constant instead of considering it as a decision variable, we dramatically reduce the computational cost of solving Prob. VI.11, by combinatorially reducing the search space size compared to searching over the entirety of $(\mathbf{X}, \mathbf{L}, \mathbf{R})$. Furthermore, this does not overly restrict the LTL formula search, since we can still represent different formulas by searching over $\mathbf{L}$ and $\mathbf{R}$, and by allowing for multiple root nodes, we can still find different formulas involving a different number of nodes (i.e., the method can return formula defined by a subtree containing only a subset of the nodes in $\mathbf{X}$). For instance, consider representing the formula $\varphi = \Diamond p_1 \wedge \Diamond p_2$ using either formulation. Using the original formulation, one can represent φ by searching for a DAG with 5 nodes, resulting in the structure in Fig. 6.2. Using the reformulation, we can represent φ even when selecting one instance each of p_1 and p_2 and three instances each of $\wedge$, $\Diamond$, and $\Box$, as long as some subgraph in the resulting DAG replicates the structure in Fig. 6.2, and the root of that subgraph (say node i) is marked as a root node ($r_i = 1$). However, these computational gains can come at the cost of easily finding the shortest LTL formula consistent with the demonstrations,

as we discuss in Cor. VI.20 (see Rem. VI.21 for more discussion). Thus, this formulation should be used for large-scale learning problems with many APs and LTL grammar elements, while it should be avoided when the primary priority is to return the simplest possible LTL formula.

6.6.3 Prioritized variants on the falsification loop

Depending on the desired application, it may be useful to impose an ordering in which candidate structures θ^s are returned in line 4 of Alg. VI.1. For example, the user may want to return the most restrictive formulas first (i.e., formulas with the smallest language), since more restrictive formulas are less likely to admit counterexamples (and hence the falsification should terminate in fewer iterations). On the other hand, the user may want to return the least restrictive formulas first, generating many invalid formulas in order to explicitly know what formulas do not satisfy the demonstrator's wishes.

However, imposing an entailment-based ordering on the returned formulas is computationally challenging, as in general this will involve pairwise LTL entailment checks over a large set of possible LTL formulas, and each check is in PSPACE *Demri and Schnoebelen* (2002). Despite this, we can heuristically approximate this by assigning weights to each node type in the DAG based on their logical "strength", such that each DAG with the same set of nodes has an overall weight $w = \sum_{u=1}^{N_{\text{DAG}}} \sum_{v=1}^{N_{\text{g}}} w_{u,v} \mathbf{X}_{u,v}$. For example, $\vee$ should be assigned a lower weight than $\wedge$, since $\vee$s can never restrict language size, while $\wedge$ can never grow it. Then, stronger/weaker formulas can be returned first by adding constraint $w \geq w_{\text{thresh}}/w \leq w_{\text{thresh}}$, where w_{thresh} is reduced/increased until a consistent formula is found.

Note that multiple consistent formula structures can be also generated by adding a constraint for Prob. VI.11 to not return the same formula structure and continuing the falsification loop after the first consistent formula is found.

6.6.4 Demonstration suboptimality

We conclude this section by describing a method for estimating the suboptimality slack parameter δ, which is crucial for maintaining the correctness of Alg. VI.1, and by discussing how demonstrator suboptimality can affect the performance of our algorithm.

We first describe how δ can be estimated. Assume that the cost function parameters θ^c are fixed. Suppose that the demonstrator repeats task j R times, generating suboptimal demonstrations $\{\xi_{j,r}^{\text{dem}}\}_{r=1}^{R}$ with corresponding costs $\{c(\xi_{j,r}^{\text{dem}})\}_{r=1}^{R}$, where $c(\xi_{j,r})^{\text{dem}} \geq c(\xi_j^*)$, for all r, where $c(\xi_j^*)$ is the cost of a globally-optimal solution for task j, which we assume is finite. Using these repeated demonstrations, we would like to estimate the suboptimality bound δ. Assuming the demonstration costs are independent and identically distributed realizations of a random variable, we can estimate $c(\xi_j^*)$ using the location parameter of a Weibull distribution that is fit to the observed costs *De Haan and Ferreira* (2007); *Knuth et al.* (2021a); *Weng et al.* (2018). This follows from the Fisher-Tippett-Gnedenko Theorem from extreme value

theory *De Haan and Ferreira* (2007), which states that if the limiting distribution of the minimum of a set of realizations of a random variable converges to a finite value, the limit distribution is Weibull. Then, the location parameter of the Weibull distribution can be used to estimate the minimum $c(\xi_j^*)$; let this estimate be denoted $\hat{c}_j^*$. One can also compute a confidence interval around $\hat{c}_j^*$ *Knuth et al.* (2021a), which can be used to determine if the demonstration needs to be further repeated (i.e., if the confidence interval is large). Finally, we can recover an estimate of δ by taking the lowest-cost trajectory (which will be selected as the demonstration used in learning[2]) with cost $c(\xi_j^{\text{dem}}) \doteq \min_{1 \leq r \leq R} c(\xi_{j,r}^{\text{dem}})$ and setting

$$\delta = \frac{c(\xi_j^{\text{dem}}) - \hat{c}_j^*}{\hat{c}_j^*}. \tag{6.18}$$

We demonstrate this procedure on a simulated example in Sec. 6.8.2.

This δ estimation procedure can also be altered to work for the case of unknown θ^c. Since Alg. VI.2 fixes a single consistent θ^c in an outer loop and then runs the falsification loop of Alg. VI.1 for that fixed θ^c, we can estimate δ for each θ^c which comes up in the outer loop directly using the procedure described previously for a fixed θ^c. Thus, δ changes based on the current candidate θ^c.

We now discuss the overall effect of suboptimality on our method. Recall that our approach relies on a continuous notion of optimality to learn θ^p and θ^c (the KKT conditions) and discrete notions of optimality in a falsification loop to learn the LTL structure θ^s. We first discuss the effect of suboptimality on learning θ^p and θ^c; in these cases, any demonstrator suboptimality is reflected by the KKT conditions failing to hold exactly on the demonstrations (i.e., with an error in the stationarity or complementary slackness terms). This can be dealt with by solving Prob. VI.3, which relaxes the KKT conditions to a penalty, so the optimization problem remains feasible despite the suboptimality. In essence, Prob. VI.3 finds the cost function/AP parameters which make the demonstrations as close to satisfying the KKT conditions as possible. Unfortunately, these parameters may not reflect the true parameters if the demonstrations are extremely suboptimal; as a result, the accuracy of the recovered parameters can be sensitive to suboptimality. Quantifying uncertainty in the learned parameters as a function of the demonstrator's suboptimality may help mitigate any performance degradation, and is an interesting direction for future work.

Learning the LTL structure θ^s is in general less sensitive to suboptimality. To understand this, let us return to the two-AP setting of Fig. 6.3. In this setting, we first sort the possible LTL structures on a number line by the optimal trajectory cost that they admit (see Fig. 6.4 for a depiction of this idea). There are finitely many possible LTL structures θ^s, and many different θ^s may be semantically identical (for example, many θ^s have corresponding formulas which are just permutations of each other), thus admitting optimal trajectories of the same cost. Thus, while there may be exponentially many possible θ^s, there tends to only be a small number of

[2]Provided that the remaining higher-cost demonstrations are feasible, they can still be used in the learning process; we can enforce that these demonstrations should still be feasible for any candidate LTL formula.

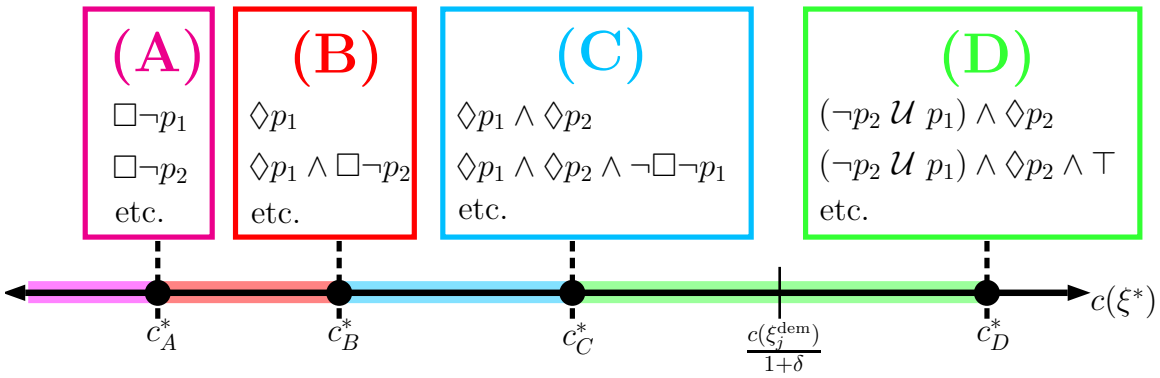


Figure 6.4: Consider the two-AP setting first shown in Fig. 6.3. We visualize here sets of LTL formulas which can be distinguished based on cost. Formulas within group $(\cdot)$ have an optimal cost $c^*_\cdot$. The formulas listed in each group (A), (B), (C), and (D) are just a small subset of a much larger set of cost-indistinguishable formulas. For instance, if a demonstration has a δ-adjusted cost $c(\xi_j^{\text{dem}})/(1+\delta)$ falling in the green range, Alg. VI.1 will return some LTL formula structure in group (D), each of which would have an optimal cost of c^*_D.

groups of cost-distinguishable formulas (i.e., each such group contains formulas with equal optimal cost). Recall that in running Alg. VI.1 using the given demonstrations to learn θ^s, the falsification loop terminates when the optimal cost of a trajectory satisfying the current candidate LTL formula and the known constraints exceeds the δ-adjusted demonstration cost $c(\xi_j^{\text{dem}})/(1+\delta)$. As an example, consider a suboptimal demonstration of $\varphi = (\neg p_2 \,\mathcal{U}\, p_1) \wedge \Diamond p_2$ which belongs to group (D) in Fig. 6.4 and has a δ-adjusted cost $c(\xi_j^{\text{dem}})/(1+\delta)$. As long as this adjusted cost lies anywhere within the green interval in Fig. 6.4, some formula from group (D) is returned, which will be a formula consistent with the bounded suboptimality of the demonstration. Note that the estimate of δ must be an overestimate of the true δ in order for the adjusted cost to lie in the green region in Fig. 6.4; this can be encouraged by setting the confidence interval described earlier in this section to be large, and selecting δ as the fit Weibull location parameter padded by the confidence interval. In Sec. 6.8.2, we show that we can obtain an overestimate of the true δ using this approach.

6.7 Theoretical Analysis

In this section, we prove some theoretical guarantees of our method: that it is complete under some assumptions, without (Thm. VI.19) or with (Cor. VI.22) spec-optimality, that it returns the shortest LTL formula consistent with the demonstrations (Cor. VI.20), and that we can compute guaranteed conservative estimates of $\mathcal{S}_i/\mathcal{A}_i$ (Thm. VI.24).

Assumption VI.16. *Prob. VI.10 is solved with a complete planner.*

Assumption VI.17. *Each demonstration is locally-optimal (i.e., satisfies the KKT conditions) for fixed boolean variables.*

Assumption VI.18. *The true parameters θ^p, θ^s, and θ^c are in the hypothesis space of Prob. VI.11: $\theta^p \in \Theta^p$, $\theta^s \in \Theta^s$, $\theta^c \in \Theta^c$.*

We will use these assumptions to show that when the cost function parameters θ^p are known, our falsification loop in Alg. VI.1 is guaranteed to return a consistent formula; that is, it makes the demonstrations globally-optimal.

Theorem VI.19 (Completeness and consistency, unknown θ^s, θ^p)**.** *Under Assumptions VI.16-VI.18, Alg. VI.1 is guaranteed to return a formula $\varphi(\theta^s, \theta^p)$ such that 1) $\xi_j^{dem} \models \varphi(\theta^s, \theta^p)$ and 2) ξ_j^{dem} is globally-optimal under $\varphi(\theta^s, \theta^p)$, for all j, 3) if such a formula exists and is representable by the provided grammar.*

Proof. To see the first point - that Alg. VI.1 returns $\varphi(\hat{\theta}^s, \hat{\theta}^p)$ such that $\xi_j^{\text{dem}} \models \varphi(\hat{\theta}^s, \hat{\theta}^p)$ for all j, note that in Prob. VI.11, the constraints (6.13)-(6.15) on the satisfaction matrices $\mathbf{S}_j^{\text{dem}}$ encode that each demonstration is feasible for the choice of θ^p and θ^s; hence, the output of Prob. VI.11 will return a feasible $\varphi(\hat{\theta}^s, \hat{\theta}^p)$. As Alg. VI.1 will eventually return some $\varphi(\hat{\theta}^s, \hat{\theta}^p)$ which is an output of Prob. VI.11, the $\varphi(\hat{\theta}^s, \hat{\theta}^p)$ that is ultimately returned is feasible.

Next, to see the second point - that the ultimately returned $\varphi(\hat{\theta}^s, \hat{\theta}^p)$ makes each ξ_j^{dem} globally-optimal. Note that at some iteration of the inner loop, if Prob. VI.10 is feasible and its solution algorithm is complete (Assumption VI.16), it will return a trajectory which is lower-cost than the demonstration and satisfies $\varphi(\hat{\theta}^s, \hat{\theta}^p)$. Note that disregarding the lower-cost constraint, Prob. VI.10 will always be feasible, since Prob. VI.11 returns θ^p, θ^s for which the demonstration is feasible, and the feasible set of Prob. VI.10 contains the demonstration. The falsification loop will continue until Prob. VI.10 cannot produce a trajectory of strictly lower cost for each demonstration; this is equivalent to ensuring that each demonstration is globally optimal for the $\varphi(\hat{\theta}^s, \hat{\theta}^p)$.

To see the last point, we note that if there exists a formula $\varphi(\hat{\theta}^s, \hat{\theta}^p)$ which satisfies the demonstrations, it is among the feasible set of possible outputs of Alg. VI.1; that is, the representation of LTL formulas, $\mathcal{D}$, is complete (cf. Lemma 1 in *Neider and Gavran* (2018)). □ □

We will further show that the formula returned by Alg. VI.1 is the shortest formula which is consistent with the demonstrations; this is due to N_{DAG} only being incremented upon infeasibility of a smaller N_{DAG} to explain the demonstrations.

Corollary VI.20 (Shortest formula)**.** *Let N^* be the size of a minimal DAG for which there exists (θ^p, θ^s) such that $\xi_j^{dem} \models \varphi(\theta^s, \theta^p)$ for all j. Under Assumptions VI.16-VI.18, Alg. VI.1 is guaranteed to return a DAG of size N^*.*

Proof. The result follows since Algorithm VI.1 increases N_{DAG} incrementally (in the outer loop) until some $\varphi(\hat{\theta}^s, \hat{\theta}^p)$ is returned which makes all of the demonstrations feasible and globally-optimal, and each inner iteration of Algorithm VI.1 is guaranteed to find a consistent $\varphi(\hat{\theta}^s, \hat{\theta}^p)$ if one exists (cf. Theorem VI.19). □ □

Remark VI.21. *A similar shortest formula guarantee can be obtained for the reformulation of Alg. VI.1 described in Sec. 6.6.2 only if it is tractable to perform an exhaustive search over the number of nodes allocated to each grammar element, in order to find the shortest-length combination. This can be computationally intensive, and is in contrast to the simple "line-search" over a single complexity variable, N_{DAG}, that the original Alg. VI.1 enjoys.*

Using Lem. VI.8, we can show that modifying Alg. VI.1 to additionally impose the spec-optimality conditions in Prob. VI.11 still enjoys the completeness properties discussed in Theorem VI.19, while also in general reducing the number of falsification iterations needed as a result of the reduced search space.

Corollary VI.22 (Alg. VI.1 with spec-optimality)**.** *By modifying Alg. VI.1 so that Prob. VI.11 uses constraints* (6.16)*, Alg. VI.1 still returns a consistent solution $\varphi(\hat{\theta}^s, \hat{\theta}^p)$ if one exists, i.e., each ξ_j^{dem} is feasible and globally optimal for each $\varphi(\hat{\theta}^s, \hat{\theta}^p)$.*

Proof. The result follows from completeness of Alg. VI.1 (cf. Theorem VI.19) and Lemma VI.8: adding (6.16a)-(6.16c) enforces that ξ_j^{dem} are spec-optimal, and via Lemma VI.8, ξ_j^{dem}, which is a globally-optimal demonstration, must also be spec-optimal. Hence, imposing constraints (6.16a)-(6.16c) is consistent with the demonstration. □ □

Next, we show how the consistency properties extend to the case of unknown cost function, if Alg. VI.2 returns a solution, which it is not guaranteed to do in finite time.

Corollary VI.23 (Consistency, unknown θ^c)**.** *Under Assumptions VI.16-VI.18, if Alg. VI.2 terminates in finite time, it returns a formula $\varphi(\theta^s, \theta^p)$ such that 1) $\xi_j^{dem} \models \varphi(\theta^s, \theta^p)$ and 2) ξ_j^{dem} is globally-optimal with respect to θ^c under the constraints of $\varphi(\theta^s, \theta^p)$, for all j, 3) if such a formula exists and is representable by the provided grammar.*

Proof. Note that Alg. VI.2 is simply Alg. VI.1 with an outer loop where potential cost parameters θ^c are chosen. From Theorem VI.19, we know that under Assumptions VI.16-VI.17, for the true cost parameter θ^c, Alg. VI.1 is guaranteed to return θ^p and θ^s which make the demonstrations globally-optimal under θ^c. From Assumption VI.18 and the fact that the true parameters θ^p, θ^s, and θ^c will make the demonstrations globally-optimal, we know there exists at least one consistent set of parameters (the true parameters). Then, Alg. VI.2 will eventually find a consistent solution (possibly the true parameters), as it iteratively runs Alg. VI.1 for all consistent θ^c. □ □

Finally, we show that for fixed LTL structure and cost function, querying and volume extraction (Problems VI.5 and VI.6) are guaranteed to return conservative estimates of the true $\mathcal{S}_i$ or $\mathcal{A}_i$.

Theorem VI.24 (Conservativeness for unknown θ^p)**.** *Suppose that θ^s and θ^c are known, and θ^p is unknown. Then, extracting $\mathcal{G}_s^i$ and $\mathcal{G}_{\neg s}^i$, as defined in* (6.11)-(6.12)*, from the feasible set of Prob. VI.4 projected onto Θ_i^p (denoted $\mathcal{F}_i$), returns $\mathcal{G}_s^i \subseteq \mathcal{S}_i$ and $\mathcal{G}_{\neg s}^i \subseteq \mathcal{A}_i$, for all $i \in \{1, \ldots, N_{AP}\}$.*

Proof. We first prove that $\mathcal{G}^i_{\neg_s} \subseteq \mathcal{A}_i$. Suppose that there exists $\kappa \in \mathcal{G}^i_{\neg_s}$ such that $\kappa \notin \mathcal{A}_i$. Then by definition, for all $\theta^p_i \in \mathcal{F}_i$, $G_i(\kappa, \theta^p_i) \geq 0$. However, we know that all locally-optimal demonstrations satisfy the KKT conditions with respect to the true parameter $\theta^{p,*}_i$; hence, $\theta^{p,*}_i \in \mathcal{F}$. Then, $x \in \mathcal{A}(\theta^{p,*}_i)$. Contradiction. Similar logic holds for proving that $\mathcal{G}^i_s \subseteq \mathcal{S}_i$. Suppose that there exists $x \in \mathcal{G}^i_s$ such that $x \notin \mathcal{S}_i$. Then by definition, for all $\theta^p_i \in \mathcal{F}_i$, $G_i(\kappa, \theta^p_i) \leq 0$. However, we know that all locally-optimal demonstrations satisfy the KKT conditions with respect to the true parameter $\theta^{p,*}_i$; hence, $\theta^{p,*}_i \in \mathcal{F}_i$. Then, $\kappa \in \mathcal{S}_i(\theta^{p,*}_i)$. Contradiction. □ □

6.8 Simulation Experiments

We show that our algorithm outperforms a competing method (Sec. 6.8.1), can be robust to suboptimality in the demonstrations (Sec. 6.8.2), can learn shared task structure from demonstrations across environments (Sec. 6.8.3), and can learn LTL formulas θ^p, θ^s and uncertain cost functions θ^c on high-dimensional problems. Specifically, we demonstrate Alg. VI.1 on a simulated manipulation example (Sec. 6.8.4) and the one-shot learning described in Rem. VI.15 on a quadrotor surveillance task (Sec. 6.8.5). Please refer to the supplementary video for visualizations of the results.

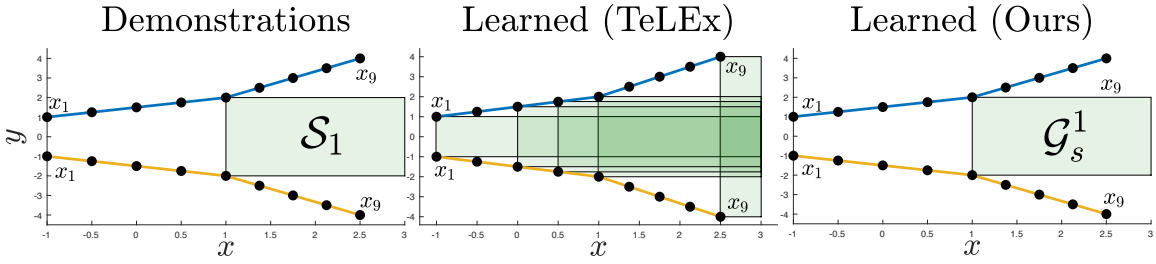


Figure 6.5: Toy example for baseline comparison *Jha et al.* (2019). The baseline is unable to disambiguate between possible APs as it does not consider the demonstrator's objective.

6.8.1 Baseline comparison

Likely the closest method to ours is *Jha et al.* (2019), which learns a pSTL formula that is tightly satisfied by the demonstrations via solving a nonconvex problem to local optimality: $\arg\max_{\theta^p} \min_j \tau(\theta^p, \xi^{\text{dem}}_j)$, where $\tau(\theta^p, \xi^{\text{dem}}_j)$ measures how tightly ξ^{dem}_j fits the learned formula. We run the authors' code *Jha* (2017) on a toy problem (see Fig. 6.5), where the demonstrator has kinematic constraints, minimizes path length, and satisfies start/goal constraints and $\varphi = \Diamond_{[0,8]} p_1$, where $x \models p_1 \Leftrightarrow [I_{2\times 2}, -I_{2\times 2}]^\top x \leq [3, 2, -1, 2]^\top = [3, \theta^p_1]^\top$. We assume the structure θ^s is known, and we aim to learn θ^p to explain why the demonstrator deviated from an optimal straight-line path to the goal. Solving Prob. VI.6 returns $\mathcal{G}^1_s = \mathcal{S}_1$ (Fig. 6.5, right). On the other hand, we run TeLEx multiple times, converging to different local optima, each corresponding to a "tight" θ^p (Fig. 6.5, center): TeLEx cannot

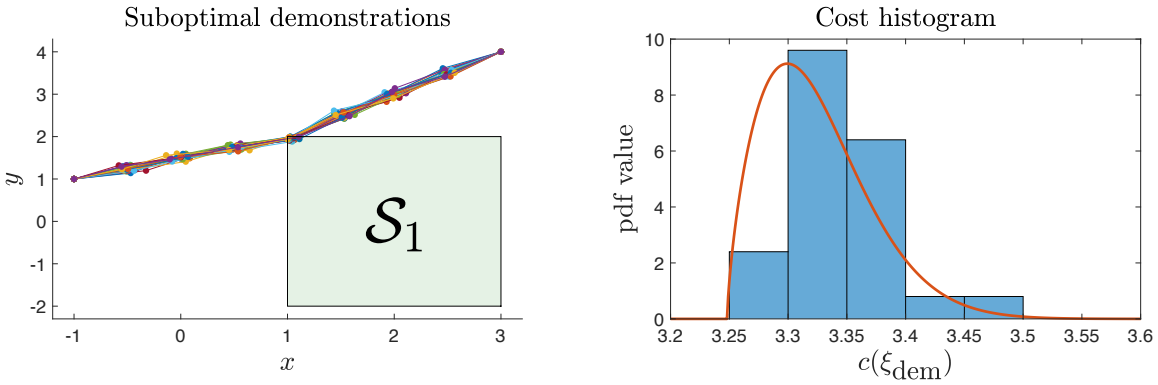


Figure 6.6: **Left**: We are given 25 suboptimal demonstrations of the same task, with each demonstration starting at $[-1, 1]$, ending at $[3, 4]$, and satisfying $\Diamond_{[0,8]} p_1$. The globally-optimal cost is 3.25, while the best cost observed within the 25 demonstrations is 3.274. **Right**: We fit a Weibull distribution (orange) to the demonstration costs (right). The fitted location parameter, adjusted by its 95% confidence interval, is $3.248 < 3.25$, which leads to a valid overestimate of δ.

distinguish between multiple different "tight" θ^p, which makes sense, as the method tries to find *any* "tight" solution. This example suggests that if the demonstrations are goal-directed, a method that leverages their optimality is likely to better explain them.

6.8.2 δ-estimation for suboptimal demonstrations

In this example, we demonstrate the suboptimality estimation method described in Sec. 6.6.4. In this example, we consider the same problem setting as in Sec. 6.8.1, but instead use suboptimal versions of the blue demonstration in Fig. 6.5. We are given 25 such demonstrations (Fig. 6.6, left), and we are interested in estimating the suboptimality slack parameter δ. To do so, we follow the method in Sec. 6.6.4, fitting a Weibull distribution (Fig. 6.6, right, orange) to the demonstration costs (Fig. 6.6, right, blue histogram). The fitted Weibull distribution has a location parameter of 3.248 after being adjusted by its 95% confidence interval, which is smaller than the optimal cost of 3.25. Using the suboptimal demonstration with the lowest cost (in this case, 3.274), we can estimate $\delta = 0.008$ using (6.18), which overestimates the true $\delta = 0.007$. Per the discussion in Sec. 6.6.4, it is important to be able to obtain an estimate of δ which is a tight overestimate of the true δ, which this example achieves. Overall, this example suggests that our δ-estimation technique can effectively estimate the suboptimality bound, which is important for learning consistent LTL formulas in spite of suboptimality in the demonstrations.

6.8.3 Learning shared task structure

In this example, we show that our method can extract logical structure shared between demonstrations that complete the same high-level task, but in different environments (Fig. 6.7). A point robot must first go to the mug (p_1), then go to the

coffee machine (p_2), and then go to goal (p_3) while avoiding obstacles (p_4, p_5). As the floor maps differ, θ^p also differ, and are assumed known. We add two relevant primitives to the grammar, sequence:

$$\varphi_1 \ \mathcal{Q} \ \varphi_2 \doteq \neg\varphi_2 \ \mathcal{U}_{[0,T_j-1]} \ \varphi_1,$$

enforcing that φ_2 cannot occur until after φ_1 has occurred for the first time, and avoid: $\mathcal{V}\varphi \doteq \Box_{[0,T_j-1]}\neg\varphi$, enforcing φ never holds over $[1, T_j]$. Then, the true formula is:

$$\varphi^* = \mathcal{V}p_4 \wedge \mathcal{V}p_5 \wedge (p_1 \ \mathcal{Q} \ p_2) \wedge (p_2 \ \mathcal{Q} \ p_3) \wedge \Diamond_{[0,T_j-1]} p_3.$$

Suppose first that we are given the blue demonstration in Environment 2. Running Alg. VI.1 with 1-SO constraints (6.16) terminates in one iteration at $N_{\text{DAG}} = 14$ with

$$\varphi_0 = \mathcal{V}p_4 \wedge \mathcal{V}p_5 \wedge \Diamond_{[0,T_j-1]} p_2 \wedge \Diamond_{[0,T_j-1]} p_3 \wedge (p_1 \ \mathcal{Q} \ p_2).$$

That is, always avoid obstacles 1 and 2, eventually reach coffee and goal, and visit mug before coffee. This formula is insufficient to complete the true task (the ordering constraint between coffee and goal is not learned). This is because the optimal trajectories satisfying φ_0 and φ^* are the same cost, i.e., both φ_0 and φ^* are consistent with the demonstration and could have been returned, and $\varphi_0, \varphi^* \in \varphi_g$ (cf. Sec. 6.7). Now, we also use the blue demonstration from Environment 1 (two examples total). Running Alg. VI.1 terminates in two iterations at $N_{\text{DAG}} = 14$ with the formulas

$$\varphi_1 = \mathcal{V}p_4 \wedge \mathcal{V}p_5 \wedge \Diamond_{[0,T_j-1]} p_1 \wedge \Diamond_{[0,T_j-1]} p_2 \wedge \Diamond_{[0,T_j-1]} p_3$$

(which enforces that the mug, coffee, and goal must be eventually visited, but in any order, while avoiding obstacles) and $\varphi_2 = \varphi^*$. Since the demonstration in Environment 1 doubles back to the coffee before going to goal, increasing its cost over first going to goal and then to coffee, the ordering constraint between the two is learnable. We also plot the generated counterexample (Fig. 6.7, yellow), which achieves a lower cost, as φ_1 involves no ordering constraints. We can use the learned formula to plan a path completing the task in a new environment (with different AP parameters θ^p) in Fig. 6.8.

Overall, this example suggests we can use demonstrations from different environments to learn common task structure and disambiguate between potential explanations.

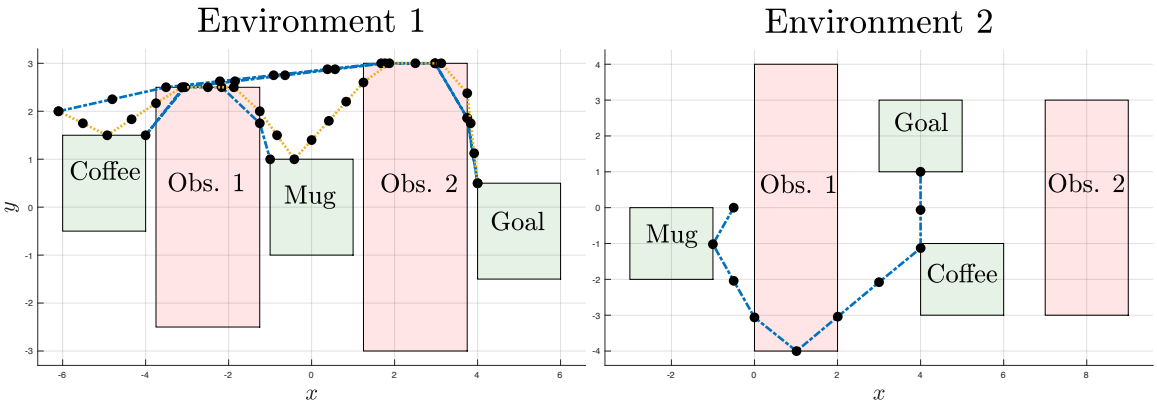


Figure 6.7: We learn a common LTL formula from demonstrations in different environments (different θ^p) with shared task (same θ^s).

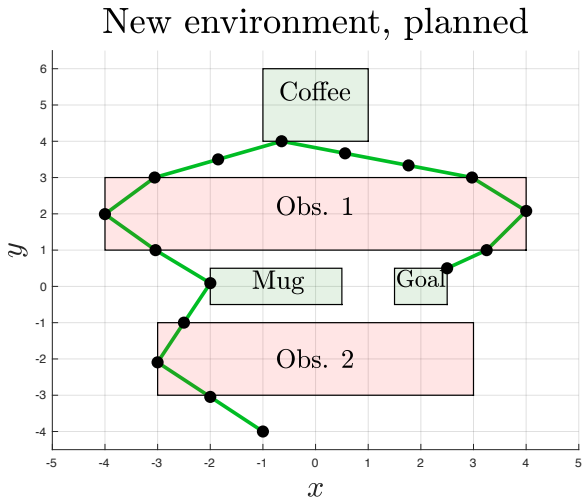


Figure 6.8: Trajectory planned with the learned LTL formula on the environment-transfer example.

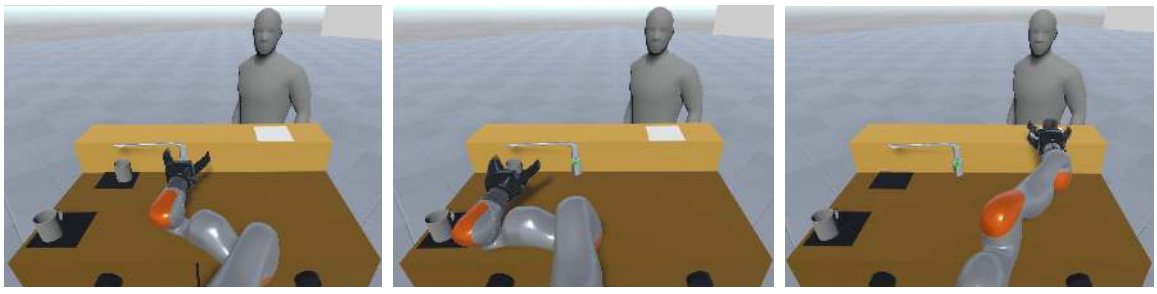


Figure 6.9: Multi-stage simulated manipulation task: first fill the cup, then grasp it, and then deliver it. To avoid spills, a pose constraint is enforced after the cup is grasped.

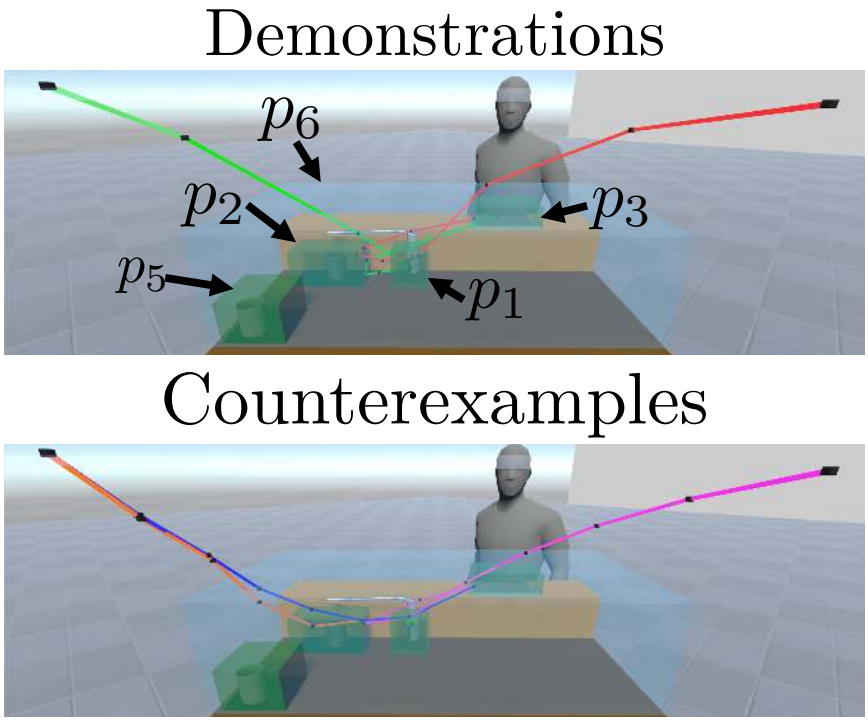


Figure 6.10: Demonstrations and counterexamples for the simulated manipulation task.

6.8.4 Multi-stage manipulation task

We consider the setup in Figs. 6.9, 6.10 of teaching a 7-DOF Kuka iiwa robot arm to prepare a drink: first move the end effector to the button on the faucet (p_1), then grasp the cup (p_2), then move the cup to the customer (p_3), all while avoiding obstacles. After grasping the cup, an end-effector pose constraint $(\alpha, \beta, \gamma) \in \mathcal{S}_4(\theta_4^p)$ (p_4) must be obeyed. We add two "distractor" APs: a different cup (p_5) and a region (p_6) where the robot can hand off the cup. We also modify the grammar to include the sequence operator $\mathcal{Q}$, (defined as before), and add an "after" operator

$$\varphi_1 \ \mathcal{T} \ \varphi_2 \doteq \Box_{[0,T_j-1]}(\varphi_2 \rightarrow \Box_{[0,T_j-1]}\varphi_1),$$

that is, φ_1 must hold after and including the first timestep where φ_2 holds. The true formula is:

$$\varphi^* = (p_1 \ \mathcal{Q} \ p_2) \wedge (p_2 \ \mathcal{Q} \ p_3) \wedge \Diamond_{[0,T_j-1]} p_3 \wedge (p_4 \ \mathcal{T} \ p_2).$$

We use a kinematic arm model: $j_{t+1}^i = j_t^i + u_t^i$, $i = 1, \ldots, 7$, where $\|u_t\|_2^2 \leq 1$ for all t. Two suboptimal human demonstrations ($\delta = 0.7$) optimizing $c(\xi_{xu}) = \sum_{t=1}^{T-1} \|j_{t+1} - j_t\|_2^2$ are recorded in a Unity virtual reality (VR) environment. We assume we have nominal estimates of the AP regions $\mathcal{S}_i(\theta_{i,\text{nom}}^p)$ (e.g., from a vision system), and we want to learn the θ^s and θ^p of φ^*. We use IPOPT *Wächter and Biegler* (2006) to solve the nonlinear optimization problems needed to compute counterexamples.

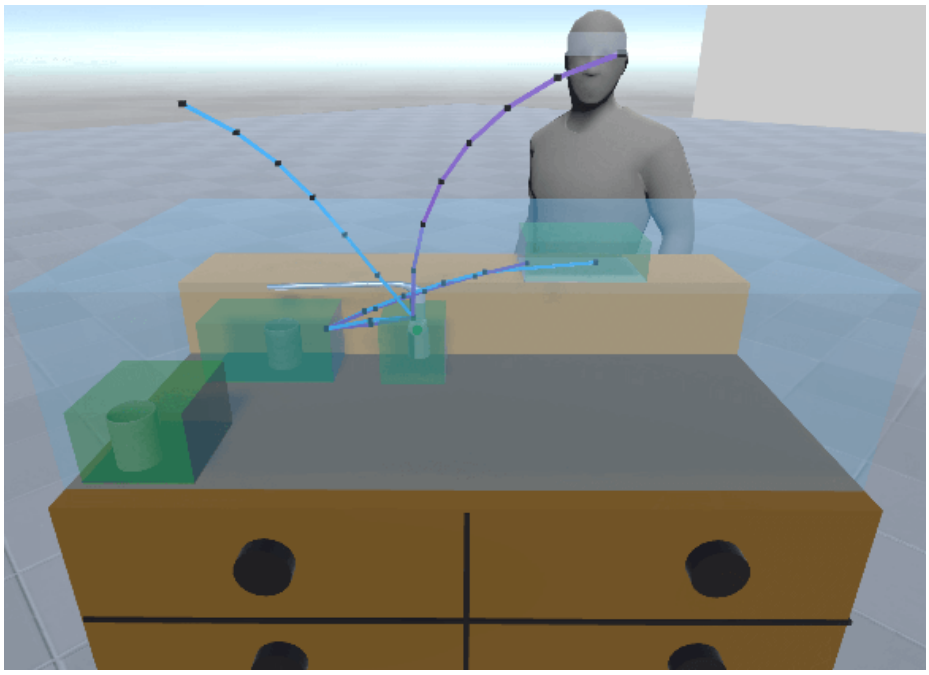


Figure 6.11: Trajectories planned using the learned LTL formula, for the simulated 7-DOF arm.

We run Alg. VI.1 with the 1-SO constraints (6.16), and encode the nominal θ_i^p by enforcing that $\Theta_i^p = \{\theta_i^p \mid \|\theta_i^p - \theta_{i,\text{nom}}^p\|_1 \leq 0.05\}$. At $N_{\text{DAG}} = 11$, the inner loop runs for 3 iterations (each taking 30 minutes on an i7-7700K processor), returning candidates

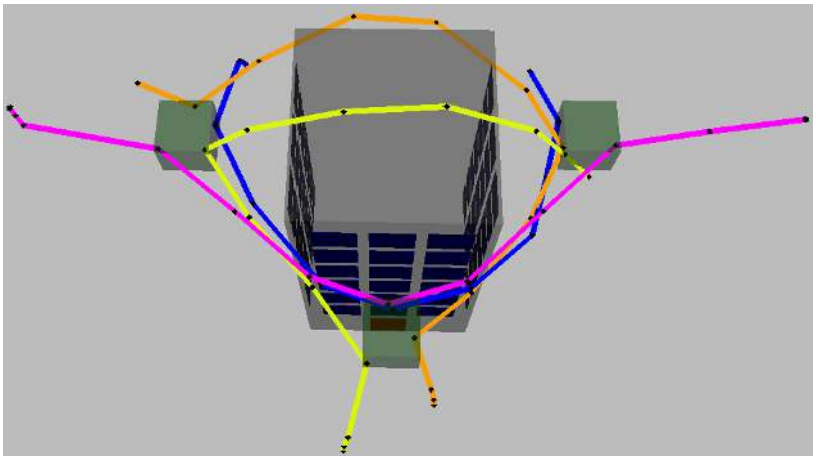
$$\varphi_1 = (p_1 \mathcal{Q} p_3) \wedge (p_2 \mathcal{Q} p_3) \wedge (\Diamond_{[0,T_j-1]} p_3) \wedge (p_4 \mathcal{T} p_3),$$
$$\varphi_2 = (p_1 \mathcal{Q} p_3) \wedge (p_2 \mathcal{Q} p_3) \wedge (\Diamond_{[0,T_j-1]} p_3) \wedge (p_4 \mathcal{T} p_2),$$

and $\varphi_3 = \varphi^*$. φ_1 says that before going to the customer, the robot has to visit the button and cup in any order, and then must satisfy the pose constraint after visiting the cup. φ_2 has the meaning of φ^*, except the robot can go to the button or cup in any order. Note that φ_3 is a stronger formula than φ_2, and φ_2 than φ_1; this is a natural result of the falsification loop, which returns incomparable or stronger formulas with more iterations, as the counterexamples rule out weaker or equivalent formulas. Also note that the distractor APs don't feature in the learned formulas, even though both demonstrations pass through p_6. This happens for two reasons: we increase N_{DAG} incrementally and there was no room to include distractor objects in the formula (since spec-optimality may enforce that p_1-p_3 appear in the formula), and even if N_{DAG} were not minimal, p_6 would not be guaranteed to show up, since visiting p_6 does not increase the trajectory cost.

We plot the counterexamples in Fig. 6.10: blue/purple are from iteration 1; orange is from iteration 2. They save cost by violating the ordering and pose constraints: from the left start state, the robot can save cost if it visits the cup before the button (blue, orange trajectories), and loosening the pose constraint can reduce joint space

cost (orange, purple trajectories). The right demonstration produces no counterexample in iteration 2, as it is optimal for this formula (changing the order does not lower the optimal cost). For the learned θ^p, $\theta^p_i = \theta^p_{i,\text{nom}}$ except for p_2, p_3, where the box shrinks slightly from the nominal; this is because by tightening the box, a Lagrange multiplier can be increased to reduce the KKT residual. We use the learned θ^p and θ^s to plan trajectories which complete the task from new initial conditions in the environment (Fig. 6.11).

Overall, this example suggests that Alg. VI.1 can recover θ^p and θ^s on a high-dimensional problem and ignore distractor APs, despite demonstration suboptimality.



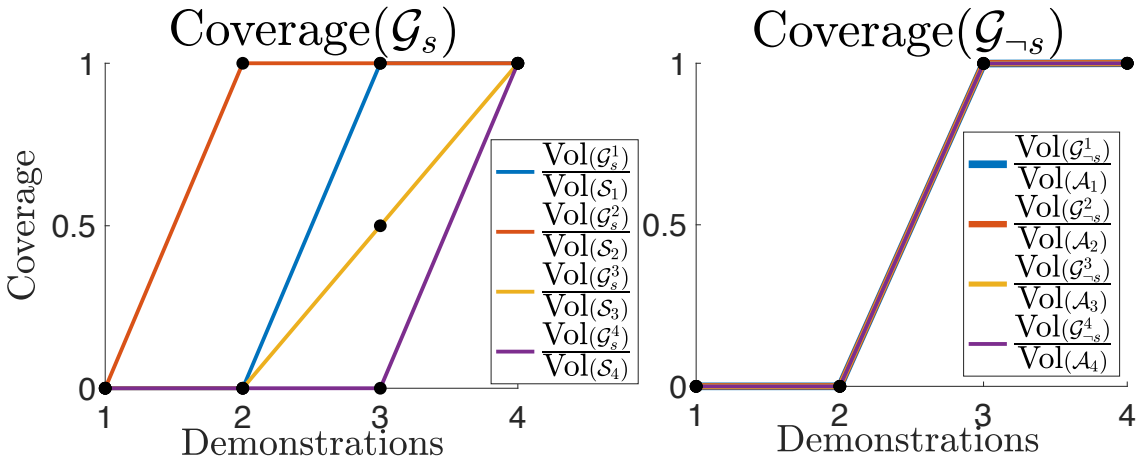


Figure 6.12: Quadrotor surveillance demonstrations (top) and learning curves (bottom).

6.8.5 Multi-stage quadrotor surveillance

We demonstrate that we can jointly learn θ^p, θ^s, and θ^c in one shot on a 12D nonlinear quadrotor system. The system dynamics for the quadrotor *Sabatino* (2015)

are:

$$\begin{bmatrix} \dot{\chi} \\ \dot{y} \\ \dot{z} \\ \dot{\alpha} \\ \dot{\beta} \\ \dot{\gamma} \\ \ddot{\chi} \\ \ddot{y} \\ \ddot{z} \\ \ddot{\alpha} \\ \ddot{\beta} \\ \ddot{\gamma} \end{bmatrix} = \begin{bmatrix} \dot{\chi} \\ \dot{y} \\ \dot{z} \\ \dot{\beta}\frac{\sin(\gamma)}{\cos(\beta)} + \dot{\gamma}\frac{\cos(\gamma)}{\cos(\beta)} \\ \beta\cos(\gamma) - \dot{\gamma}\sin(\gamma) \\ \dot{\alpha} + \dot{\beta}\sin(\gamma)\tan(\beta) + \dot{\gamma}\cos(\gamma)\tan(\beta) \\ -\frac{1}{m}[\sin(\gamma)\sin(\alpha) + \cos(\gamma)\cos(\alpha)\sin(\beta)]u_1 \\ -\frac{1}{m}[\cos(\alpha)\sin(\gamma) - \cos(\gamma)\sin(\alpha)\sin(\beta)]u_1 \\ g - \frac{1}{m}[\cos(\gamma)\cos(\beta)]u_1 \\ \frac{I_y - I_z}{I_x}\dot{\beta}\dot{\gamma} + \frac{1}{I_x}u_2 \\ \frac{I_z - I_x}{I_y}\dot{\alpha}\dot{\gamma} + \frac{1}{I_y}u_3 \\ \frac{I_x - I_y}{I_z}\dot{\alpha}\dot{\beta} + \frac{1}{I_z}u_4 \end{bmatrix}, \tag{6.19}$$

with control constraints $\|u_t\|_2 \leq 10$. We time-discretize the dynamics by performing forward Euler integration with discretization time $\delta t = 1.2$ seconds. The 12D state is $x = [\chi, y, z, \alpha, \beta, \gamma, \dot{x}, \dot{y}, \dot{z}, \dot{\alpha}, \dot{\beta}, \dot{\gamma}]^\top$, and the relevant constants are $g = -9.81\text{m/s}^2$, $m = 1\text{kg}$, $I_x = 0.5\text{kg}\cdot\text{m}^2$, $I_y = 0.1\text{kg}\cdot\text{m}^2$, and $I_z = 0.3\text{kg}\cdot\text{m}^2$.

We are given four demonstrations of a quadrotor surveilling a building (Fig. 6.12): it needs to visit three regions of interest (Fig. 6.12, green) while not colliding with the building. All visitation constraints can be learned directly with 1-SO (see Rem. VI.15) and collision-avoidance can also be learned with 1-SO, with enough demonstrations. The true formula is

$$\varphi^* = \Diamond_{[0,T_j-1]} p_1 \wedge \Diamond_{[0,T_j-1]} p_2 \wedge \Diamond_{[0,T_j-1]} p_3 \wedge \Box_{[0,T_j-1]} \neg p_4,$$

where p_1-p_3 represent the regions of interest and p_4 is the building. We aim to learn θ_i^p for the parameterization $\mathcal{S}_i(\theta_i^p) = \{[I_{3\times 3}, -I_{3\times 3}]^\top [x, y, z]^\top \leq \theta_i^p\}$, assuming $\theta_{4,6}^p = 0$ (the building is not hovering). The demonstrations minimize $c(\xi_{xu}, \theta^c) = \sum_{r\in R}\sum_{t=1}^{T-1} \gamma_r (r_{t+1} - r_t)^2$, where $R = \{x, y, z, \dot{\alpha}, \dot{\beta}, \dot{\gamma}\}$ and $\gamma_r = 1$, i.e., equal penalties to path length and angular acceleration. We assume $\gamma_r \in [0.1, 1]$ and is unknown: we want to learn the cost weights for each state.

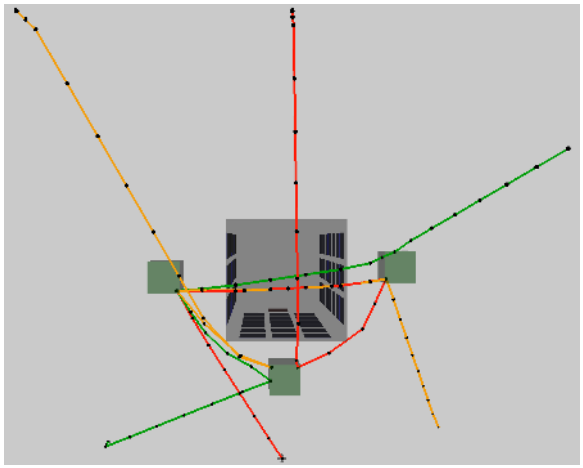


Figure 6.13: Trajectories planned using the learned LTL formula, for the quadrotor system.

Solving Prob. VI.11 with 1-SO conditions (at $N_{\text{DAG}} = 12$) takes 44 minutes and recovers θ^p, θ^s, and θ^c in one shot. To evaluate the learned θ^p, we show in Fig. 6.12 that the coverage of the $\mathcal{G}_s^i$ and $\mathcal{G}_{\neg_s}^i$ for each p_i (computed by fixing the learned θ^s and running Prob. VI.6) monotonically increases with more data. In terms of recovered θ^s, with one demonstration, we return

$$\varphi_1 = \Diamond_{[0,T_j-1]} p_2 \wedge \Diamond_{[0,T_j-1]} p_3 \wedge \Diamond_{[0,T_j-1]} p_4 \wedge \Box_{[0,T_j-1]} \neg p_1.$$

This highlights the fact that since we are not provided labels, there is an inherent ambiguity of how to label the regions of interest (i.e., p_i, $i = 1, \ldots, 3$ can be associated with any of the green boxes in Fig. 6.12 and be consistent). Also, one of the regions of interest in φ gets labeled as the obstacle (i.e., p_1 and p_4 are swapped), since one demonstration is not enough to disambiguate which of the four p_i should touch the ground. Note that this ambiguity can be eliminated if labels are provided (see Sec. 6.6.1) or if more demonstrations are provided: for two and more demonstrations, we learn $\varphi_i = \varphi^*$, $i = 2, \ldots, 4$. When using all four demonstrations, we recover the cost parameters θ^c and structure θ^s exactly, i.e., $\varphi(\hat{\theta}^s, \hat{\theta}^p) = \varphi^*$, and fixing the learned θ^s and running Prob. VI.6 returns $\mathcal{G}_s^i = \mathcal{S}_i$ and $\mathcal{G}_{\neg_s}^i = \mathcal{A}_i$, for all i. The learned θ^c, θ^s, and θ^p are used to plan trajectories that efficiently complete the task for different initial and goal states. Furthermore, assuming that the parameterization is correct, these plans are guaranteed to satisfy the true LTL formula; these trajectories are presented in Fig. 6.13.

Overall, this example suggests that our method can jointly recover a consistent set of θ^p, θ^s, and θ^c for high-dimensional systems.

6.9 Physical experiments

To demonstrate that our method can scale to handle the challenges of real hardware, we use our method to learn a real-world multi-stage manipulation task. A video

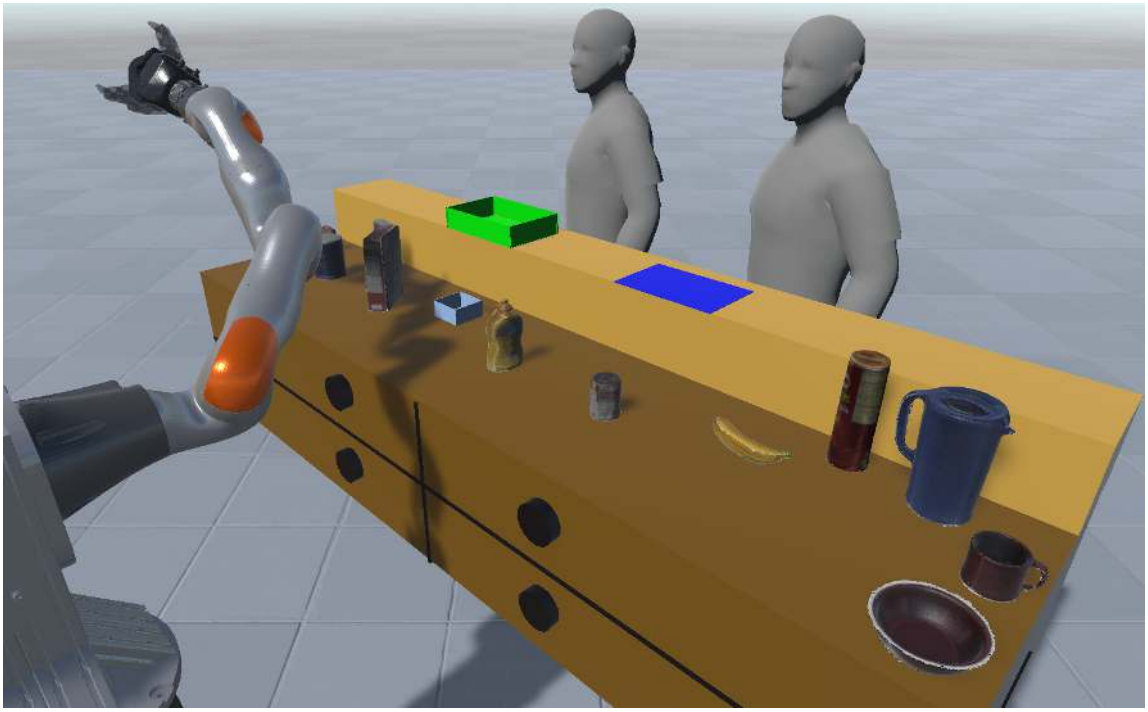
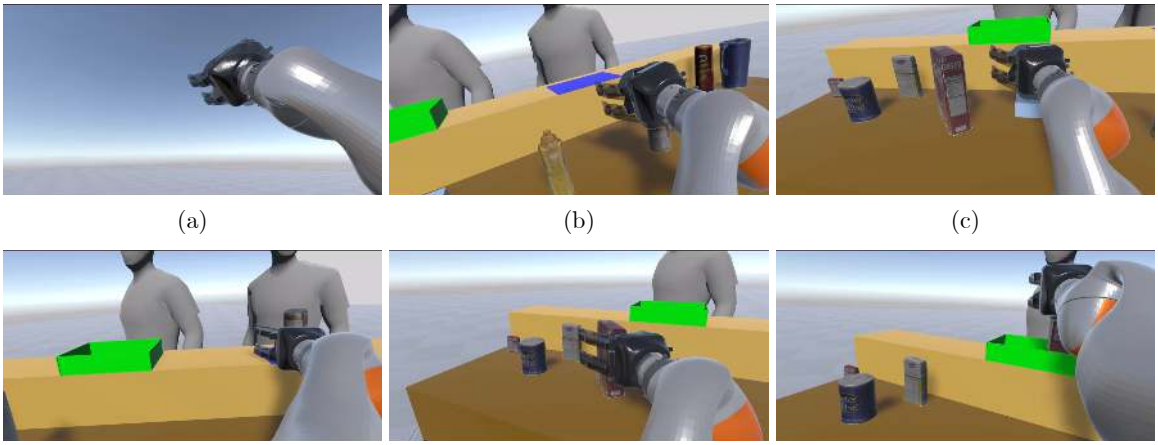


Figure 6.14: We build a Unity virtual reality environment to collect demonstrations for the real-world object delivery manipulation task.



(a) (b) (c)

(d) (e) (f)

Figure 6.15: One demonstration is recorded in the Unity virtual reality environment for the object delivery task, seen here from a first-person perspective. (a) Initial state. (b) First, grasp the soup. (c) Next, place the soup in the blue box, avoiding the mustard bottle which is in the way. (d) Place the box with the soup in the blue delivery region while satisfying a pose constraint. (e) Move to grasp the Cheez-It box. (f) Place the Cheez-It box in the green delivery box.

of our physical experiment can be found in the supplementary material.

6.9.1 Environment and task description

Consider a tabletop manipulation task where the arm needs to retrieve several objects, put them in boxes, and deliver them in a particular order (see Fig. 6.14). Specifically, the task of interest is to first place a can of soup into a box (Fig. 6.15 (b)-(c)), to then deliver that box to a blue delivery region (Fig. 6.15 (d)). Next, the robot must move a Cheez-It box into a box located at a green delivery region (Fig. 6.15 (e)-(f)). Finally, while the box containing the soup is grasped by the robot, the robot must keep its end effector upright so that the soup does not fall out of the box. The robot should also avoid colliding with the furniture as well as any other objects in the scene. There are a total of 11 objects in the scene, not including the delivery boxes or the furniture, which are taken from the YCB dataset *Çalli et al.* (2017).

To describe the aforementioned task concisely in LTL, we define another new grammar element:

$$\varphi_1 \; \mathcal{M} \; \varphi_2 \doteq \square_{[0,T_j-1]}((\varphi_2 \rightarrow \varphi_1)) \wedge \Diamond_{[0,T_j-1]}\varphi_2,$$

i.e., if φ_2 holds, then φ_1 must also hold, and φ_2 must eventually hold. We define the following atomic propositions:

- p_S: The soup is grasped
- p_B: The movable box is grasped
- p_{G1}: The end effector is inside the blue delivery region
- p_C: The Cheez-It box is grasped
- p_{G2}: The end effector is inside the green delivery region
- p_P: The end effector is pointed upwards
- p_{D1}: End effector is within 0.05 distance of the gelatin
- p_{D2}: End effector is within 0.05 distance of the bowl
- p_{D3}: End effector is within 0.05 distance of the Master Chef coffee can
- p_{D4}: End effector is within 0.05 distance of the sugar
- p_{D5}: End effector is within 0.05 distance of the mustard bottle
- p_{D6}: End effector is within 0.05 distance of the banana
- p_{D7}: End effector is within 0.05 distance of the Pringles
- p_{D8}: End effector is within 0.05 distance of the pitcher

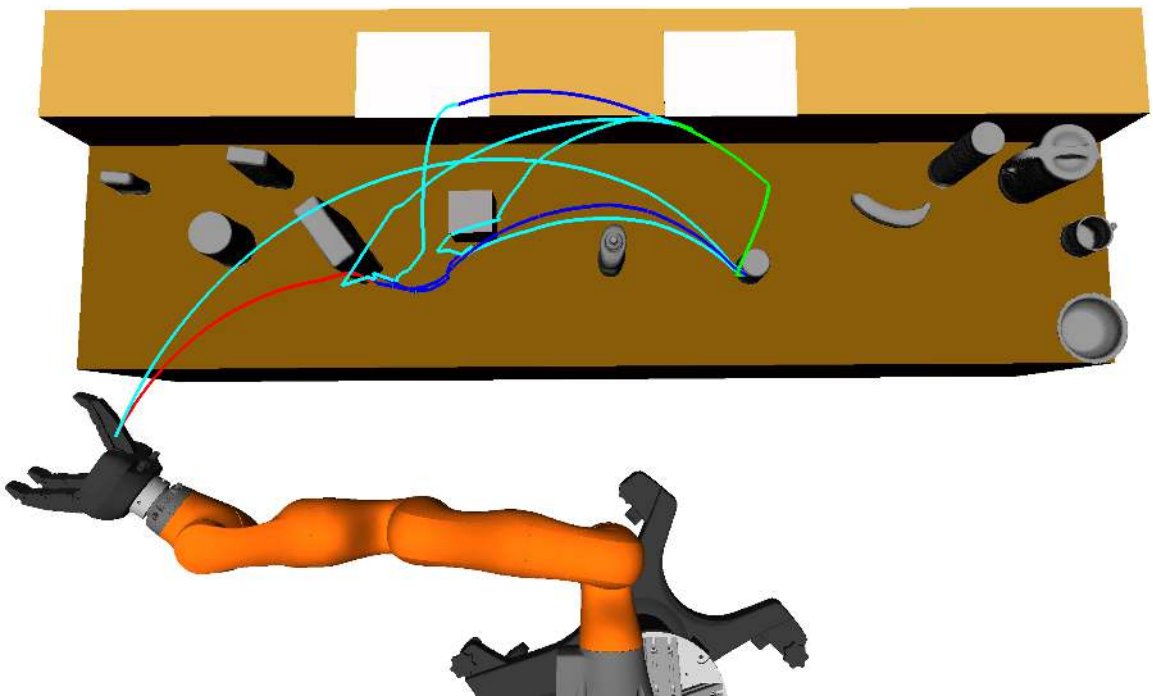


Figure 6.16: Counterexample visualization on the object delivery task. The red, green blue, and cyan trajectories correspond to φ_2, φ_3, φ_8, and φ_{13}, respectively, as described in Sec. 6.9.2.

- p_{D9}: End effector is within 0.05 distance of the mug

We can then write an LTL formula which enforces the task as

$$\varphi^* = (p_S \ \mathcal{M} \ p_B) \ \wedge \ (p_B \ \mathcal{M} \ p_{G1}) \ \wedge \ (p_C \ \mathcal{M} \ p_{G2}) \wedge \\ (p_{G1} \ \mathcal{Q} \ p_C) \ \wedge \ (p_P \ \mathcal{M} \ p_B).$$

The first through fourth clauses enforce that the soup, moving box, blue goal region, Cheez-It, and green delivery region are visited in the correct order, while the fifth clause enforces that the pose constraint is satisfied when the moving box is grasped. This is not overly restrictive, since per the first clause, it is not possible for the moving box to be grasped without the soup also being grasped. Note that we assume the demonstrator performs collision avoidance by avoiding contact with any object which is not the current grasp target.

6.9.2 LTL formula learning

For this experiment, we seek to learn the LTL formula structure θ^s while the AP parameters θ^p and cost function parameters γ are assumed known. This is reasonable for this example, since the APs detailed in Sec. 6.9.1 can be readily measured and the suboptimality parameter δ can be used to handle an imprecisely-known cost function. Specifically, we assume the cost function is

$$c(\xi, \gamma) = \sum_{t=1}^{T-1} \|j_{t+1} - j_t\|_2^2 + c_{\text{grasp}} \sum_{o \in \mathcal{O}} \sum_{t=1}^{T} z^o_{\text{grasp, t}},$$

where j_t denotes the arm joint values at time t, $z^o_{\text{grasp, t}} \in \{0, 1\}$ evaluates to 1 if object o is grasped at time t and 0 otherwise, $\mathcal{O}$ is the set of all manipulable objects, and $c_{\text{grasp}} = 0.01$ is a small penalty which discourages the unnecessary grasping of objects. Note that the learning is relatively robust to the specific value of c_{grasp}, as long as c_{grasp} is kept small enough such that the grasp cost term does not outweigh the path length term (in our experiments, this holds if $c_{\text{grasp}} \leq 0.115$). Mapping back to the notation of Prob. VI.1, the state x_t contains the joint values j_t and the grasp status of each object z^o_t, while the control input contains the joint velocities and a binary variable for each object to model grasping and releasing. The dynamics are constructed such that the grasp input for a given object is nullified if the end effector is far from that object.

We obtain one demonstration of this task which is recorded in a Unity VR environment (see Fig. 6.14 and 6.15). The demonstration consists of the state-control trajectory of the arm, as well as a binary trajectory for each object, evaluating to 0 or 1 at a given timestep depending on if that object is currently grasped. Furthermore, the initial configurations of all of the objects are given. Note that this information is sufficient to reconstruct the value of every atomic propositions. We also note that the VR environment does not simulate the grasp physics, and simply allows the demonstrator to attach an object to the grippers when it is close by. To learn θ^s, we run Alg. VI.1, where Prob. VI.11 uses the variant described in Sec. 6.6.2. We elect to use this variant instead of the original Prob. VI.11 as in the simulated manipulation example (Sec. 6.8.4) since there are many more APs in this example (15 compared to 6 in Sec. 6.8.4), causing the original Prob. VI.11 to be slow. We allocate one node for each AP, four "$\wedge$" nodes, four "$\mathcal{M}$" nodes, one "$\mathcal{Q}$" node, and one "$\Diamond$" node. We use a suboptimality parameter $\delta = 0.1$. Running Alg. VI.1 generates 13 falsified candidate LTL formulas, including the following:

- $\varphi_2 = (p_C \; \mathcal{M} \; p_{G2}) \wedge (p_S \; \mathcal{M} \; p_P) \wedge (p_B \; \mathcal{Q} \; p_{G1}) \wedge (p_P \; \mathcal{M} \; p_B) \wedge (\Diamond p_{D5})$. This formula does not capture that the Cheez-Its should only be grasped after the soup has been grasped.

- $\varphi_3 = (p_B \; \mathcal{M} \; p_P) \wedge (p_P \; \mathcal{M} \; p_B) \wedge (p_{G1} \; \mathcal{Q} \; p_C) \wedge (p_C \; \mathcal{M} \; p_{G2}) \wedge (p_S \; \mathcal{M} \; p_{G1})$. This formula does not capture that the soup should be contained in the box upon delivery.

- $\varphi_8 = (p_C \; \mathcal{M} \; p_{G2}) \wedge (p_S \; \mathcal{M} \; p_P) \wedge (p_B \; \mathcal{M} \; p_{G1}) \wedge (p_P \; \mathcal{Q} \; p_C) \wedge (\Diamond \; p_{D5})$. This formula does not capture that the Cheez-Its should only be grasped after the movable box has been grasped.

- $\varphi_{13} = (p_P \; \mathcal{M} \; p_{G1}) \wedge (p_C \; \mathcal{M} \; p_{G2}) \wedge (p_B \; \mathcal{M} \; p_P) \wedge (p_S \; \mathcal{M} \; p_P) \wedge (p_{G1} \; \mathcal{Q} \; p_C)$. This formula does not enforce the pose constraint at the correct timesteps.

Figure 6.17: Setup of the object delivery task in the real world. The small brown box corresponds to the small blue box in the VR environment, while the large brown box corresponds to the green box in the VR environment.

The candidate LTL formulas are falsified by the counterexample generation, for which we employ TrajOpt *Schulman et al.* (2014) as the nonlinear trajectory optimizer (see Sec. 6.4.3). We visualize the counterexamples for φ_2, φ_3, φ_8, and φ_{13} in Fig. 6.16. One can observe that the missing constraints in these candidate LTL formulas accept lower-cost trajectories (achieved for example by not delivering the goods in the desired order, or by not picking up particular objects) which contradict the optimality of the demonstration. We emphasize that our method can ignore the large number of distractor objects. Limiting the expressibility of the DAG by limiting the number of nodes encourages the learned formula to be parsimonious, since the free nodes will be needed to explain demonstrator optimality rather than involving the distractor objects. In the 14th iteration, our method terminates after a total of 5 minutes, returning the true formula φ^*.

(a) (b) (c)

Figure 6.18: Object segmentation. (a) RGBD data provided by the Kinect sensor. (b) Segmented image. (c) Segmented point cloud, which is used to infer object poses.

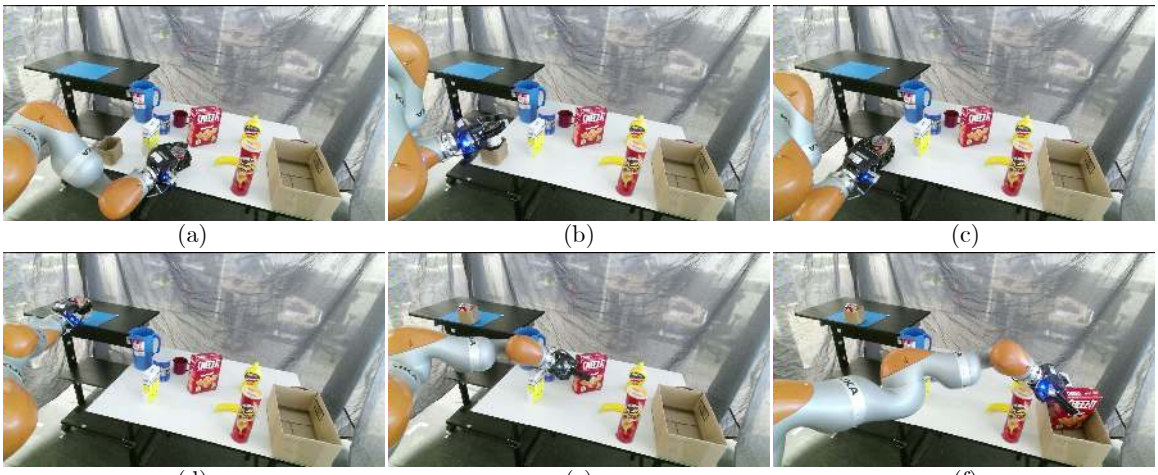


Figure 6.20: Executed trajectory on the real robot. The robot first grasps the tomato soup (a), moves to place it inside the movable box (b), drops the soup into the box and grasps the loaded box (c), and moves the loaded the box to the blue delivery region (d). The robot then moves to grasp the Cheez-It box (e), and finally places it in the box located at the green delivery region.

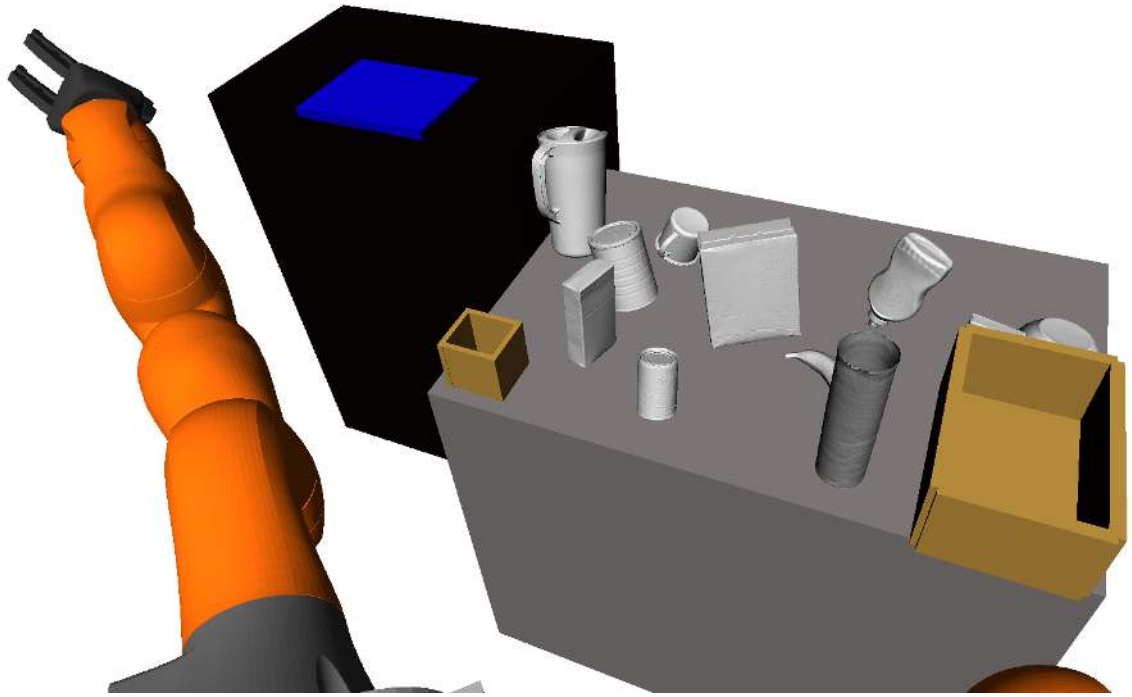


Figure 6.19: Planning environment used. Object poses are recovered from the segmented depth cloud by running ICP.

6.9.3 Real-world planning and execution

Now that an LTL formula describing the desired task has been learned, we seek to use the learned formula to plan in the real world. We work with the real-world setup in Fig. 6.17. This setup has different furniture and object configurations compared to the VR demonstration environment. However, recall that since the learned LTL formula is parameterized by the APs, the learned LTL formula is not hardcoded to specific configurations and can handle changes in the object locations.

To reflect the realistic situation where the robot may be tasked to find and deliver a set of objects scattered across the workspace with *a priori* unknown locations, we assume that the locations of the delivery regions and the movable box are known, while the YCB objects have unknown location. The movable blue box in the VR environment corresponds to the small brown box on the left in Fig. 6.17, while the green box in the VR environment corresponds to the big brown box on the right in Fig. 6.17.

To apply our learned LTL formula, we first estimate the poses of the YCB objects using RGBD (image and point cloud) data provided by a Kinect sensor mounted above the base of the arm. We do so by leveraging the deep learning-based object segmentation framework in *Zhou et al.* (2019) and train it on the YCB object dataset. The trained network takes the Kinect RGBD data as input and returns a segmented point cloud (Fig. 6.18). We use the iterative closest point (ICP) algorithm *Rusu and Cousins* (2011) with 1000 random initializations to estimate the object poses from the segmented point cloud by fitting them to the source point clouds. We visualize the objects at their estimated poses in an Openrave environment, which we also use for trajectory planning (Fig. 6.19). We note that due to occlusions and sensor noise present in the point cloud data, the poses recovered for the objects further from the Kinect can suffer from rotational inaccuracies (e.g., the mustard bottle is upside down and the pitcher is rotated around 90 degrees). While this degree of pose accuracy is sufficient to complete our task, we also note that more sophisticated methods can be employed (e.g., *Deng et al.* (2019), which provides good pose recovery on the YCB dataset in the presence of occlusions and object symmetry).

Now that the object poses have been determined (and thus so have the APs), we can employ the learned LTL formula to plan in the real environment. To do so, we solve Prob. VI.1 for $\varphi(\theta^s, \hat{\theta}^p)$ using the approach detailed in Sec. 6.4.3. Specifically, we construct a high-level plan $\mathbf{Z}$ by solving a MILP, and then find a low-level joint trajectory which is consistent with $\mathbf{Z}$ with the trajectory optimization algorithm TrajOpt *Schulman et al.* (2014). Like for the counterexample generation, we choose TrajOpt instead of IPOPT as it is better tuned for manipulation in cluttered environments. Snapshots of the executed plan are presented in Fig. 6.20. Please see the supplementary video for a full visualization.

Overall, this experiment suggests that our learned LTL formulas can be used to transfer complex long-horizon task specifications across environments, and that the method is applicable to high-dimensional robotic systems acting in the real world.

6.10 Conclusion

This chapter presents a method that learns LTL formulas with unknown atomic propositions and logical structure from only positive demonstrations, assuming the demonstrator is optimizing an uncertain cost function. We leverage both implicit and explicit optimality conditions on the demonstrations, namely the KKT conditions and algorithmically-generated lower-cost counterexample trajectories, respectively, in order to reduce the hypothesis space of LTL specifications consistent with the demonstrations. The generated lower-cost counterexample trajectories, together with the rejected candidate LTL formulas which admitted them, are concrete examples of the alternative behaviors and task specifications rejected by our method, which can make our approach more explainable for an end user. We also derive theoretical guarantees for our method and demonstrate its applicability across a wide range of experiments in simulation and hardware. Specifically, we show that our method outperforms baseline approaches (Sec. 6.8.1), can learn abstract high-level task structure shared across demonstrations, which can transfer to tasks in different environments (Sec. 6.8.3 and Sec. 6.9), and scales to high-dimensional systems in simulation (Sec. 6.8.4 and Sec. 6.8.5) and in the real world (Sec. 6.9).

In future work, we aim to robustify our method to mislabeled demonstrations, explicitly consider demonstration suboptimality arising from risk, and reduce our method's computation time. We are also interested in integrating the methods presented in this chapter with our results on uncertainty-aware constraint learning (Chapter VII) in order to plan with uncertain LTL formulas.

CHAPTER VII

Uncertainty-Aware Constraint Learning and Planning via Constraint Beliefs

In this chapter, we present a method for learning to satisfy uncertain constraints from demonstrations. Our method uses robust optimization to obtain a belief over the potentially infinite set of possible constraints consistent with the demonstrations, and then uses this belief to plan trajectories that trade off performance with satisfying the possible constraints. We use these trajectories in a closed-loop policy that executes and replans using belief updates, which incorporate data gathered during execution. We derive guarantees on the accuracy of our constraint belief and probabilistic guarantees on plan safety. We present results on a 7-DOF arm and 12D quadrotor, showing our method can learn to satisfy high-dimensional (up to 30D) uncertain constraints, and outperforms baselines in safety and efficiency. This chapter is based on the paper *Chou et al.* (2020a).

7.1 Introduction

A core problem in learning from demonstration (LfD), and constraint-learning in particular, is the unidentifiability of the constraints: there is often an infinite set of *possible constraints* which are sufficient to explain a demonstration. While previous work (Chapter IV) has evaded this problem when planning with the learned constraint by planning guaranteed-safe trajectories that satisfy *all possible constraints* consistent with the data, this is impossible in most realistic scenarios, where the set of possible constraints is so large that the planning problem becomes infeasible. For example, consider planning for an arm in a cluttered home environment: unless the demonstrations activate each of the multitude of constraints, we cannot claim that a trajectory is guaranteed-safe.

Our insight is that to plan under large constraint uncertainty, it is vital to plan trajectories that trade off safety and efficiency by reasoning over the set of possible constraints, and to update this set using constraint information gathered when executing these trajectories. Specifically, we leverage robust optimization and Bayesian inference to obtain and update our belief over the possible constraints consistent with the demonstrations and gathered data. Then, we propose a policy for adaptively satisfying the constraints which interleaves chance-constrained planning, execution, and

belief updates until the task is completed. This chapter makes following specific contributions:

1. We show how to extract *all possible constraints*, for some constraint parameterization, consistent with a set of locally-optimal demonstrations, which we use to construct a belief over constraints.

2. We provide a novel method for planning approximately-optimal open-loop trajectories between new start and goal states, which are safe under the constraint belief with a prescribed probability.

3. We show how to use these open-loop trajectories to construct a closed-loop policy to adaptively satisfy the uncertain constraints, incorporating constraint information observed during execution.

4. We theoretically analyze our algorithm, proving the completeness of constraint extraction for various constraint parameterizations and providing probabilistic safety guarantees for our planner.

5. We evaluate our method by planning for a 7-DOF arm and a quadrotor with uncertain high-dimensional constraints, showing that our methods outperform baselines in efficiency and safety.

7.2 Preliminaries and Problem Setup

We consider demonstrations performed on systems $x_{t+1} = f(x_t, u_t, t)$, $x \in \mathcal{X}$, $u \in \mathcal{U}$ completing tasks $\Pi \in \mathcal{P}$, represented as constrained optimizations over state/control trajectories $\xi_{xu} \doteq (\xi_x, \xi_u)$:

Problem VII.1 (Forward (demonstrator's) problem / "task" Π)**.**

$$\begin{array}{lll} \underset{\xi_{xu}}{\text{minimize}} & c_\Pi(\xi_{xu}) & \\ \text{subject to} & \phi(\xi_{xu}) \in \mathcal{S}(\theta) \subseteq \mathcal{C} & \Leftrightarrow \quad \mathbf{g}_{\neg k}(\xi_{xu}, \theta) \leq \mathbf{0} \\ & \bar{\phi}(\xi_{xu}) \in \bar{\mathcal{S}} \subseteq \bar{\mathcal{C}}, \quad \phi_\Pi(\xi_{xu}) \in \mathcal{S}_\Pi \subseteq \mathcal{C}_\Pi & \Leftrightarrow \quad \mathbf{h}_k(\xi_{xu}) = \mathbf{0}, \quad \mathbf{g}_k(\xi_{xu}) \leq \mathbf{0} \end{array}$$

where $c_\Pi(\cdot)$ is task-dependent and $\phi(\cdot)$ maps from trajectories to constraint space $\mathcal{C}$; elements of $\mathcal{C}$ are denoted *constraint states* $\kappa \in \mathcal{C}$. $\bar{\phi}(\cdot)$ and $\phi_\Pi(\cdot)$ map to constraint spaces $\bar{\mathcal{C}}$ and $\mathcal{C}_\Pi$, containing a known shared safe set $\bar{\mathcal{S}}$ and task-dependent safe set $\mathcal{S}_\Pi$; we embed the dynamics in $\bar{\mathcal{S}}$ and start/goal constraints in $\mathcal{S}_\Pi$. We group the constraints of Prob. VII.1 as (in)equality (ineq/eq) and (un)known ($\neg k/k$), where $\mathbf{h}_k(\xi_{xu}) \in \mathbb{R}^{N_k^{\text{eq}}}$, $\mathbf{g}_k(\xi_{xu}) \in \mathbb{R}^{N_k^{\text{ineq}}}$, and $\mathbf{g}_{\neg k}(\xi_{xu}, \theta) \in \mathbb{R}^{N_{\neg k}^{\text{ineq}}}$, and let $g(\kappa, \theta) \doteq \max_{i \in \{1,\ldots,N_{\neg k}^{\text{ineq}}\}} \big(g_{i,\neg k}(\kappa, \theta)\big)$. Let the unknown safe and unsafe sets defined by parameter $\theta \in \Theta \subseteq \mathbb{R}^d$ be $\mathcal{S}(\theta)$ and $\mathcal{A}(\theta)$, respectively:

$$\mathcal{S}(\theta) \doteq \{\kappa \in \mathcal{C} \mid g(\kappa, \theta) \leq 0\} \tag{7.1}$$

$$\mathcal{A}(\theta) \doteq \mathcal{S}(\theta)^c = \{\kappa \in \mathcal{C} \mid g(\kappa, \theta) > 0\} \tag{7.2}$$

We assume each state-control demonstration ξ^{loc} approximately solves Prob. VII.1 to local optimality, satisfying Prob. VII.1's Karush-Kuhn-Tucker (KKT) conditions *Boyd and Vandenberghe* (2004) within a tolerance. Intuitively, this means ξ^{loc} is feasible for Prob. VII.1 (it remains within the safe set $\mathcal{S}(\theta)$ and satisfies the known constraints) and is within the neighborhood of a local optimum. With Lagrange multipliers λ, ν, the relevant KKT conditions for the jth demonstration ξ_j^{loc}, denoted $\text{KKT}(\xi_j^{\text{loc}})$, are:

$$\mathbf{g}_{\neg k}(\xi_j^{\text{loc}}, \theta) \leq \mathbf{0} \tag{7.3a}$$

$$\boldsymbol{\lambda}_k^j \odot \mathbf{g}_k(\xi_j^{\text{loc}}) = \mathbf{0}, \qquad \boldsymbol{\lambda}_k^j \geq \mathbf{0} \tag{7.3b}$$

$$\boldsymbol{\lambda}_{\neg k}^j \odot \mathbf{g}_{\neg k}(\xi_j^{\text{loc}}, \theta) = \mathbf{0}, \qquad \boldsymbol{\lambda}_{\neg k}^j \geq \mathbf{0} \tag{7.3c}$$

$$\nabla_{\xi_{xu}} c_\Pi(\xi_j^{\text{loc}}) + \boldsymbol{\lambda}_k^{j\top} \nabla_{\xi_{xu}} \mathbf{g}_k(\xi_j^{\text{loc}}) + \boldsymbol{\lambda}_{\neg k}^{j\top} \nabla_{\xi_{xu}} \mathbf{g}_{\neg k}(\xi_j^{\text{loc}}, \theta) + \boldsymbol{\nu}_k^{j\top} \nabla_{\xi_{xu}} \mathbf{h}_k(\xi_j^{\text{loc}}) = \mathbf{0} \tag{7.3d}$$

Problem VII.2 (Inverse constraint learning problem)**.**

$$\begin{array}{ll} \text{find} & \theta, \mathcal{L} \doteq \{\boldsymbol{\lambda}_k^j, \boldsymbol{\lambda}_{\neg k}^j, \boldsymbol{\nu}_k^j\}_{j=1}^{N_s} \\ \text{subject to} & \{\text{KKT}(\xi_j^{\text{loc}})\}_{j=1}^{N_s} \end{array}$$

where $\nabla_{\xi_{xu}}(\cdot)$ differentiates with respect to a flattened ξ_{xu} and $\odot$ denotes element-wise product. (7.3a) enforces primal feasibility, (7.3b)-(7.3c) enforces complementary slackness, and stationarity (7.3d) enforces the demonstration cannot be locally improved. As in Chapter IV, we can solve Prob. VII.2 to find constraints that make the demonstrations locally-optimal by finding a θ and Lagrange multipliers which are together consistent with the KKT conditions of the demonstrations. To handle approximate local-optimality in Prob. VII.2, we relax constraints (7.3b)-(7.3d) to penalties. This framework can also learn unknown cost function parameters (cf. App. B.2). Let $\mathcal{F}$ denote the feasible set of Prob. VII.2. Denote the projection of $\mathcal{F}$ onto Θ (the set of all consistent constraints θ) as $\mathcal{F}_\theta$, and the projection of $\mathcal{F}_\theta$ onto $\mathcal{C}$ (the set of possibly-unsafe constraint states) as $\text{proj}_{\mathcal{C}}(\mathcal{F}_\theta)$:

$$\mathcal{F}_\theta \doteq \{\theta \mid \exists \mathcal{L} : (\theta, \mathcal{L}) \in \mathcal{F}\} \tag{7.4}$$

$$\text{proj}_{\mathcal{C}}(\mathcal{F}_\theta) \doteq \{\kappa \in \mathcal{C} \mid \exists \theta \in \mathcal{F}_\theta, g(\kappa, \theta) > 0\} \tag{7.5}$$

While Prob. VII.2 returns *one possible* θ, there can be an *infinite set* of possible θ: $\mathcal{F}_\theta$. Thus, $\mathcal{F}_\theta$ represents the constraint uncertainty. We use $\mathcal{F}_\theta$ to build a constraint belief (Sec. 7.3) and use $\text{proj}_{\mathcal{C}}(\mathcal{F}_\theta)$ for planning (Sec. 7.4). Finally, let the set of learned guaranteed-safe/unsafe constraint states be $\mathcal{G}_s$ and $\mathcal{G}_{\neg s}$, where κ is learned guaranteed (un)safe if it is marked (un)safe for all $\theta \in \mathcal{F}_\theta$ (cf. Fig. 7.2):

$$\mathcal{G}_s \doteq \bigcap_{\theta \in \mathcal{F}_\theta} \{\kappa \mid g(\kappa, \theta) \leq 0\} \tag{7.6}$$

$$\mathcal{G}_{\neg s} \doteq \bigcap_{\theta \in \mathcal{F}_\theta} \{\kappa \mid g(\kappa, \theta) > 0\} \tag{7.7}$$

Previous work (Chapter IV) plans guaranteed-safe trajectories by enforcing that they always remain in $\mathcal{G}_s$, but $\mathcal{G}_s$ can be tiny or disconnected (Fig. 7.2), making planning infeasible. In this work, we allow plans to pass through possibly-unsafe space $\text{proj}_{\mathcal{C}}(\mathcal{F}_\theta)$, but seek to minimize constraint violations.

Problem statement: We are given N_s demonstrations $\{\xi_j^{\text{loc}}\}_{j=1}^{N_s}$, known shared and task-dependent safe sets $\bar{\mathcal{S}}$ / $\mathcal{S}_\Pi$, and a prior $p(\theta)$ over the unknown constraint. Our goals are: 1) recover the set of all constraint parameters $\mathcal{F}_\theta \subseteq \Theta$ consistent with the demonstrations to obtain a constraint belief $b_{\text{dem}}(\theta) \doteq p(\theta \mid \{\xi_j^{\text{loc}}\}_{j=1}^{N_s}) \in \mathcal{P}(\Theta)$, and 2) compute a policy $\pi(\cdot, \cdot, \cdot) : \mathcal{P}(\Theta) \times \mathcal{X} \times (\mathcal{O})^* \to \mathcal{U}$, which takes a prior, start state x_0, and a sequence of constraint observations, and returns a control input u, and completes a task (in this chapter, we consider a task as reaching a goal x_g from x_0) while minimizing one of two objectives. In the first variant, denoted Prob. MCV, we want to reach the goal with the minimum number of expected constraint violations. In the second variant, denoted Prob. MEC, we want to minimize the expected cost of some general objective function.

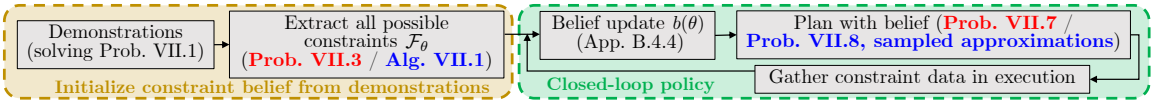


Figure 7.1: Overall method flow for adaptive planning from demonstrations. We refer to both the ideal (red), but intractable, subproblems, as well as the tractable (blue) variants of those subproblems.

Method overview: To prime the reader, we outline two variants of our method in Fig. 7.1: 1) an ideal variant that requires the solution of intractable optimizations, and 2) tractable variants which approximate the idealized problems or exploit simplifying problem structure. For closed-loop planning, both variants compute the constraint belief (Sec. 7.3), then iteratively plan with the belief, update the belief with constraint data measured in execution, and replan with the updated belief (Sec. 7.4).

7.3 Obtaining a belief over constraints

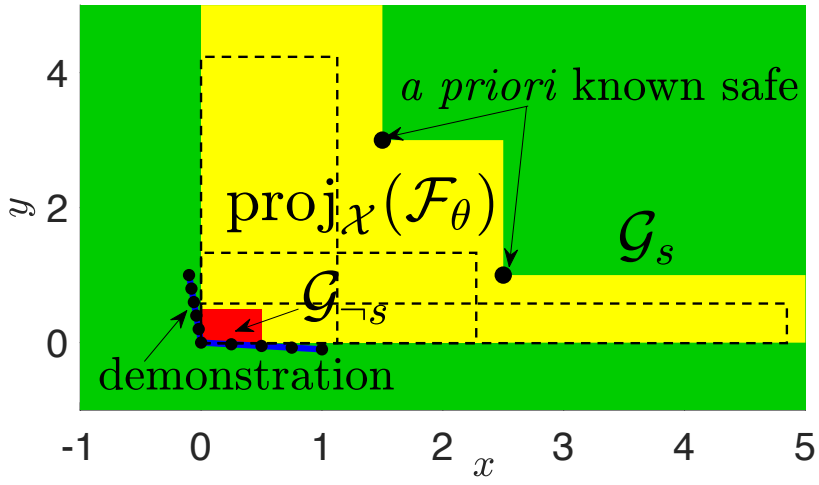


Figure 7.2: $\mathcal{F}_\theta$ for a one-box parameterization of $\mathcal{A}(\theta)$, induced by a demo. and two safe states, projected onto $\mathcal{X}$. With the data, the upper x / y bounds $\bar{x}(\theta)$ / $\bar{y}(\theta)$ remain uncertain. Also: some possible $\mathcal{A}(\theta)$ (dotted).

Extracting $\mathcal{F}_\theta$ is crucial for obtaining an accurate belief over constraints. In this section, we show how to use robust optimization to obtain $\mathcal{F}_\theta$ for some constraint parameterizations. We can robustify Prob. VII.2, where θ is considered as an uncertain variable in uncertainty set $\hat{\mathcal{F}}_\theta \subseteq \Theta$:

Problem VII.3 (Inverse constraint learning, robustified in θ)**.**

$$\begin{aligned} \sup_{\hat{\mathcal{F}}_\theta} \quad & \text{Vol}(\hat{\mathcal{F}}_\theta) \\ \text{s.t.} \quad & \forall \theta \in \hat{\mathcal{F}}_\theta, \quad \exists \{\boldsymbol{\lambda}_k^j, \boldsymbol{\lambda}_{\neg k}^j, \boldsymbol{\nu}_k^j \mid \text{KKT}(\xi_j^{\text{loc}})\}_{j=1}^{N_s} \end{aligned}$$

and search for the largest set $\hat{\mathcal{F}}_\theta \subseteq \Theta$ where each $\theta \in \hat{\mathcal{F}}_\theta$ satisfies KKT; the optimizer of Prob. VII.3 is $\mathcal{F}_\theta$. However, Prob. VII.3 is intractable due to 1) the optimization over arbitrarily-shaped sets $\hat{\mathcal{F}}_\theta$, 2) measuring the volume of such sets, and 3) the existential quantifiers $\exists$, implying we may need to find different Lagrange multipliers for each $\theta \in \hat{\mathcal{F}}_\theta$. We address these challenges in the following.

7.3.1 Obtaining the set of demonstration-consistent constraints $\mathcal{F}_\theta$

We assume the unknown constraint $\mathcal{A}(\theta)$ can be represented as a union of boxes in constraint space $\{\kappa \mid \bigcup_i [I, -I]^\top \kappa \leq \theta_i\}$. This assumption is reasonable as any shape can be represented by unioning enough boxes *Tao* (2016), though this can be inefficient (thus, we relax the assumption in App. B.3.1). In App. B.2.3.1, we prove that if $\mathcal{A}(\theta)$ can be described as a union of boxes, so can $\mathcal{F}_\theta$. By exploiting this structure, we develop a tractable variant of Prob. VII.3 using robust linear programming *Ben-Tal et al.* (2009).

We address the challenge of set optimization by optimizing over only boxes. Using the identity $\sup_{\|u\|_\infty \le 1} a^\top u = \|a\|_1$, a linear constraint $a^\top(x + s \odot u) \le b$ involving uncertain variable $u : \|u\|_\infty \le 1$, can be equivalently written without u as $a^\top x + \|a \odot s\|_1 \le b$, where $s \in \mathbb{R}^d_{\ge 0}$ scales the uncertainty. We can use this idea to enforce that the KKT conditions robustly hold everywhere in some box $\theta + s \odot u$, where $\|u\|_\infty \le 1$. Concretely, we can replace (7.3a) with $\mathbf{g}_{\neg k}(\xi_j^{\text{loc}}, \theta + s \odot u) \le \mathbf{0}$, (7.3c) with $\boldsymbol{\lambda}^j_{\neg k} \odot \mathbf{g}_{\neg k}(\xi_j^{\text{loc}}, \theta + s \odot u) = \mathbf{0}$, and (7.3d) with $\nabla_{\xi_{xu}} c(\xi_j^{\text{loc}}) + \boldsymbol{\lambda}_k^{j\top} \nabla_{\xi_{xu}} \mathbf{g}_k(\xi_j^{\text{loc}}) + \boldsymbol{\lambda}_{\neg k}^{j\top} \nabla_{\xi_{xu}} \mathbf{g}_{\neg k}(\xi_j^{\text{loc}}, \theta + s \odot u) + \boldsymbol{\nu}_k^{j\top} \nabla_{\xi_{xu}} \mathbf{h}_k(\xi_j^{\text{loc}}) = \mathbf{0}$, and eliminate u with the identity. We denote (7.3b) and the robustified (7.3a), (7.3c), and (7.3d) together as $\text{KKT}^{\text{box}}_{\text{rob}}(\xi_j^{\text{dem}})$, which are representable in a mixed integer linear program (MILP) (we need binary variables to enforce the robustified (7.3c)).

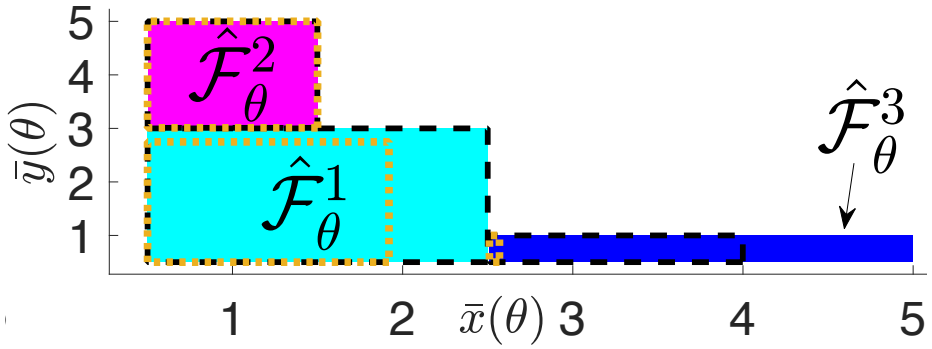


Figure 7.3: Extracting $\mathcal{F}_\theta$ via Alg. VII.1: requires 3 iterations. Overlaid: $\mathcal{B}_i^{\varepsilon_{\min}}$ (black dotted) and $\mathcal{B}_i^R$ (orange dotted), $i = 1, ..., 3$, optimized by solving Probs. VII.8-$\varepsilon_{\min}$ and VII.8-R (plans in Fig. 7.4).

The box representation of $\hat{\mathcal{F}}_\theta$ simplifies volume optimization. Since s scales the uncertainty (and thus the volume of $\hat{\mathcal{F}}_\theta$), we can satisfy $a^\top(x + s \odot u) \le b$ with "maximum robustness" by jointly finding x and s to maximize the volume of $\hat{\mathcal{F}}_\theta$, $\prod_i s_i$, where s_i is the ith entry of s. While $\prod_i s_i$ is non-convex in s, its geometric mean, $(\prod_i s_i)^{1/d}$, is conic-representable *Alizadeh and Goldfarb* (2003). It is also a monotonic transform of the volume, and thus an exact surrogate for volume maximization.

Finally, we can ignore the existential quantifiers in this case: (7.3d) does not involve θ (as ξ_{xu} does not multiply θ), (7.3c) implies that $\lambda^j_{\neg k,i} = 0$ for any coordinates i where $g_{\neg k,i}(\xi_j^{\text{dem}}, \theta)$ varies (hence one value, $\lambda = 0$, suffices), and (7.3a) does not involve $\mathcal{L}$. Thus, a single set of multipliers suffices, and we find the largest box-shaped $\hat{\mathcal{F}}_\theta$ via Prob. VII.4, a mixed integer second order cone program (MISOCP).

Problem VII.4 (Box robustification)**.**

$$\begin{array}{ll} \underset{s,\theta,\mathcal{L}}{\text{maximize}} & \left(\prod_i s_i\right)^{1/d} \\ \text{subject to} & \{\text{KKT}^{\text{box}}_{\text{rob}}(\xi_j^{\text{loc}})\}_{j=1}^{N_s} \end{array}$$

Solving Prob. VII.4 returns the largest box contained within $\mathcal{F}_\theta$: $\hat{\mathcal{F}}_\theta = \{\theta' \mid \bigwedge_{i=1}^d |\theta'_i - \theta_i| \le s_i\} \subseteq \mathcal{F}_\theta$. As $\mathcal{F}_\theta$ is a union of boxes, we can extract $\mathcal{F}_\theta$ in its entirety

by solving Prob. VII.4, removing the extracted $\hat{\mathcal{F}}_\theta$ from its feasible set (done with binary variables), and re-solving Prob. VII.4 with the modified feasible set until it becomes infeasible (Alg. VII.1, Fig. 7.3). Concretely, $\mathcal{F}_\theta = \bigcup_{i=1}^{N_{\text{infeas}}} \hat{\mathcal{F}}_\theta^i$, where $\hat{\mathcal{F}}_\theta^i$ is the box returned at the ith iteration and N_{infeas} is the iteration when infeasibility is reached. We can also prove some theoretical guarantees on Alg. VII.1 (see App. B.5 for proofs).

Theorem VII.5. *If Alg. VII.1 terminates for any parameterization, its output is guaranteed to cover $\mathcal{F}_\theta$.*

Theorem VII.6. *Alg. VII.1 is guaranteed to terminate in finite time for union-of-boxes parameterizations.*

Algorithm VII.1: Iterative $\mathcal{F}_\theta$ extraction

$i = 0$; **while** *Prob. VII.4 feasible* **do**
  $i \leftarrow i + 1$; $\hat{\mathcal{F}}_\theta^i \leftarrow$ Prob. VII.4$(\{\hat{\mathcal{F}}_\theta^j\}_{j=1}^{i-1})$;
  remove $\hat{\mathcal{F}}_\theta^i$ from Prob. VII.4's feasible set;
return $\bigcup_i \hat{\mathcal{F}}_\theta^i$

In closing, we refer to App. B.3.1, where we modify Alg. VII.1 to more efficiently extract $\mathcal{F}_\theta$ for other constraint parameterizations by covering $\mathcal{F}_\theta$ with zonotopes instead of boxes.

7.3.2 Obtaining the constraint belief $b(\theta)$

To perform a Bayesian update of $p(\theta)$, conditioning on the extracted $\mathcal{F}_\theta$, we assume that a demonstration is equally likely to have been generated in response to any θ for which it is locally-optimal:

$$p(\{\xi_j^{\text{dem}}\}_{j=1}^{N_{\text{dem}}} \mid \theta) \propto \begin{cases} 1 & \text{if } \{\text{KKT}(\xi_j^{\text{dem}}, \theta)\}_{j=1}^{N_{\text{dem}}} \text{ satisfied} \\ 0 & \text{else} \end{cases} \tag{7.8}$$

Then, a Bayesian update incorporating the demonstrations amounts to removing all probability mass from KKT-inconsistent θ and renormalizing the probabilities for the KKT-consistent θ:

$$b_{\text{dem}}(\theta) \doteq p(\theta \mid \{\xi_j^{\text{dem}}\}_{j=1}^{N_{\text{dem}}}) = \frac{p(\{\xi_j^{\text{dem}}\}_{j=1}^{N_{\text{dem}}} \mid \theta) p(\theta)}{\int_\Theta p(\{\xi_j^{\text{dem}}\}_{j=1}^{N_{\text{dem}}} \mid \theta) p(\theta) d\theta} = \begin{cases} \frac{p(\theta)}{\int_{\mathcal{F}_\theta} p(\theta) d\theta} & \text{if } \theta \in \mathcal{F}_\theta \\ 0 & \text{else} \end{cases} \tag{7.9}$$

Finally, we note that this approach is also compatible with uninformative priors (i.e., if no demonstrations are provided) by using the initial prior as the belief: $b(\theta) = p(\theta)$.

7.4 Policies for adaptive constraint satisfaction

We describe how to use the belief over infinite constraints $b_{\text{dem}}(\theta)$ to plan open-loop trajectories with exact safety probability guarantees (Sec. 7.4.1), how to plan with more complex constraints with samples from $b_{\text{dem}}(\theta)$ (Sec. 7.4.2), how $b_{\text{dem}}(\theta)$ can be updated to use constraint data sensed in execution (Sec. 7.4.3), and how the open-loop plans can be used in a closed-loop policy (Sec. 7.4.4).

7.4.1 Planning open-loop trajectories with an *infinite set* of possible constraints

Problem VII.7. *Chance-constrained plan*

$$\min_{\xi_{xu}} \quad c_\Pi(\xi_{xu}) \tag{7.10a}$$
$$\text{s.t.} \quad \bar{\phi}(\xi_{xu}) \in \bar{\mathcal{S}} \subseteq \bar{\mathcal{C}} \tag{7.10b}$$
$$\phi_\Pi(\xi_{xu}) \in \mathcal{S}_\Pi \subseteq \mathcal{C}_\Pi \tag{7.10c}$$
$$\Pr(\xi_{xu} \text{ safe}) \geq 1 - \varepsilon \tag{7.10d}$$

We wish to solve Prob. VII.7 for convex (7.10a), which seeks to complete a task while ensuring the plan is safe with probability at least $1-\varepsilon$ under the belief $b_{\text{dem}}(\theta)$, that is, $\Pr(\xi_{xu} \text{ safe}) = \int_{\Theta_s} b_{\text{dem}}(\theta) d\theta$, where $\Theta_s = \{\theta \mid \phi(\xi_{xu}) \in \mathcal{S}(\theta)\} \subseteq \mathcal{F}_\theta$ is the set of constraints that ξ_{xu} satisfies. Here, the safety threshold $\varepsilon \in [0, 1]$ may be predetermined, or if we wish to plan the safest possible trajectory, we can find the smallest ε for which Prob. VII.7 is feasible; denote this variant as Prob. VII.7-$\varepsilon_{\min}$. Intuitively, Prob. VII.7 seeks to solve a chance-constrained variant of Prob. VII.1 for a novel task II, where the uncertain constraints must be satisfied with a sufficiently high probability. Prob. VII.7 is challenging due to (7.10d), as evaluating this probability requires integrating high-dimensional parameters θ over a possibly arbitrarily-shaped Θ_s; hence, (7.10d) is intractable to enforce exactly for arbitrary distributions and Θ_s. We show that by sacrificing global optimality, it is tractable to enforce (7.10d) exactly for simple priors $p(\theta)$ by assuming a simple shape for Θ_s.

Problem VII.8. *Riemann-sum chance-constrained plan*

$$\min_{\xi_{xu}, \mathcal{B}_i, t_i} \quad c_\Pi(\xi_{xu}) \tag{7.11a}$$
$$\text{s.t.} \quad \bar{\phi}(\xi_{xu}) \in \bar{\mathcal{S}} \subseteq \bar{\mathcal{C}},\ \phi_\Pi(\xi_{xu}) \in \mathcal{S}_\Pi \subseteq \mathcal{C}_\Pi \tag{7.11b}$$
$$\xi_{xu} \in \mathcal{S}(\theta),\ \forall \theta \in \mathcal{B}_1, \ldots, \mathcal{B}_{N_{\text{box}}} \tag{7.11c}$$
$$\mathcal{B}_i \cap \mathcal{B}_j = \emptyset,\ i \neq j, \quad \mathcal{B}_i \subseteq \mathcal{F}_\theta, \forall i \tag{7.11d}$$
$$0 \leq t_i \leq (\textstyle\prod_i b_i^{\text{scale}})^{1/d},\ i = 1, ..., N_{\text{box}} \tag{7.11e}$$
$$\textstyle\sum_i t_i^d \geq (1-\varepsilon)\text{Vol}(\mathcal{F}_\theta) \tag{7.11f}$$

Our solution, Prob. VII.8, optimizes over subsets Θ_s that can be represented as a union of boxes $\Theta_s = \bigcup_{i=1}^{N_{\text{boxes}}} \mathcal{B}_i$, $\mathcal{B}_i \subseteq \mathcal{F}_\theta$, for all i (cf. Sec. 7.3.1). Each box is parameterized with a center $b_i^{\text{cen}} \in \mathbb{R}^d$ and scalings $b_i^{\text{scale}} \in \mathbb{R}_+^d$: $\mathcal{B}_i = \{b_i^{\text{cen}} + b_i^{\text{scale}} \odot u \mid$

$u \in [-1,1]^d\}$. (7.11c)-(7.11f) implement this box-limited chance constraint (see detailed explanations for each constraint in App. B.1). We restrict focus to priors $p(\theta)$ that can be integrated over boxes in closed form, and for which a monotonic transformation of the resulting integral is concave in b_i^{cen} and b_i^{scale}. While the concavity assumption is satisfied by the broad class of log-concave distributions *Bagnoli and Bergstrom* (2005), the closed-form integral is more restrictive. In this chapter, we focus only on a uniform $p(\theta)$ (see App. B.4.3 for extensions to other distributions); note that this does *not* imply a uniform probability of safety over the constraint space $\mathcal{C}$. Intuitively, Prob. VII.8 performs a box-limited Riemann sum integration over the constraint belief. Each box $\mathcal{B}_i$ represents a subset of $\mathcal{F}_\theta$ over which the probability is integrated (cf. Fig. 7.3). For piecewise affine (PWA) dynamics, Prob. VII.8 can be written as an MISOCP, except for (7.11f) which renders Prob. VII.8 an MIBLP: solvable with *Gurobi Optimization* (2020), but possibly slow. We can replace (7.11f) with a linear surrogate $\sum_i t_i \geq (1-\varepsilon)\text{Vol}(\mathcal{F}_\theta)$, but this can still be slow if N_{box} is large. We discuss efficient reformulations of Prob. VII.8 in App. B.4. Overall, we have this result (proof in App. B.5):

Theorem VII.9. *A solution to Prob. VII.8 is a guaranteed feasible, possibly suboptimal solution to Prob. VII.7.*

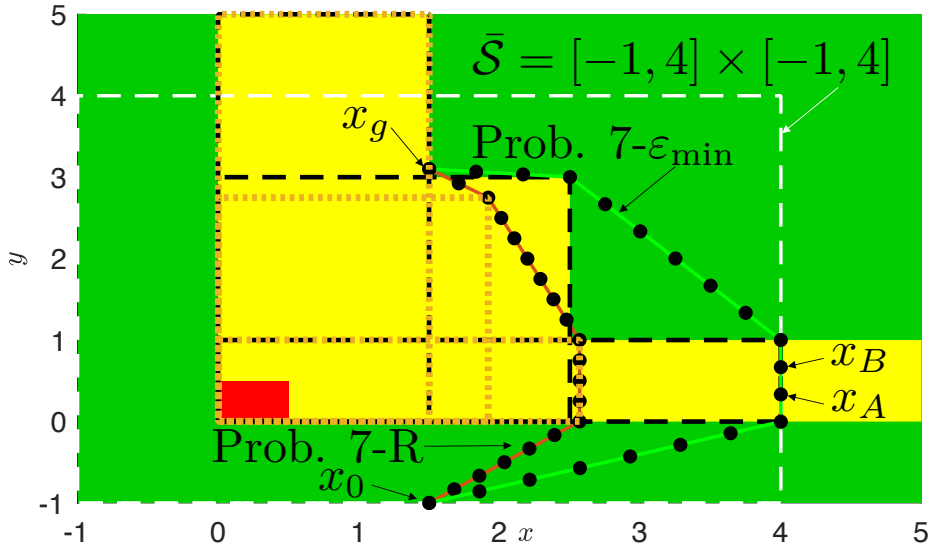


Figure 7.4: Generated plans for $\mathcal{B}_i$ in Fig. 7.3. $\text{proj}_{\mathcal{X}}(\mathcal{B}_i)$ are overlaid (with matched color).

Instead of the chance-constrained formulation, we can directly trade off the cost and the safety probability in the objective; i.e., in Probs. VII.7; VII.8, change (7.10a); (7.11a) to the ratio $c_\Pi(\xi_{xu})/\Pr(\xi_{xu} \text{ safe})$; $c_\Pi(\xi_{xu})/\sum_i t_i^d$ and remove (7.10d); (7.11f). Denote these variants Prob. VII.7-R; Prob. VII.8-R. Note that after linearizing $\sum_i t_i^d$, the objective is quasi-convex, so we can rewrite Prob. VII.8-R as a feasibility problem with a new optimization constraint $c_\Pi(\xi_{xu}) - \alpha \Pr(\xi_{xu} \text{ safe}) \leq 0$, and do a line search on α, solving Prob. VII.8-R as a sequence of MISOCPs.

7.4.2 Planning open-loop trajectories with a *finite set* of sampled possible constraints

For complex constraints arising from nonlinear dynamics or constraint parameterizations, Prob. VII.8 is hard to solve as it involves both integer variables and nonlinearities. While sometimes we can plan for a PWA model that roughly captures the nonlinear dynamics (i.e., modeling a quadrotor as a double integrator), we generally use sampled approximations of Prob. VII.8 when it cannot be written as a mixed integer convex program (MICP), in particular, Minimum Constraint Removal (MCR) *Hauser* (2014) and the Blindfolded Traveler's Problem (BTP) *Saund et al.* (2019), briefly described here (cf. App. B.4 for details):

1. MCR takes a *finite* set of constraints and incrementally constructs a roadmap to connect a start and goal state while violating the minimum number of constraints. MCR can be used to approximate Prob. VII.8-$\varepsilon_{\min}$ by sampling a finite set of constraints $\{\theta_i \sim b(\theta)\}_{i=1}^{N_{\text{sam}}}$ as input to MCR.

2. BTP is a roadmap-based planner for additive cost functions $c_\Pi(\xi_{xu})$ which takes as input a state space graph (V, E), where executing edge $e \in E$ costs $c(e)$ with probability $p(e)$ of being safe. To use BTP, we sample constraints $\{\theta_i \sim b(\theta)\}_{i=1}^{N_{\text{sam}}}$, and use them to approximate $p(e)$; we plan on the graph by running A* with modified edge costs $\sum_e c_\Pi(e) - \beta p(e \text{ safe})$, for some weight β.

We note that while the sampled approximations can be more flexible than Prob. VII.8, they are not guaranteed to return a feasible solution to Prob. VII.7, as it depends on the constraints that are sampled.

7.4.3 Updates to $b(\theta)$ in online execution

Our framework can also incorporate uncertain information about the true constraint sensed in execution by computing a belief update. Suppose that we are given $\mathfrak{C}_s \doteq \{\mathfrak{C}_s^i\}_{i=1}^{N_s}$ and $\mathfrak{C}_{\neg s} \doteq \{\mathfrak{C}_{\neg s}^i\}_{i=1}^{N_{\neg s}}$ as a set of *sets of possibly safe/unsafe states*, respectively, where each $\mathfrak{C}_s^i$ / $\mathfrak{C}_{\neg s}^i$ denotes a finite set where at least one state is safe/unsafe. Let the set of all constraints consistent with $\mathfrak{C}_s$, $\mathfrak{C}_{\neg s}$ be $\mathcal{F}_\theta^{s,\neg s} \doteq \{\theta \in \mathcal{F}_\theta \mid \bigwedge_{i=1}^{N_s}(\exists \kappa \in \mathfrak{C}_s^i, g(\kappa, \theta) \leq 0) \wedge \bigwedge_{i=1}^{N_{\neg s}}(\exists \kappa \in \mathfrak{C}_{\neg s}^i, g(\kappa, \theta) > 0)\}$. We compute $\mathcal{F}_\theta^{s,\neg s}$ iteratively with a variant of Alg. VII.1 (see App. B.4.4 for details). Finally, we perform the update:

$$b_{\text{ex}}(\theta) \doteq p(\theta \mid \{\xi_j^{\text{dem}}\}_{j=1}^{N_{\text{dem}}}, \mathcal{C}_s, \mathcal{C}_{\neg s}) = \begin{cases} \frac{p(\theta)}{\int_{\mathcal{F}_\theta^{s,\neg s}} p(\theta) d\theta} & \text{if } \theta \in \mathcal{F}_\theta^{s,\neg s} \\ 0 & \text{else} \end{cases} \tag{7.12}$$

This setup can handle data from many different constraint sensing modalities, like direct, exact sensing (from a bump sensor), long-range measurements (from bounded-range LiDAR), or uncertain contact measurements *Saund et al.* (2019) where collision cannot be exactly localized on the robot volume (see App. B.4.4 for more details).

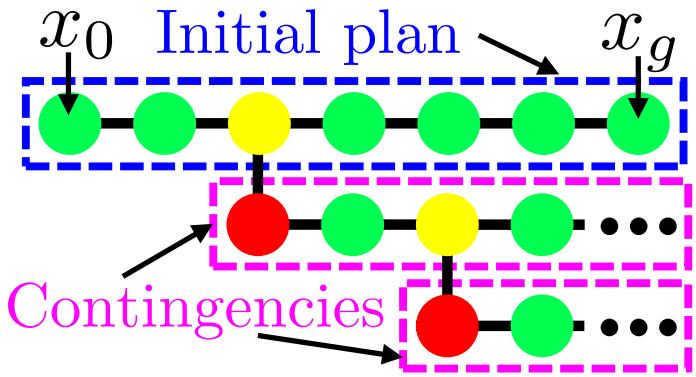


Figure 7.5: Policy tree: initial plan and contingencies, rooted at possibly unsafe states. Green / red / yellow states in $\mathcal{G}_s$ / $\mathcal{G}_{\neg s}$ / $\text{proj}_{\mathcal{C}}(\mathcal{F}_\theta)$.

7.4.4 Closed-loop policies for adaptive constraint satisfaction

Finally, we compute closed-loop policies for Probs. MCV and MEC. Our strategy is simple: for Probs. MCV and Prob. MEC, solve Prob. VII.8-$\varepsilon_{\min}$ and Prob. VII.8-R, respectively, in a receding horizon fashion, i.e at each time-step, we update $b(\theta)$ with the new constraint information, re-solve the optimization, and switch to the new solution if the previous plan is suboptimal/unsafe. This policy takes $p(\theta)$ and a sequence of observations $\mathcal{O} = (\mathfrak{C}_s, \mathfrak{C}_{\neg s})$ to estimate a belief, and uses the belief and current state to output u. These strategies are motivated by results in sequential decision making that provide approximation guarantees for greedy policies proposing solutions that minimize the ratio of cost to safety probability at each iteration *Dor et al.* (1998); Probs. VII.8-$\varepsilon_{\min}$, VII.8-R both do this.

Note that as our policy executes a plan until sensing implies it is suboptimal or unsafe, upon which it switches, it can be represented as a tree, where contingency plans are rooted only at states on the current plan where switches can occur. See Fig. 7.5 for the simple case where the policy only switches upon learning the current plan is unsafe; here, the branching is sparse, occurring only at possibly unsafe points (yellow) on the initial and replanned trajectories. In these cases, we can exploit the sparsity to avoid solving Prob. VII.8 at runtime by precomputing the contingency plans, facilitating real-time policy execution. For tractability, the precomputation assumes no unmodeled obstacles appear at runtime (this eliminates the sparse branching, as it makes each state on the plan possibly unsafe), discretizes the set of possible continuous sensing measurements (if not, we could need to compute contingencies for an infinite set of possible measurements), and terminates branching at a finite tree depth (as planning may be infeasible for a worst-case constraint). If the assumptions do not hold (as in some of our results), we can always compute new plans online, though it can be slow.

As an example, consider computing contingencies for the green plan in Fig. 7.4. Only x_A and x_B lie in the possibly unsafe (yellow) region, so if no unmodeled obstacles appear at runtime, we can only be forced to replan in two cases: 1) x_A is unsafe, 2) x_A is safe and x_B is unsafe. In case 1, we update the belief, keeping only constraints marking x_A as unsafe, and plan a contingency satisfying as many constraints as

possible from the new belief. This repeats recursively, up to a finite recursion depth, for any possibly unsafe states on the contingency. In case 2, updating the belief to mark x_A as safe and x_B as unsafe renders the belief empty, since this is impossible given the initial belief and box parameterization. We can thus avoid computing further contingencies on this policy tree branch.

Problem / Constraints/prior	Prob. MCV	Prob. MEC
MICP-representable	Prob. VII.8-$\varepsilon_{\min}$	Prob. VII.8-R
Not MICP-rep.	MCR	BTP

Table 7.1: Which open-loop planner to use?

To recap, we would ideally solve Prob. VII.7 to get open-loop plans for our policy, but it is intractable. If all constraints (dynamics, uncertain constraints, etc.) and the integrals of $p(\theta)$ are MICP-representable, we can approximate Prob. VII.7 with Prob. VII.8, which enjoys theoretical guarantees by using the *infinite* belief. If not, we use MCR/BTP, which use *finite samples* from the belief. This is summarized in Table 7.1.

7.5 Experiments

We show our method scales to safely and efficiently plan for high-dimensional (12D) systems with combined state/control constraint uncertainty, constraint sensing uncertainty, and high-dimensional (30D) constraints. See App. B.6 for more details and experiments (7-DOF arm planning with suboptimal demonstrations, and nonlinear constraint planning using zonotope-based $\mathcal{F}_\theta$ extraction), App. B.6.6 for computation time discussion, and `https://youtu.be/aWZ_U-gWQJI` for visualizations.

Mixed quadrotor uncertainty: We plan for a quadrotor (dynamics in App. B.6) carrying a payload of uncertain weight around uncertain obstacles. We are given one demonstration (Fig. 7.6.B, pink) to learn a 7D constraint $\theta \in \mathbb{R}^7$ (6 for obstacle, 1 for control). After extracting $\mathcal{F}_\theta$ with Alg. VII.1, (Fig. 7.6.B), we remain uncertain about the obstacle's y-extents. We model the uncertain weight as an unknown control constraint $\|u\|_2^2 \leq \bar{U}(\theta)$; from the demonstration, we learn that $\|u\|_2^2 \leq 97.85$ is guaranteed safe (Fig. 7.6.A), and KKT also tells us the constraint is inactive, so $\|u\|_2^2 \in (97.85, 100] = \text{proj}_{\mathcal{U}}(\mathcal{F}_\theta)$ is possibly unsafe. The quadrotor has bump and torque sensors to directly detect state and control constraint violations, respectively. We now solve Prob. MCV starting from a lower initial state (Fig. 7.6.C), which we do by solving Prob. VII.8-$\varepsilon_{\min}$ for an initial plan and contingencies, directly optimizing over the infinite constraint belief. We use a double-integrator approximation of the quadrotor dynamics and restrict each open-loop trajectory to 30 timesteps; thus, it is not possible to satisfy all constraints in $\mathcal{F}_\theta$ while reaching the goal in the time limit. Solving Prob. VII.8-$\varepsilon_{\min}$ returns Plan 1 (Fig. 7.6.C), which violates some possible control constraints in order to lift the quadrotor over all possible obstacles. Our policy also generates contingencies (Fig. 7.6.D-F) in case Plan 1 violates the true control constraint, and we must plan to avoid the possible obstacles. To emphasize the benefit of *optimizing* over the infinite set of possible constraints, we

compare to two baselines: a variant of the scenario approach *Grammatico et al.* (2016), where we iteratively *sample* and enforce $\{\theta_i \sim b(\theta)\}_{i=1}^{N_{\text{sam}}}$ until the planning problem becomes infeasible, and an optimistic planner, which only avoids $\mathcal{G}_{\neg s}$ and replans upon violating a constraint (see App. B.6.1 for baseline details). We evaluate the number of violations on constraints uniformly drawn from $\mathcal{F}_\theta$. Our policy suffers 0.54 ± 0.94 constraint violations (average $\pm$ standard deviation), the scenario approach 1.30 ± 1.36 violations, and the optimistic strategy 9.10 ± 4.65 violations. We outperform the scenario approach, as we *optimize* over the set of constraints to satisfy, and the optimistic strategy, as it ignores constraint uncertainty. Running Alg. VII.1 and Prob. VII.8 takes 3.3 and 1.1 sec., respectively. See App. B.6.3 for a constraint violation histogram empirically validating our probabilistic safety guarantees.

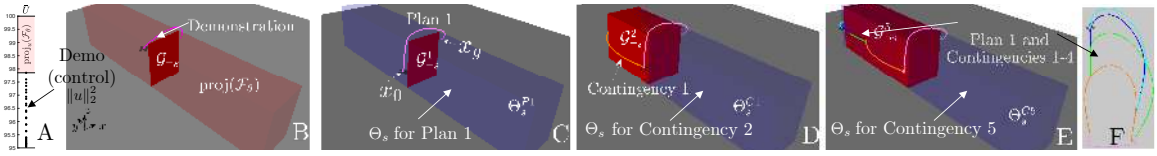


Figure 7.6: Mixed quadrotor uncertainty example. **A-B**. Initial control and state constraint uncertainty. **C**. Initial plan for a new task. **D-F**. Contingencies are pre-computed, and the system switches if the initial plan is unsafe.

7-DOF arm with contact sensing uncertainty: We plan for a 7-DOF arm (dynamics in App. B.6) near a storage rack. We are given two demonstrations (Fig. 7.7.A), and after running Alg. VII.1 to obtain $b_{\text{dem}}(\theta)$, partly reveal the shelf constraint (Fig. 7.7.B), which has parameters $\theta \in \mathbb{R}^6$. We assume an uncertain contact sensing model *Saund and Berenson* (2018), where contact is assumed to be at any point on the arm downstream on the kinematic chain where a torque limit is exceeded (cf. App. B.4). We solve Prob. MEC for $c_\Pi(\xi_{xu}) = \sum_{t=1}^{T-1} \|x_{t+1} - x_t\|_2$, for the task of moving the arm from below to above the shelf (Fig. 7.7.B), using 100 (uniform) samples from $b_{\text{dem}}(\theta)$ in BTP. We compare our policy to 1) BTP without demonstrations and the union-of-boxes parameterization, and 2) an optimistic approach that executes the optimal path on the BTP graph after removing unsafe edges, and replanning if a constraint is violated. From the demonstrations, we can determine that a subset of the shelf is guaranteed unsafe (Fig. 7.7, dark red); beyond that, we are uncertain (Fig. 7.7.B). Thus, we plan to avoid that region, swinging the arm around and over the uncertain area. However, in doing so, the arm bumps into an unmodeled obstacle: a box on the lower shelf (Fig. 7.7.C). The contact sensor informs our method that some point on the end effector is in collision; we add 300 sampled points on the end effector to $\mathfrak{C}_{\neg s}$ and samples from the traversed free space to $\mathfrak{C}_s$. Our method then automatically determines that it needs to update the constraint parameterization, as geometrically there is no single box that can explain the demonstrations, $\mathfrak{C}_s$, and $\mathfrak{C}_{\neg s}$. After updating θ to include two boxes (now $\theta \in \mathbb{R}^{12}$), we extract $\mathcal{F}_\theta$ for this updated parameterization, and resample 100 (uniform) samples from the updated $b_{\text{ex}}(\theta)$ as input to BTP (see Fig. 7.7.D). As our belief now indicates an uncertain region near the lower shelf obstacle, our policy plans to move further out from the shelf to avoid the uncertain region, and reaches

the goal. Running Alg. VII.1 and BTP takes 20 and between 5-20 min. (can be sped up by precomputing arm swept volumes, see *Saund et al.* (2019), App. B.6.6), respectively. Overall, our policy reaches the goal with a cost of 8.19 rad, while the cost without demonstrations/boxes is 18.24 rad, and the optimistic approach is 143.31 rad. Without demonstrations/boxes, we need several more iterations bumping into the shelf before it is sufficiently localized, and the optimistic approach ignores spatial correlation in edge validity, exploring far more edges (cf. App. B.6 for details). This example suggests our method scales to high-dimensional systems, can detect when the constraint representation is insufficient, and can use complex constraint measurements.

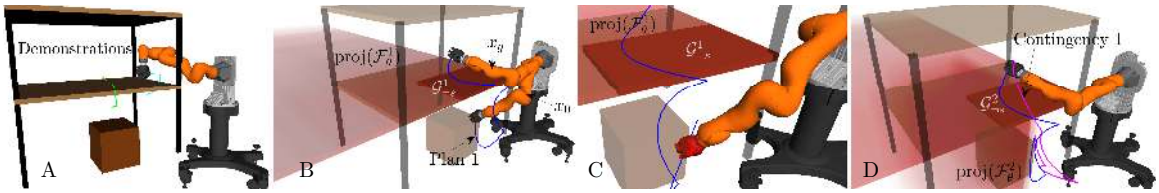


Figure 7.7: Arm with contact sensing uncertainty. **A**: Demonstrations. **B**. Initial constraint uncertainty (red) and plan (blue). **C**. The initial plan violates an unmodeled constraint, triggering a belief update. **D**. Replan online.

Quadrotor maze: We plan for a quadrotor in clutter with two-meter radius LiDAR sensing (Fig. 7.8). We know the brown obstacles *a priori*, and are given five demonstrations that reveal five obstacles ($\theta \in \mathbb{R}^{30}$) (Fig. 7.8.A), but provide little information about their size. We solve Prob. MEC steering from the bottom to the top of the maze (Fig. 7.8.D) while minimizing $c(\xi_{xu}) = \sum_{t=1}^{T-1} \|u_t\|_2^2$ by solving Prob. VII.8-R, optimizing directly over the continuous $b(\theta)$, and computing contingencies. We also modify Prob. VII.8-R to never collide in execution by avoiding the set of inevitable collision states *Fraichard and Asama* (2004) under the double-integrator model; this is modeled with additional constraints (see App. B.4). We obtain dynamically-feasible quadrotor trajectories by warmstarting the nonlinear optimization with the double-integrator trajectory. We visualize our policy in Fig. 7.8. Running Alg. VII.1 and Prob. VII.8 takes 1 sec. and 1 min., respectively. Plan 1 (pink) intelligently trades off risk and performance. Note there may exist a direct path to the goal between the brown obstacles; however, the orange demonstration induces an obstacle that likely blocks this path. Also, moving left is riskier than moving right, as the uncertain obstacle on the left may create a dead end. Plan 1 avoids both traps, moving to the right and increasing altitude to avoid all possible obstacles induced by the dark blue demonstration. This enables maintenance of higher speed and thus lower cost, instead of cautiously approaching the uncertain region to determine if it is safe to cut through. Finally, Plan 1 cuts through the possibly-unsafe region induced by the green demonstration, as the obstacle is unlikely to extend down to the brown obstacle. We discretize the possible constraint measurements in this region on a grid, planning contingencies if the passage is partially (Contingencies 1-3) or completely blocked (4). We compare to two approaches, *Chou et al.* (2020b), which plans trajectories which are guaranteed-safe under the constraint parameterization,

and *Janson et al.* (2018), which plans optimistically over sets of subgoals on the frontier (see App. B.6.1 for more details). Drawing constraints uniformly from $\mathcal{F}_\theta$, our policy solves Prob. MCE with a cost of 1.28 ± 0.27, while *Chou et al.* (2020b) is conservative, with a cost of 6.29, and *Janson et al.* (2018) returns a cost of 5.51 ± 1.65, as it explores the likely dead end between the brown obstacles. This example suggests our method scales to high-dimensional systems and constraint spaces and can compute a policy integrating sensor inputs to adaptively switch between plans and contingencies.

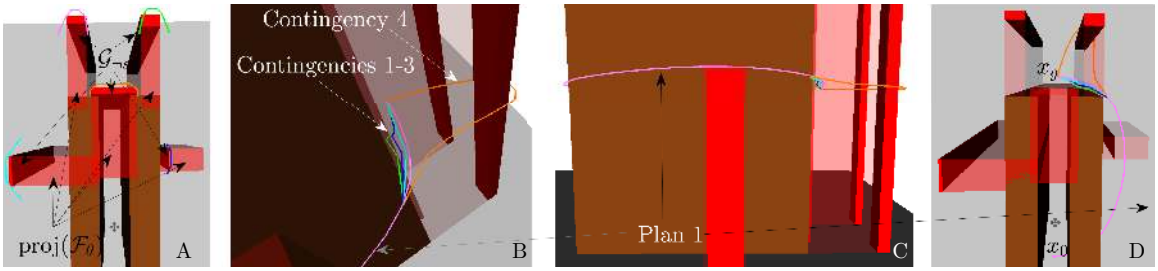


Figure 7.8: Quadrotor maze. **A**. Demos., initial constraint uncertainty. **B-D**. Three views of the initial plan (pink) and contingencies for different sensing possibilities, obtained by gridding the possible sensor measurements.

7.6 Conclusion

We present a method to address uncertainty in constraints learned from demonstrations. Instead of trying to satisfy all possible constraints, we obtain a belief over constraints, then design open-loop planners which use the belief to "be as safe as possible while remaining efficient". We use these planners in a closed-loop policy that uses constraint data gathered online to help complete the task. In future work, we aim to speed up our method with parallel extraction and fast integer programming *Bertsimas and Stellato* (2019), and extend our method to adaptively plan with beliefs over temporal logic formulas (Chapter VI).

CHAPTER VIII

Safe Planning and Execution with Learned Dynamics via Data-Driven Model Error Bounds

In this chapter, we present a method for feedback motion planning of systems with unknown dynamics which provides probabilistic guarantees on safety, reachability, and goal stability. To find a domain in which a learned control-affine approximation of the true dynamics can be trusted, we estimate the Lipschitz constant of the difference between the true and learned dynamics, and ensure the estimate is valid with a given probability. Provided the system has at least as many controls as states, we also derive existence conditions for a one-step feedback law which can keep the real system within a small bound of a nominal trajectory planned with the learned dynamics. Our method imposes the feedback law existence as a constraint in a sampling-based planner, which returns a feedback policy around a nominal plan ensuring that, if the Lipschitz constant estimate is valid, the true system is safe during plan execution, reaches the goal, and is ultimately invariant in a small set about the goal. We demonstrate our approach by planning using learned models of a 6D quadrotor and a 7DOF Kuka arm. We show that a baseline which plans using the same learned dynamics *without* considering the error bound or the existence of the feedback law can fail to stabilize around the plan and become unsafe. This chapter is based on the paper *Knuth et al.* (2021a).

8.1 Introduction

Planning and control with guarantees on safety and reachability for systems with unknown dynamics has long been sought-after in the robotics and control community. Model-based optimal control can achieve this if the dynamics are precisely modeled, but modeling assumptions inevitably break down when applied to real physical systems due to unmodeled effects from friction, slip, flexing, etc. To account for this gap, data-driven machine learning methods and robust control seek to sidestep the need to precisely model the dynamics *a priori*. While robust control can provide strong guarantees when the unmodeled component of the dynamics is small and satisfies strong structural assumptions *Zhou and Doyle* (1998), such methods requires an accurate prior which may not be readily available. In contrast, machine learning methods are

flexible but often lack formal guarantees, precluding their use in safety-critical applications. For instance, small perturbations from training data cause drastically poor and costly predictions in stock prices and power consumption *Mode and Hoque* (2020). Since even small perturbations from the training distribution can yield untrustworthy results, applying AI systems to predict dynamics can lead to unsafe, unpredictable behavior.

To address this gap, we propose a method for planning with learned dynamics which yields probabilistic guarantees on safety, reachability, and goal invariance in execution on the true system. Our core insight is that we can determine where a learned model can be trusted for planning using the Lipschitz constant of the error (the difference between the true and learned dynamics), which also informs how well the training data covers the task-relevant domain. Under the assumption of deterministic true dynamics, we can plan trajectories in this trusted domain with strong safety guarantees for an important class of learned dynamical systems.

Specifically, with a Lipschitz constant, we can bound the difference in dynamics between a novel point (that our model was not trained on) and a training point. Since the bound grows with the distance to training points, we can naturally define a domain where the model can be trusted as the set of points within a certain distance to training points. Conversely, to obtain a small bound over a desired domain, it is necessary to have good training data coverage in the task-relevant domain. At a high level, to obtain a small bound on the error in a domain, we want to have good coverage over the domain and regularity of the learned model via the Lipschitz constant of the error.

Our safety and reachability guarantees ultimately rely on an overestimate of the smallest Lipschitz constant. To find an estimate that exceeds the smallest Lipschitz constant with a given probability ρ, we use a statistical approach based on Extreme Value Theory *De Haan and Ferreira* (2007) and validate its result with a Kolmogorov-Smirnov goodness-of-fit test *DeGroot and Schervish* (2013). If the test validates our estimate, we can choose a confidence interval *DeGroot and Schervish* (2013) with an upper bound that overestimates the true Lipschitz constant with probability ρ. Our method requires the estimation of three Lipschitz constants, translating to system safety and reachability guarantees which hold with a probability of at least ρ^3. This guarantee is fairly strong as it holds for all time, unlike many methods offering probabilistic guarantees on a per trajectory or episode basis *Akametalu et al.* (2014) *Berkenkamp et al.* (2017) *van den Berg et al.* (2011).

If the learned dynamics have at least as many controls as states and are control-affine (note we do not assume the true dynamics are also control-affine), then we also determine conditions for the existence of a feedback controller that tightly tracks the planned trajectory in execution under the true dynamics. The tight tracking error bound yields favorable properties for our planner and controller: if we have a valid Lipschitz constant estimate for a sufficiently-accurate learned model, 1) we guarantee safety if no obstacle is within the tracking error of the trajectory, 2) we guarantee we can reach the goal within a small tolerance, and 3) if we can assert a feedback law that keeps the system at the goal exists, then the closed-loop system is guaranteed to remain in a small region around the goal. In this chapter, we assume the learned

dynamics are control-affine, deterministic, and have at least as many controls as states (such as a robotic arm under velocity control), the true dynamics are deterministic, and that independent samples of the true dynamics can be taken in the domain of interest. Our contributions are:

1. A method to bound error between two general dynamics functions in a domain by using a Lipschitz constant
2. A condition for uncertain control-affine systems that guarantees the existence of a feasible feedback law
3. A planner that probabilistically guarantees safety and closed-loop stability-like properties about the goal for learned dynamics with as many controls as states
4. Evaluation on a 7DOF Kuka arm and a 6D quadrotor

8.2 Preliminaries

Let $f : \mathcal{X} \times \mathcal{U} \to \mathcal{X}$ be the true unknown discrete-time dynamics where $\mathcal{X}$ is the state space and $\mathcal{U}$ is the control space, which we assume are deterministic. We define $g : \mathcal{X} \times \mathcal{U} \to \mathcal{X}$ to be an approximation of the true dynamics that is control-affine and therefore can be written as follows

$$g(x, u) = g_0(x) + g_1(x)u. \tag{8.1}$$

In this chapter, we represent the approximate dynamics with a neural network, though our method is agnostic to the structure of the model and how it is derived. Let $\mathcal{S} = \{(x_i, u_i, f(x_i, u_i))\}_{i=1}^N$ be the training data for g, and let $\Psi = \{(x_j, u_j, f(x_j, u_j))\}_{j=1}^M$ be another set of samples collected near $\mathcal{S}$ that will be used to estimate the Lipschitz constant. We use $\bar{\cdot}$ to refer to data points from $\mathcal{S}$ or Ψ. We place no assumption on how $\mathcal{S}$ is obtained; any appropriate method (uniform sampling, perturbations from expert trajectories, etc.) may be employed, although we require independent and identically distributed (i.i.d.) samples for Ψ. A single state-control pair is written as (x, u). With some abuse of notation, we write $(\bar{x}, \bar{u}) \in \mathcal{S}$ if $(\bar{x}, \bar{u}) = (x_i, u_i)$ for some $1 \leq i \leq N$ (similarly for Ψ).

A Lipschitz constant bounds how much outputs change with respect to a change in the inputs. For some function h, a Lipschitz constant over a domain $\mathcal{Z}$ is any number L such that for all $z_1, z_2 \in \mathcal{Z}$

$$\|h(z_1) - h(z_2)\| \leq L\|z_1 - z_2\| \tag{8.2}$$

Norms $\|\cdot\|$ are always the 2-norm or induced 2-norm. We define L_{f-g}, L_{g_0}, and L_{g_1} as the smallest Lipschitz constants of the error $f - g$, g_0, and g_1. The input to $f - g$ is a state-control pair (x, u) and its output is a state. For g_0, both the input and output are a state. For g_1, its input is a state and its output is a $\texttt{dim}(\mathcal{X}) \times \texttt{dim}(\mathcal{U})$ matrix where $\texttt{dim}(\cdot)$ is the dimension of the space. A ball $\mathcal{B}_r(x)$ of radius r about a point x is defined as the set $\{y \mid \|y - x\| < r\}$, also referred to as a r-ball about

x. We suppose the state space $\mathcal{X}$ is partitioned into safe $\mathcal{X}_{\text{safe}}$ and unsafe $\mathcal{X}_{\text{unsafe}}$ sets (e.g., the states in collision with an obstacle).

The method consists of two major components. First, we determine a trusted domain $D \subseteq \mathcal{X} \times \mathcal{U}$ and estimate the Lipschitz constants. Second, we use D to find a path to the goal satisfying our safety and reachability requirements.

Problem VIII.1. *Given a learned model g, unknown dynamics f, and datasets Ψ and $\mathcal{S}$, determine the trusted domain D where $\|f(x,u)-g(x,u)\| \leq \epsilon$, for some $\epsilon > 0$. Additionally determine the Lipschitz constants L_{f-g}, L_{g_0}, and L_{g_1} in D.*

Problem VIII.2. *Given control-affine g, unknown f, start x_I, goal x_G, goal tolerance λ, D, L_{f-g}, L_{g_0}, L_{g_1}, and $\mathcal{X}_{unsafe}$, plan a trajectory $(x_0, \ldots, x_K)$, $(u_0, \ldots, u_{K-1})$ such that $x_0 = x_I$, $x_{k+1} = g(x_k, u_k)$, $K < \infty$, and $\|x_K - x_G\| \leq \lambda$. Additionally, under the true dynamics f, guarantee that closed loop execution does not enter $\mathcal{X}_{unsafe}$, converges to $\mathcal{B}_{\epsilon+\lambda}(x_G)$, and remains in $\mathcal{B}_{\epsilon+\lambda}(x_G)$ after reaching x_K.*

8.3 Method

Secs. 8.3.1 - 8.3.2 and 8.3.3 - 8.3.4 cover our approaches to Probs. VIII.1 and VIII.2, respectively. In Sec. 8.3.1, we show how L_{f-g} can establish a trusted domain and how L_{f-g} can be estimated in Sec. 8.3.2. In Sec. 8.3.3, we design a planner that ensures safety, that the system remains in the trusted domain, and that a feedback law maintaining minimal tracking error exists. We present the full algorithm in Sec. 8.3.4.

8.3.1 The trusted domain

For many systems, we are only interested in a task-relevant domain, and it is often impossible to collect data everywhere in state space, especially for high-dimensional systems. Hence, it is natural that our learned model is only accurate near training data. With a Lipschitz constant of the error, we can precisely define how accurate the learned dynamics are in a domain constructed from the training data. We note this derivation can also be done for systems without the control-affine assumption on the learned dynamics, and thus it can still be useful for determining where a broader class of learned models can be trusted. However, removing the control-affine structure makes controller synthesis much more difficult, and is the subject of future work.

Consider a single training point $(\bar{x}, \bar{u})$ and a novel point (x, u). We derive a bound on the error between the true and estimated dynamics at (x, u) using the triangle inequality and Lipschitz constant of the error:

$$\begin{aligned} &\|f(x,u) - g(x,u)\| \\ &\quad = \|f(x,u) - g(x,u) - f(\bar{x},\bar{u}) + g(\bar{x},\bar{u}) \\ &\qquad + f(\bar{x},\bar{u}) - g(\bar{x},\bar{u})\| \\ &\quad \leq L_{f-g}\|(x,u) - (\bar{x},\bar{u})\| + \|f(\bar{x},\bar{u}) - g(\bar{x},\bar{u})\|. \end{aligned} \tag{8.3}$$

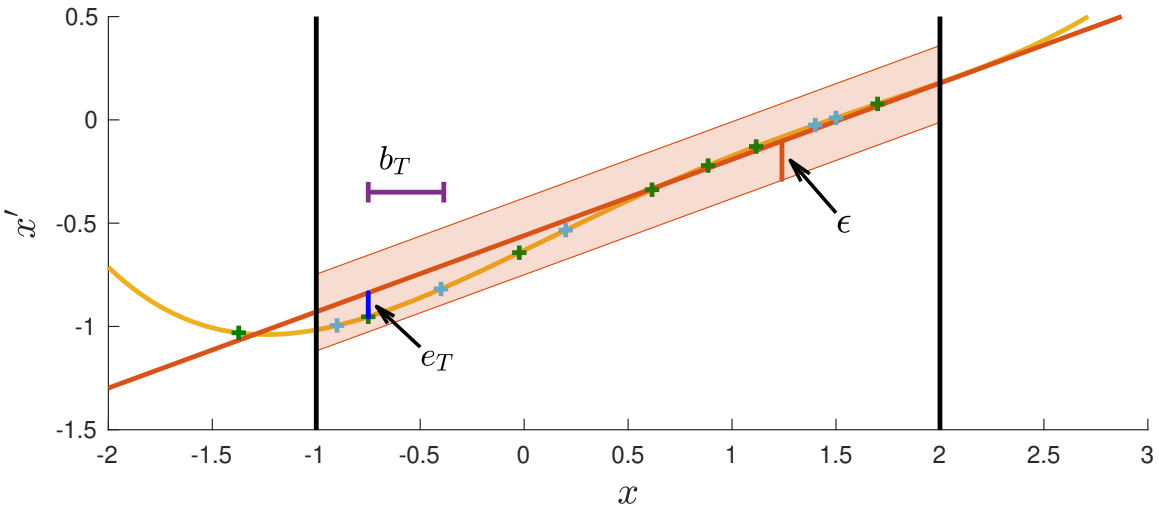


Figure 8.1: An example with $f(x) = x'$, $\texttt{dim}(\mathcal{X}) = 1$, and $\texttt{dim}(\mathcal{U}) = 0$. True dynamics: yellow; learned linear dynamics: orange; $\mathcal{S}$: green crosses; Ψ: light blue crosses; domain D: interval $[-1, 2]$, bordered in black. Here, $b_T = 0.3633$ (purple) and $e_T = 0.1161$ (blue). The Lipschitz constant of the error is $L_{f-g} = 0.1919$, yielding $\epsilon = 0.1859$. We can use this bound to ensure the difference between the learned and true dynamics is no more than ϵ in D (shaded orange area). Note L_{f-g} can be larger outside of D.

The above relation describes the error at a novel point, but we can also generalize to any domain D. Define b_T to be the dispersion *LaValle* (2006) of $\mathcal{S} \cap D$ in D and define e_T to be the maximum training error of the learned model. Explicitly,

$$b_T \doteq \sup_{(x,u)\in D} \min_{(\bar{x},\bar{u})\in\mathcal{S}\cap D} \|(x,u) - (\bar{x},\bar{u})\| \tag{8.4}$$

$$e_T \doteq \max_{(\bar{x},\bar{u})\in\mathcal{S}\cap D} \|f(\bar{x},\bar{u}) - g(\bar{x},\bar{u})\| \tag{8.5}$$

Then, we can uniformly bound the error across the entire set D to yield a simple and exact relation between f and g.

$$\epsilon \doteq L_{f-g} b_T + e_T \tag{8.6}$$

$$\forall (x,u) \in D \quad f(x,u) = g(x,u) + \delta, \quad \|\delta\| \leq \epsilon \tag{8.7}$$

See Fig. 8.1 for an example of these quantities. For the remainder of the method, we select D to be the union of r-balls about a subset of the training data $\mathcal{S}_D \subset \mathcal{S}$:

$$D = \bigcup_{(\bar{x},\bar{u})\in\mathcal{S}_D} \mathcal{B}_r(\bar{x},\bar{u}) \tag{8.8}$$

In the next section we discuss selection of $\mathcal{S}_D$, its role in estimating L_{f-g}, and how r is selected.

8.3.2 Estimating the Lipschitz constant

For (8.7) to hold over all of D, we require that L_{f-g} is a Lipschitz constant for the error. We use results from Extreme Value Theory to obtain an estimate $\hat{L}_{f-g}$ that overestimates L_{f-g}, i.e., $\hat{L}_{f-g} \geq L_{f-g}$, with a user-defined probability ρ.

We build on *Wood and Zhang* (1996)-*Weng et al.* (2018), which find an estimate $\hat{L}_h$ of the Lipschitz constant L_h for a function $h(z)$ over a domain $\mathcal{Z}$ by estimating the location parameter γ of a three-parameter reverse Weibull distribution, which for a random variable W has the cumulative distribution function (CDF)

$$F_W(w) = \begin{cases} \exp(-(\frac{\gamma - w}{\alpha})^\beta), & \text{if } w < \gamma \\ 1, & \text{if } w \geq \gamma. \end{cases}$$

Here, the location parameter γ is the upper limit on the support of the distribution, and α and β are the scale and shape parameters, respectively. Consider the random variable described by the maximum slope taken over N_L pairs of i.i.d. samples $\{(z_1^i, z_2^i)\}_{i=1}^{N_L}$ from $\mathcal{Z}$, i.e., $s = \max_i \frac{\|h(z_1^i) - h(z_2^i)\|}{\|z_1^i - z_2^i\|}$. From the Fisher-Tippett-Gnedenko Theorem *De Haan and Ferreira* (2007), s follows one of the Frechet, reverse Weibull, or Gumbel distributions in the limit as N_L approaches infinity. If s follows the reverse Weibull distribution, which we validate in our results using the Kolmogorov-Smirnov (KS) goodness-of-fit test *DeGroot and Schervish* (2013) with a significance value of 0.05 (the same threshold used in *Weng et al.* (2018)), then L_h is finite and equals γ. We estimate L_h using the location parameter $\hat{\gamma}$ of a reverse Weibull distribution fit via maximum likelihood to N_S samples of s. Finally, we compute a confidence interval $c = \Phi^{-1}(\rho)\xi$ on $\hat{\gamma}$. Here ξ is the standard error of the fit $\hat{\gamma}$, which correlates with the quality of the fit, and $\Phi(\cdot)$ is the standard normal CDF *DeGroot and Schervish* (2013). We select the upper end of the confidence interval as our estimate $\hat{L}_h = \hat{\gamma} + c$, which overestimates L_h with probability ρ. Note that increasing ρ increases c, improving the safety probability at the cost of loosening $\hat{L}_h$, which can make planning more conservative. We also note that this probability is valid in the limit as N_L approaches infinity, due to the Fisher-Tippett-Gnedenko theorem making claims only on the asymptotic distribution. We summarize the estimation method in Alg. VIII.1.

Algorithm VIII.1: Lipschitz estimation for $h(z)$ over $\mathcal{Z}$

Input: N_S, N_L, ρ
for $j = 1, \ldots, N_S$ **do**
  sample $\{(z_1^{i,j}, z_2^{i,j})\}_{i=1}^{N_L}$ uniformly in $\mathcal{Z}$
  compute $s_j = \max_i \|h(z_1^{i,j}) - h(z_2^{i,j})\| / \|z_1^{i,j} - z_2^{i,j}\|$
fit reverse Weibull to $\{s_j\}$ to obtain $\hat{\gamma}$ and standard error ξ
validate fit using KS test with significance level 0.05
if validated **return** $\hat{L}_h = \hat{\gamma} + \Phi^{-1}(\rho)\xi$ **else return** failure

We wish to choose D to be large enough for planning while also keeping L_{f-g} small. To achieve this, we use a filtering procedure to reduce the impact of outliers in $\mathcal{S}$. Let μ and σ be the mean and standard deviation of the error over $\mathcal{S}$. Then, let

$\mathcal{S}_D = \{(\bar{x}, \bar{u}) \in \mathcal{S} \mid \|f(\bar{x}, \bar{u}) - g(\bar{x}, \bar{u})\| \leq \mu + a\sigma\}$ where a is a user-defined parameter. Then, we run Alg. VIII.2 in order to grow D. This method works by proposing values of r, estimating L_{f-g}, and increasing r until $r > \epsilon$ or $L_{f-g} \geq 1$. Finding D with $r > \epsilon$ and $L_{f-g} < 1$ is useful for planning (described further in Sec. 8.3.3, see (8.10)). Note that, in Euclidean spaces, $r \geq b_T$. If no filtering is done, $r = b_T$, since no point in D is further than a distance r from $\mathcal{S}_D$ and the furthest any point in D can lie from a point in $\mathcal{S}_D$ is r; however, filtering shrinks D and thus decreases the dispersion, making it possible that $r \geq b_T$. The parameter a should be chosen to balance the size of D against the magnitude of L_{f-g}, which we tune heuristically.

This filtering lets us exclude regions where our learned model is less accurate, yielding smaller e_T. Note that filtering does not affect the i.i.d. property of the samples needed for Alg. VIII.1; it only applies a mask to the domain. We also note that Alg. VIII.2 returns a minimum value for r, but a larger r can be chosen as long as L_{f-g} is estimated with Alg. VIII.1. A larger r makes planning easier by expanding the trusted domain.

Algorithm VIII.2: Selecting r and D

Input: μ, σ, a, S_D, Ψ, $\alpha > 0$

1 $r \leftarrow \mu + a\sigma$
2 **while** *True* **do**
3 construct D using equation (8.8)
4 estimate L_{f-g} using Alg. VIII.1 and Ψ
5 calculate ϵ using equation (8.6)
6 **if** $L_{f-g} \geq 1$ **then return** failure
7 **if** $r > \epsilon$ **then return** r and D
8 **else** $r \leftarrow \epsilon + \alpha$ `// α is a small constant`

While we never explicitly address the assumption that the true dynamics are deterministic, the estimated Lipschitz constant may be unbounded in the stochastic case, such as when two samples have the same inputs but different outputs due to noise, causing a division by 0 in line 3 of Alg. VIII.1.

L_{g_0} and L_{g_1} may also be estimated with Alg. VIII.1, which we employ in the results. Alternatively, *Fazlyab et al.* (2019) can give tight upper bounds on the Lipschitz constant of neural networks, though it could not scale to the networks used in our results. Other approaches *Jordan and Dimakis* (2020) improve scalability at the cost of looser Lipschitz upper bounds, and will be examined in the future.

8.3.3 Planning

We want to plan a trajectory from start x_I to goal x_G using the learned dynamics while remaining in $\mathcal{X}_{\text{safe}}$ in execution. We constrain the system to stay inside D, as model accuracy may degrade outside of the trusted domain. We develop a planner similar to a kinodynamic RRT *LaValle and James J. Kuffner* (2001), growing a search tree $\mathcal{T}$ by sampling controls that steer towards novel states until we reach the goal. If a path is found the we can ensure the goal is reachable with safety guarantees.

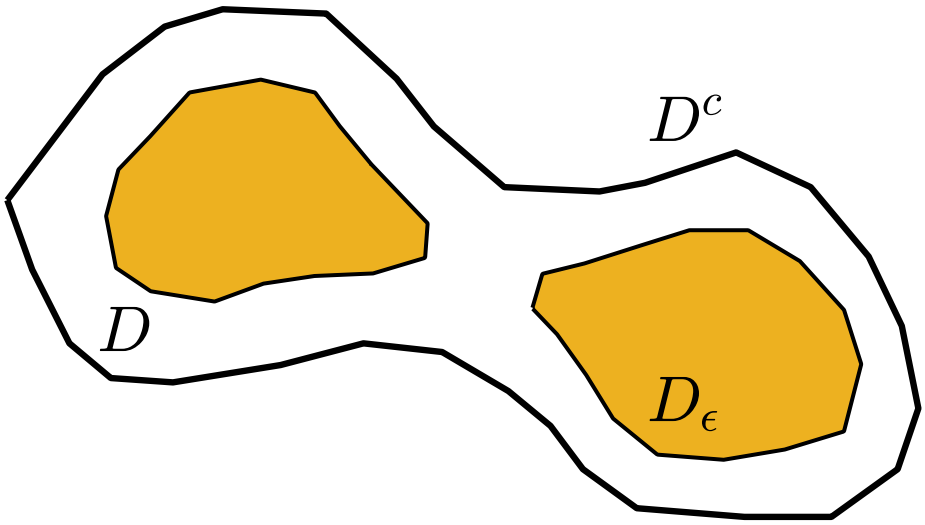


Figure 8.2: Visualizing D (boundary in black), D_ϵ (yellow), and D^c (complement of D). Each point in D_ϵ is at least ϵ distance away from D^c. If the system is controlled to a point in D_ϵ from anywhere in D under the learned dynamics, then it remains in D under the true dynamics.

8.3.3.1 Staying inside D

To remain inside the set D, we introduce another set $D_\epsilon := D \ominus \mathcal{B}_\epsilon(0)$, which is the Minkowski difference between D and a ball of radius ϵ. Every point in D_ϵ is at least a distance of ϵ from any point in the complement of D. Since the learned dynamics differs from the true dynamics by at most ϵ in D, controlling to a point in D_ϵ under the learned dynamics ensures the system remains within D under the true dynamics (see Fig. 8.2).

How do we determine if a query point (x, u) is inside of D_ϵ? Since we define D to be a union of balls (8.8), it would suffice to find a subset of training points $\mathcal{W} \subset S_D$ such that the union of r-balls about the training points completely covers an ϵ-ball about (x, u). Explicitly,

$$\bigcup_{(\bar{x},\bar{u})\in\mathcal{W}} \mathcal{B}_r(\bar{x}, \bar{u}) \supset \mathcal{B}_\epsilon(x, u) \tag{8.9}$$

In general, checking (8.9) is difficult, but if $L_{f-g} < 1$ and

$$r > \frac{e_T}{1 - L_{f-g}}, \tag{8.10}$$

then only one training point within a distance $r - \epsilon$ is needed to ensure a query point is in D_ϵ (see Fig. 8.3). Note by Alg. VIII.2 lines 6-7, either (8.10) is guaranteed or $L_{f-g} \geq 1$, in which we return failure.

Lemma VIII.3. *If $L_{f-g} < 1$ and r is selected according to equation (8.10), then a point (x, u) is in D_ϵ if there exists $(\bar{x}, \bar{u}) \in S$ such that $\|(x, u) - (\bar{x}, \bar{u})\| \leq r - \epsilon$.*

Proof. To prove, note we can rearrange terms in equation (8.10) to get $r > L_{f-g}r + e_T \geq \epsilon$. If there exists $(\bar{x}, \bar{u}) \in S_D$ such that $\|(x, u) - (\bar{x}, \bar{u})\| \leq r - \epsilon$, then $\mathcal{B}_\epsilon(x, u) \subset$

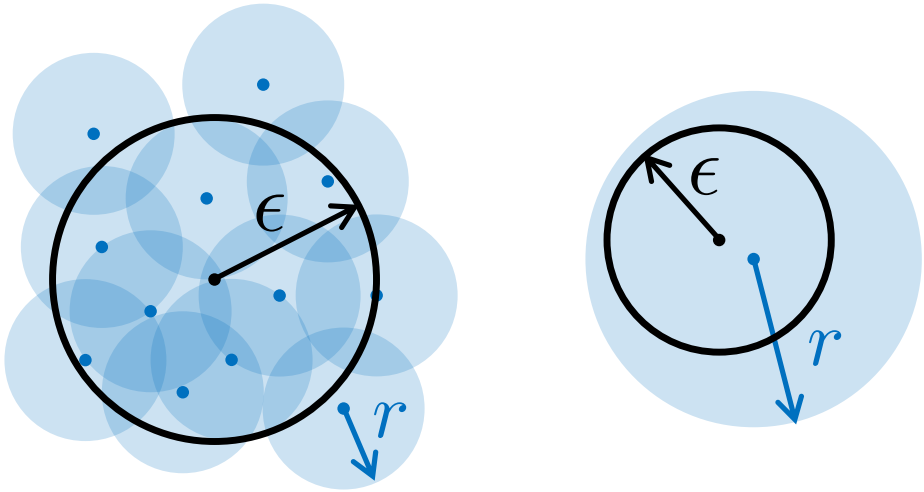


Figure 8.3: Illustrating the advantage of $L_{f-g} < 1$ and r selected according to (8.10). An ϵ-ball about a query point is shown in black; r-balls about training data are shown in blue. **Left**: $L_{f-g} > 1$, therefore requiring many training points to cover an ϵ-ball about the query point. **Right**: $L_{f-g} < 1$ and r is selected according to (8.10). Under these conditions, only one training point within a $r - \epsilon$ distance ensures an ϵ-ball about the (x, u) is entirely in D, ensuring that the query point is in D_ϵ.

$\mathcal{B}_r(\bar{x}, \bar{u}) \subset D$ since no point in $\mathcal{B}_\epsilon(x, u)$ is further than r distance from $(\bar{x}, \bar{u})$. Since $\mathcal{B}_\epsilon(x, u) \subset D$, (x, u) is at least ϵ distance from any point in D^c and therefore $(x, u) \in D_\epsilon$. □

In order to ensure $L_{f-g} < 1$, since it is derived from the training data and learned model, we must train a learned model that is sufficiently accurate (i.e., low error on $\mathcal{S} \cup \Psi$). In our experiments, it was enough to minimize mean squared error over the training set to learn models with this property.

To ensure that the resulting trajectory remains in D_ϵ, we ensure that corresponding pairs of state and control lie in D_ϵ at each step. In growing the search tree $\mathcal{T}$, we break down this requirement into two separate checks, the first of which optimistically adds states to the search tree and the second that requires pairs of states and controls to lie in D_ϵ. To illustrate, suppose we sample a new configuration x_{new} and grow the tree from some x to x_{new}. At this point, when sampling a control u to steer from x to x_{new} we enforce that $(x, u) \in D_\epsilon$ (see line 11 in Alg. VIII.3). However, how do we know the resulting state, $x' = g(x, u)$, will lie in D_ϵ? Since x' is a state and not a state-control pair, the above question is not well defined. Instead, we perform an optimistic check in adding x' which requires that there exists some $\hat{u}$ such that $(x', \hat{u}) \in D_\epsilon$ (see line 14 in Alg. VIII.3). In turn, when growing the search tree from x' to some other sampled point x'_{new} we ensure that the pair of state and newly sampled control u' lies in D_ϵ, i.e., $(x', u') \in D_\epsilon$.

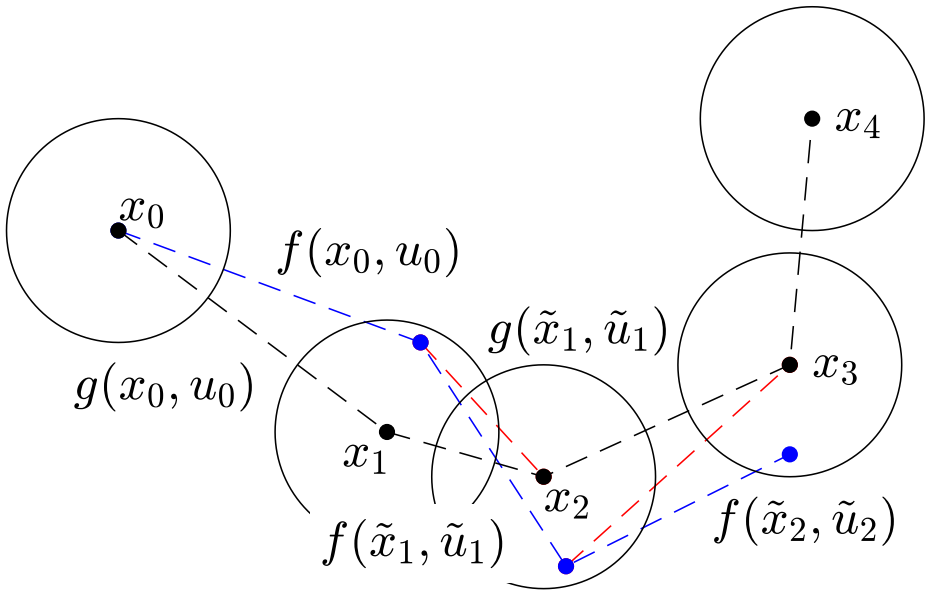


Figure 8.4: The one-step feedback law: plan with the learned dynamics (dashed black); rollout with the true dynamics (blue); prediction with the learned dynamics using the feedback law (red). At each point, we use (8.12) to find a feedback control $\tilde{u}_k$ so $x_{k+1} = g(\tilde{x}_k, \tilde{u}_k)$. We arrive within ϵ of the next state under the true dynamics. This repeats until we reach the goal.

8.3.3.2 One step feedback law

To prevent drift in execution, we also seek to ensure the trajectory planned with RRT can be tracked with minimal error. One key requirement to guarantee a feedback law exists is that the system is sufficiently actuated under the learned dynamics. This requires that $\texttt{dim}(\mathcal{U}) \geq \texttt{dim}(\mathcal{X})$. The check for sufficient actuation is done on a per state basis and can be done as we grow $\mathcal{T}$. This feedback law ensures that, under the learned dynamics, we can return to a planned trajectory in exactly one step.

Suppose we are executing a trajectory $(x_0, \ldots, x_K)$ with corresponding control $(u_0, \ldots, u_{K-1})$ planned with the learned dynamics, and the system is currently at x_{k-1}. Under the learned dynamics, the plan is to move to $x_k = g(x_{k-1}, u_{k-1})$, but, under the true dynamics, the system will end up at some $\tilde{x}_k = f(x_{k-1}, u_{k-1})$ which is no more than an ϵ distance from x_k. Our goal is to find an input $\tilde{u}_k$ such that $x_{k+1} = g(\tilde{x}_k, \tilde{u}_k)$. If this one-step feedback law exists for all $1 \leq k \leq K-1$, it ensures the executed trajectory stays within ϵ distance of the planned trajectory (see Fig. 8.4).

With Lipschitz constants L_{g_0} and L_{g_1}, we can bound how much the learned dynamics varies in the ϵ-ball about x_k.

$$\forall \tilde{x}_k \in \mathcal{B}_\epsilon(x_k) \;\; g(\tilde{x}_k, u) = g_0(x_k) + \Delta_0 + (g_1(x_k) + \Delta_1)u \tag{8.11}$$

where $\|\Delta_0\| \leq L_{g_0}\epsilon$ and $\|\Delta_1\| \leq L_{g_1}\epsilon$. With (8.11), the existence of $\tilde{u}_k$ is informed by

a perturbed linear equation:

$$\begin{aligned} & x_{k+1} = g(\tilde{x}_k, \tilde{u}_k), \quad \tilde{x}_k \in \mathcal{B}_\epsilon(x_k) \\ & \Rightarrow x_{k+1} = g_0(x_k) + \Delta_0 + (g_1(x_k) + \Delta_1)\tilde{u}_k \\ & \Rightarrow A\tilde{u}_k = b \\ \text{with } & A = g_1(x_k) + \Delta_1 \text{ and } b = x_{k+1} - g_0(x_k) - \Delta_0 \end{aligned} \tag{8.12}$$

Prior to execution, we seek to answer two questions: when does $\tilde{u}_k$ exist and does $\tilde{u}_k$ lie in the control space $\mathcal{U}$ (for instance in the presence of box constraints)? Results from the literature *Lötstedt* (1983) give a bound on the difference between the nominal solution u_k and perturbed solution $\tilde{u}_k$,

$$\|u_k - \tilde{u}_k\| \leq \frac{\|g_1(x_k)^+\|(\|\Delta_1\|\|u_k\| + \|\Delta_0\|)}{1 - \|g_1(x_k)^+\|\|\Delta_1\|} \doteq u_{\text{pert}}, \tag{8.13}$$

where $g_1(x_k)^+$ is the pseudo-inverse of $g_1(x_k)$ (in general $g_1(x_k)$ is not square). We can use this bound to ensure that $\tilde{u}_k$ is guaranteed to lie in $\mathcal{U}$ by enforcing that $u_k + u_{\text{pert}}\mathbf{1}_\infty \subseteq \mathcal{U}$, where $\mathbf{1}_\infty$ is the unit infinity-norm ball. Furthermore, A may become singular if $1 - \|g_1(x_k)^+\| \, \|\Delta_1\| \leq 0$. In this case, $\tilde{u}_k$ is not guaranteed to exist.

If $\tilde{u}_k$ exists and satisfies the control constraints for all $1 \leq k \leq K-1$, then we ensure that the system will track the path up to an ϵ error under the one-step feedback law. In planning, we add the existence of a valid one step feedback law as a check when growing the search tree. Formally:

Theorem VIII.4. *For trajectory $(x_0, \ldots, x_K)$ and $(u_0, \ldots u_{K-1})$, if the solution to the perturbed linear equation* (8.12)*, $\tilde{u}_k$, exists for all $k \in \{1, \ldots, K-1\}$, then under the true dynamics $\|\tilde{x}_k - x_k\| \leq \epsilon$ for all k, given L_{f-g}, L_{g_0}, and L_{g_1} are each an overestimate of the true Lipschitz constant of $f - g$, g_0, and g_1, respectively.*

Proof. Proof by induction. For the induction step, assume $\|\tilde{x}_k - x_k\| \leq \epsilon$ for some k. Since $\tilde{x}_k \in \mathcal{B}_\epsilon(x_k)$, the perturbed linear equation (8.12) is valid. If a solution exists, then $x_{k+1} = g(\tilde{x}_k, \tilde{u}_k)$ and $\|f(\tilde{x}_k, \tilde{u}_k) - x_{k+1}\| \leq \epsilon$. This satisfies the induction step. For the base case, we have $g(x_0, u_0) = x_1$ and $\|f(x_0, u_0) - x_1\| \leq \epsilon$. Thus, for all k, $\|\tilde{x}_k - x_k\| \leq \epsilon$. □

8.3.3.3 Ensuring safety and invariance about the goal

Since it is guaranteed by Thm. VIII.4 that $\|\tilde{x} - x_k\| \leq \epsilon$, we check that $\mathcal{B}_\epsilon(x_k) \subset \mathcal{X}_{\text{safe}}$ for each x_k on the path to ensure safety.

The exact nature of this check depends on the system and definition of $\mathcal{X}_{\text{unsafe}}$. For example, in our experiments on quadrotor, the state includes the quadrotor's position in $\mathbb{R}^3$ and $\mathcal{X}_{\text{unsafe}}$ is defined by unions of boxes in $\mathbb{R}^3$. By defining a bounding sphere that completely contains the quadrotor, we can verify a path is safe via sphere-box intersection. With the Kuka arm, we randomly sample joint configurations in an ϵ-ball about states, transform the joint configurations via forward kinematics, and check collisions in workspace. While this method is not guaranteed to validate the

entire ball around a state, in practice no collisions resulted from execution of plans. Another approach computes a free-space bubble *Quinlan* (1994) around a given state x and check if it contains $\mathcal{B}_\epsilon(x)$, however this is known to be conservative.

To stay near the goal after executing the trajectory, we use the same perturbed linear equation to ensure the existence of a one-step feedback law. Here, rather than checking the next state along the trajectory is reachable from the previous, we check that the final state is reachable from itself, i.e., x_K is reachable from x_K. Similar to the arguments above, we can repeatedly execute the feedback law to ensure the system remains in an $(\epsilon + \lambda)$-ball about the goal. Formally, we have:

Theorem VIII.5. *If the solution, denoted u_{st}, to the perturbed linear equation exists for $A = g_1(x_K) + \Delta_1$ and $b = x_K - g_0(x_K) - \Delta_0$ for all $x \in \mathcal{B}_\epsilon(x_K)$, then the closed loop system will remain in $\mathcal{B}_{\epsilon+\lambda}(x_G)$, given L_{f-g}, L_{g_0}, and L_{g_1} are each an overestimate of the true Lipschitz constant of $f - g$, g_0, and g_1, respectively.*

Proof. By Thm. VIII.4, $\|\tilde{x}_K - x_K\| \leq \epsilon$. Thus, if the solution to the perturbed linear equation with $A = g_1(x_K) + \Delta_1$ and $b = x_K - g_0(x_K) - \Delta_0$ exists and is valid then $g(\tilde{x}_K, u_{\text{st}}) = x_K$ and $\|f(\tilde{x}_K, u_{\text{st}}) - x_K\| \leq \epsilon$. Since $\|x_K - x_G\| \leq \lambda$, the system remains in $\mathcal{B}_{\epsilon+\lambda}(x_K)$ by the triangle inequality. □

To close, we note that the overall safety and invariance probability of our method is ρ^3, arising from our need to estimate three Lipschitz constants: L_{f-g}, L_{g_0}, and L_{g_1}. Given independent samples for overestimating each constant with probability ρ via Alg. VIII.1, the overall correctness probability is the product of the correctness of each constant, i.e., ρ^3.

8.3.4 Algorithm

We present our full method, **L**earned **M**odels in **T**rusted **D**omains (LMTD-RRT), in Alg. VIII.3. In practice, we implemented `SampleState` and `SampleControl` in two different ways: uniform sampling and perturbations from training data. Sampling perturbations (up to a norm of $r - \epsilon$) does not exclude valid (x, u) pairs since all points in D_ϵ lie within $r - \epsilon$ from a training point, and, in cases where D_ϵ is a relatively small volume, can yield a faster search. However, it also biases samples near regions where training data is more dense. We define the set $\mathcal{S}_\mathcal{X} = \{\bar{x} \mid \exists \bar{u} \text{ s.t. } (\bar{x}, \bar{u}) \in \mathcal{S}_D\}$ to describe the optimistic check described in Sec. 8.3.3.1. `NN` finds the nearest neighbor and `OneStep` checks that a valid feedback exists as described in Sec. 8.3.3.2. `Model` evaluates the learned dynamics and `InCollision` checks if an ϵ-ball is in $\mathcal{X}_{\text{safe}}$ as described in Sec. 8.3.3.3.

Once a plan has been computed, it can be executed in closed-loop with Alg. VIII.4. `ModelG0` and `ModelG1` evaluate g_0 and g_1 of the learned model. `SolveLE` solves the linear equation and `Dynamics` executes the true dynamics f.

8.4 Results

We present results on 1) a 2D system to illustrate the need for remaining near the trusted domain, 2) a 6D quadrotor to show scaling to higher-dimensional systems,

Algorithm VIII.3: LMTD-RRT

Input: x_I, x_G, $S_{\mathcal{X}}$, S_D, r, ϵ, λ, N_{samples}, goal_bias

$\mathcal{T} \leftarrow \{x_I\}$
while True **do**
  while ¬sampled **do**
    $x_{\text{new}} \leftarrow$ `SampleState(`goal_bias`)`
    if $\|x_{\text{new}} -$ `NN(`$S_{\mathcal{X}}$, x_{new}`)`$\| \leq r - \epsilon$ **then**
      sampled $\leftarrow$ True
  $x_{\text{near}} \leftarrow$ `NN(`$\mathcal{T}$, x_{new}`)`
  $i \leftarrow 0, u_{\text{best}} \leftarrow \emptyset, x_{\text{best}} \leftarrow \emptyset, d \leftarrow \infty$
  while $i < N_{\text{samples}}$ **do**
    $u \leftarrow$ `SampleControl()`
    if $\|(x_{near}, u) -$`NN(`S_D, (x_{near}, u)`)`$\| \leq r - \epsilon$ **then**
      $x_{\text{next}} \leftarrow$ `Model(`x_{near}, u`)`
      if `OneStep(`x_{near}, u, x_{next}`)` $\wedge$
        $\|x_{\text{next}} -$ `NN(`$S_{\mathcal{X}}$, x_{next}`)`$\| \leq r - \epsilon \wedge$
        $\|x_{\text{next}} - x_G\| < d \wedge$
        ¬`InCollision(`x_{next}, ϵ`)` **then**
          $u_{\text{best}} \leftarrow u$, $x_{\text{best}} \leftarrow x_{\text{next}}$
          $d \leftarrow \|x_{\text{next}} - x_G\|$
    $i \leftarrow i + 1$
  if u_{best} **then**
    $\mathcal{T} \leftarrow \mathcal{T} \cup \{x_{\text{best}}\}$
    if $\|x_{\text{best}} - x_G\| \leq \lambda$ **then**
      return `ConstructPath(`$\mathcal{T}$, x_{best}`)`

Algorithm VIII.4: LMTD-Execute

Input: $\{x_k\}_{k=0}^{K}$, $\{u_k\}_{k=0}^{K-1}$

$\tilde{x}_0 \leftarrow x_0$, $k \leftarrow 0$
for $k = 1 \ldots n-1$ **do**
  $b \leftarrow x_{k+1} -$ `ModelG0(`$\tilde{x}_k$`)`, $A \leftarrow$`ModelG1(`$\tilde{x}_k$`)`
  $\tilde{u}_k \leftarrow$`SolveLE(`A, b`)`
  $\tilde{x}_{k+1} \leftarrow$ `Dynamics(`$\tilde{x}_k$, $\tilde{u}_k$`)`

and 3) a 7DOF Kuka arm simulated in Mujoco *Todorov et al.* (2012) to show scaling to complex dynamics that are not available in closed form. Using $\rho = 0.975$, we plan with LMTD-RRT and rollout the plans in open-loop (no computation of $\tilde{u}_k$) and closed-loop (Alg. VIII.4). We compare with a naïve kinodynamic RRT that skips the checks on lines 5, 11, 13-14 of Alg. VIII.3 in both open and closed loop. See the video for experiment visualizations.

8.4.1 2D Sinusoidal Model

To aid in visualization, we demonstrate LMTD-RRT on a 2D system with dynamics $f(x, u) = f_0(x) + f_1(x)u$:

$$f_0(x) = \begin{bmatrix} x \\ y \end{bmatrix} + \Delta T \left[\begin{array}{c|c} 3\sin(0.3(x+4.5)) & \sin\left(0.3(y+4.5)\right) \\ 3\sin(0.3(y+4.5)) & \sin\left(0.3(x+4.5)\right) \end{array}\right]$$

$$f_1(x) = \Delta T \begin{bmatrix} 1 + 0.05\cos(y) & 0 \\ 0 & 1 + 0.05\sin(x) \end{bmatrix}$$

where $\Delta T = 0.2$. We are given 9000 training points $(x_i, u_i, f(x_i, u_i))$, where x_i is drawn uniformly from an 'L'-shaped subset of $\mathcal{X}$ (see Fig. 8.5) and u_i is drawn uniformly from $\mathcal{U} = [-1, 1]^2$. $g_0(x)$ and $g_1(x)$ are modeled with separate neural networks with one hidden layer of size 128 and 512, respectively. We select $a = 3$ in Alg. VIII.2. 1000 more samples are used to estimate L_{f-g} via Alg. VIII.1, which we validate with a KS test with a p value of 0.56, far above the 0.05 threshold significance value. We obtain $\hat{\gamma} = 0.117$ and $c = 6.85 \times 10^{-4}$, giving $\epsilon = 0.215$ over D.

See Fig. 8.5 for examples of the nominal, open-loop, and closed-loop trajectories planned with LMTD-RRT and a naïve kinodynamic RRT. The plan computed with LMTD-RRT remains in regions where we can trust the learned model (i.e. within D_ϵ) and the closed-loop execution of the trajectory converges to $\mathcal{B}_{\epsilon+\lambda}(x_G)$. In contrast, both the open-loop and closed-loop execution of the naïve RRT plan diverge. We provide statistics in Table 8.1 of maximum ℓ_2 tracking error $\max_{i \in \{1,\dots,T\}} \|\tilde{x}_i - x_i\|$ and final ℓ_2 distance to the goal $\|\tilde{x}_T - x_G\|_2$ for both the open loop (OL) and closed loop (CL) variants, averaged over 70 random start/goal states. To give the baseline an advantage, we fix the start/goal states and plan with naïve RRT using two different dynamics models: 1) the same learned dynamics model used in LMTD-RRT and 2) a learned dynamics model with the same hyperparameters trained on the full dataset (10^4 datapoints), and report the statistics on the minimum of the two errors. The worst case tracking error for the plan computed with LMTD-RRT was 0.199, which is within the guaranteed tracking error bound of $\epsilon = 0.215$, while despite the data advantage, plans computed with naïve RRT suffer from higher tracking error. Average planning times for LMTD-RRT and naïve RRT are 4.5 and 17 seconds, respectively. Overall, this suggests that planning with LMTD-RRT avoids regions where model error may lead to poor tracking, unlike planning with a naïve RRT.

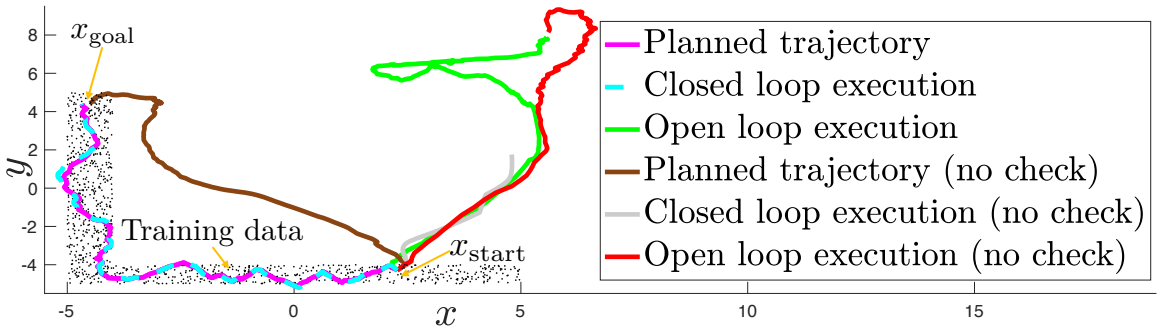


Figure 8.5: 2D sinusoidal dynamics. The LMTD-RRT plan (magenta) stays in D and ensures a valid feedback law exists at each step. The plan can be tracked within ϵ under closed loop control (cyan). If feedback is not applied, the system drifts to the edge of the trusted domain, exits, and diverges (green). The naïve RRT plan (brown) does not consider D, and does not reach the goal under closed loop (grey) or open loop (red) control.

	LMTD-RRT	Naïve kino. RRT
Max. trck. err. (CL)	0.099 ± 0.036 (0.199)	8.746 ± 4.195 (15.21)
Goal error (CL)	0.039 ± 0.020 (0.113)	7.855 ± 3.851 (14.78)
Max. trck. err. (OL)	12.84 ± 4.444 (20.92)	10.39 ± 1.962 (15.31)
Goal error (OL)	12.40 ± 4.576 (20.92)	10.12 ± 1.762 (15.20)

Table 8.1: Sinusoid errors in closed loop (CL) and open loop (OL).
Mean ± standard deviation (worst case).

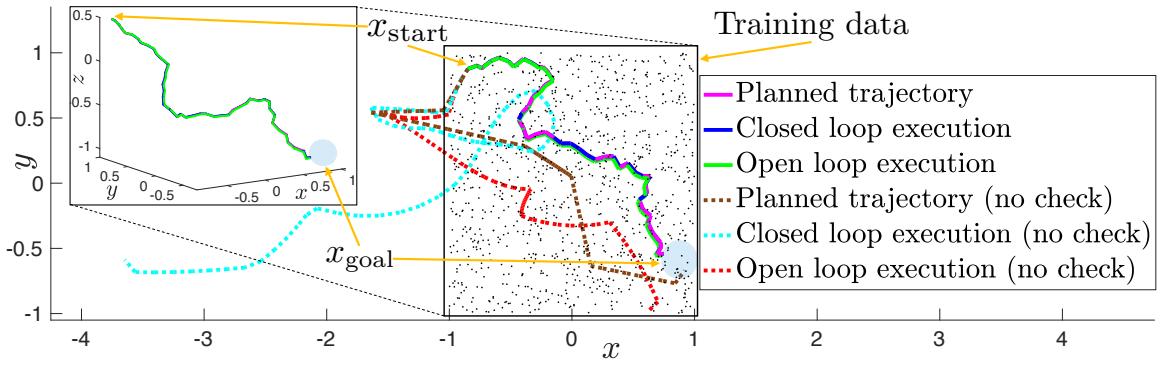


Figure 8.6: Quadrotor tracking example. The trajectory planned with LMTD-RRT (magenta) is tracked in closed loop (blue) and reaches the goal. The open loop (green) also converges near the goal, but not as close as the closed loop. The naïve RRT produces a plan (brown) that leaves the trusted domain. Thus, both the open (red) and closed (light blue) loop rapidly diverge.

8.4.2 6D Quadrotor Model

We evaluate our method on 6-dimensional fully-actuated quadrotor dynamics *Sabatino* (2015) with state $x = [\chi, y, z, \phi, \theta, \psi]^\top$, where $f(x,u) = f_0(x) + f_1(x)u$, $f_0(x) = x$ and $f_1(x) =$

$$\Delta T \begin{bmatrix} c_\theta c_\psi & -c_\phi s_\psi + c_\psi s_\phi s_\theta & s_\psi s_\phi + c_\phi c_\psi s_\theta & 0 & 0 & 0 \\ c_\theta s_\psi & c_\phi c_\psi + s_\phi s_\psi s_\theta & -c_\psi s_\phi + c_\phi s_\psi s_\theta & 0 & 0 & 0 \\ -s_\theta & c_\theta s_\phi & c_\phi c_\theta & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & s_\phi t_\theta & c_\phi t_\theta \\ 0 & 0 & 0 & 0 & c_\phi & -s_\phi \\ 0 & 0 & 0 & 0 & s_\phi / c_\theta & c_\phi / c_\theta \end{bmatrix},$$

where $\Delta T = 0.1$ and $s_{(\cdot)}$, $c_{(\cdot)}$, and $t_{(\cdot)}$ are short for $\sin(\cdot)$, $\cos(\cdot)$, and $\tan(\cdot)$ respectively. We are given 9×10^6 training data tuples $(x_i, u_i, f(x_i, u_i))$, where x_i, u_i are generated with Halton sampling over $[-1,1]^3 \times [-\frac{\pi}{20}, \frac{\pi}{20}]^3$ and $[-1,1]^6$, respectively (data is collected near hover). As $f_0(x)$ is a simple integrator term, we assume it is known and we set $g_0(x) = x$, while $g_1(x)$ is learned with a neural network with one hidden layer of size 4000. We select $a = 6$ in Alg. VIII.2. We use 10^6 more samples in Alg. VIII.1 to estimate L_{f-g}, and conduct a KS test resulting in a p-value of $0.43 \gg 0.05$. We obtain $\hat{\gamma} = 0.205$, $c = 0.011$, and $\epsilon = 0.134$.

See Fig. 8.6 for examples of the planned, open-loop, and closed-loop trajectories planned with LMTD-RRT and a naïve RRT. The trajectory planned with LMTD-RRT remains close to the training data, and the closed-loop system tracks the planned path with ϵ-accuracy converging to $\mathcal{B}_{\epsilon+\lambda}(x_G)$. We note that using the feedback controller to track trajectories planned with naïve RRT tends to worsen the tracking error, implying our learned model is highly inaccurate outside of the domain. We provide statistics in Table 8.2 for maximum tracking error and distance to goal, averaged over 100 random start/goal states. The worst case closed-loop tracking error for trajectories planned with LMTD-RRT is 0.011, again much smaller than ϵ. As with the 2D example, we give the baseline an advantage in computing tracking error statistics by reporting the minimum of the two errors when planning with 1) the same model used in LMTD-RRT and 2) a model trained on the full dataset (10^7 points). Despite the data advantage, the plans computed using naïve RRT have much higher tracking error. Average planning times for our unoptimized code are 100 sec. for LMTD-RRT and 15 min. for naïve RRT, suggesting that sampling focused near the training data can improve planning efficiency.

We also evaluate LMTD-RRT on an obstacle avoidance problem (Fig. 8.7). We perform collision checking as described in Sec. 8.3.3.3. As the tracking error tubes (of radius $\epsilon = 0.134$) centered around the nominal trajectories never intersect with any obstacles, we can guarantee that the system never collides in execution. Empirically, in running Alg. VIII.3 over 500 random seeds to obtain different nominal paths, the closed-loop trajectory never collides. In contrast, the naïve RRT plan fails to be tracked and collides (Fig. 8.7, right).

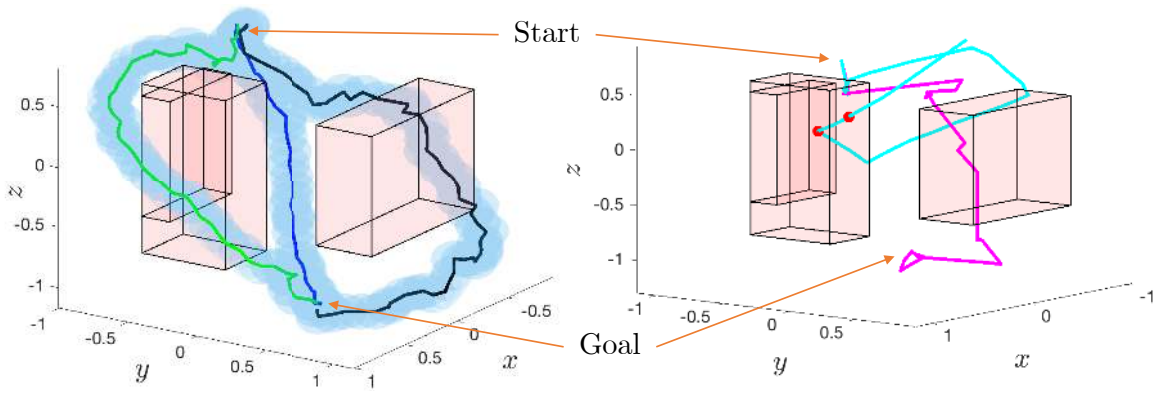


Figure 8.7: **Left**: Quadrotor obstacle (red) avoidance. Example plans (green, blue, black), tracking error bound ϵ overlaid (light blue). Closed-loop trajectories remain in the tubes, converging to the goal without colliding. **Right**: Naïve RRT plan (pink) fails to be tracked (cyan) and collides (red dots).

	LMTD-RRT	Naïve kino. RRT
Max. trck. err. (CL)	0.003 ± 0.001 (0.008)	10.59 ± 16.75 (153.26)
Goal error (CL)	0.001 ± 0.001 (0.004)	8.247 ± 8.434 (46.150)
Max. trck. err. (OL)	0.020 ± 0.007 (0.040)	4.289 ± 2.340 (12.986)
Goal error (OL)	0.019 ± 0.008 (0.040)	3.265 ± 2.028 (11.670)

Table 8.2: Quadrotor errors (no obstacles) in closed loop (CL) and open loop (OL). Mean ± standard deviation (worst case).

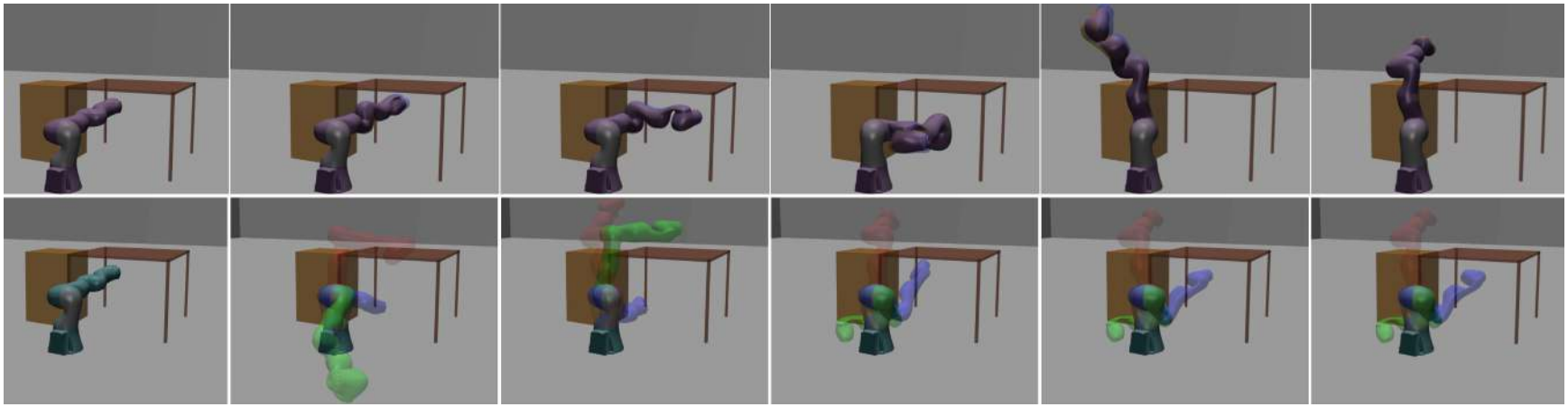


Figure 8.8: Planning to move a 7DOF arm from below to above a table. Trajectory-tracking time-lapse (time increases from left to right). Red (nominal), green (closed loop), blue (open loop). **Top**: LMTD-RRT (red, green, blue overlap due to tight tracking). **Bottom**: Naïve RRT (poor tracking causes collision).

	LMTD-RRT	Naïve kino. RRT
Max. trck. err. (CL)	0.010 ± 0.010 (0.038)	0.090 ± 0.250 (1.265)
Goal error (CL)	0.004 ± 0.004 (0.018)	0.082 ± 0.251 (1.265)
Max. trck. err. (OL)	0.036 ± 0.041 (0.142)	0.076 ± 0.084 (0.282)
Goal error (OL)	0.035 ± 0.040 (0.140)	0.071 ± 0.075 (0.225)

Table 8.3: 7DOF arm errors (no obstacles) in closed loop (CL) and open loop (OL). Mean ± standard deviation (worst case).

8.4.3 7DOF Kuka Arm in Mujoco

We evaluate our method on a 7DOF Kuka iiwa arm simulated in Mujoco *Todorov et al.* (2012) using a Kuka model from *gym-kuka mujoco* (2019). We train two models using different datasets, one for evaluating tracking error without the presence of obstacles (Table 8.3), and the other for obstacle avoidance. For both models, $g_0(x)$ is again set to be x while $g_1(x)$ is learned with a neural network with one hidden layer of size 4000. For the results in Table 8.3, we are provided 2475 training data tuples, which are collected by recording continuous state-control trajectories from an expert and evaluating $f(x, u)$ on the trajectories and on random state-control perturbations locally around the trajectories. We select $a = 5$ in Alg. VIII.2. 275 more samples are used in Alg. VIII.1 to estimate L_{f-g}, validated with a KS test with a p value of $0.58 \gg 0.05$. We obtain $\hat{\gamma} = 0.087$ and $c = 0.001$, leading to $\epsilon = 0.111$. In Table 8.3, we provide statistics on maximum tracking error and distance to goal under plans with LMTD-RRT and the naïve RRT baseline (with a model trained on the full dataset of 2750 points), averaged over 25 runs of each method. Notably, closed-loop tracking of plans found with LMTD-RRT have lowest error, with a worst case error much smaller than $\epsilon = 0.111$. Planning takes on average 1.552 and 0.167 sec. for LMTD-RRT and naïve RRT, respectively. We suspect the naïve RRT exploits poor dynamics outside of D, expediting planning.

For the obstacle avoidance example (Fig. 8.8), we are provided 15266 datapoints, which again take the form of continuous trajectories plus perturbations. We select $a = 8$ in Alg. VIII.2, and use 1696 more points to estimate L_{f-g} using Alg. VIII.1, which we validate with a KS test with a p value of $0.37 \gg 0.05$. We obtain $\hat{\gamma} = 0.156$ and $c = 0.010$, leading to $\epsilon = 0.111$. In planning, as described in Sec. 8.3.3.3, we perform collision checking by randomly sampling configurations in an ϵ-ball about each point along the trajectory. Though this collision checker is not guaranteed to detect collision, in running LMTD-RRT over 20 random seeds, we did not observe collisions in execution for any of the 20 plans, and the arm safely reaches the goal without collision. Over these trajectories, the worst case tracking error is 0.107, which remains within $\epsilon = 0.111$. One such plan computed by LMTD-RRT and the corresponding open-loop and closed-loop tracking trajectories, is shown in the top row of Fig. 8.8. The three trajectories nearly overlap exactly due to small tracking error. In contrast, the naïve RRT plan cannot be accurately tracked, even with closed-loop control, due to planning outside of the trusted domain, causing the executed trajectories to diverge and collide with the table.

8.5 Discussion and Conclusion

We present a method to bound the difference between learned and true dynamics in a given domain and derive conditions that guarantee a one-step feedback law exists. We combine these two properties to design a planner that can guarantee safety, goal reachability, and that the closed-loop system remains in a small region about the goal.

While the method presented has strong guarantees, it also has limitations which are interesting targets for future work. First, the true dynamics are assumed to be deterministic. Stochastic dynamics may be possible by estimating the Lipschitz constant of the mean dynamics while also appropriately modeling the noise. Second, the actuation requirement limits the systems that this method can be applied to. For systems with $\texttt{dim}(\mathcal{U}) < \texttt{dim}(\mathcal{X})$, it may be possible to construct a similar feedback law that guarantees the learned dynamics will lie within a tolerance of planned states which, in turn, could still give strong guarantees on safety and reachability; we will present a method for doing precisely this in the next chapter.

CHAPTER IX

Safe Planning and Execution with Learned Underactuated Dynamics via Contraction Theory

In this chapter, we present a method for contraction-based feedback motion planning of locally incrementally exponentially stabilizable systems with unknown dynamics that provides probabilistic safety and reachability guarantees. Given a dynamics dataset, our method learns a deep control-affine approximation of the dynamics. To find a trusted domain where this model can be used for planning, we obtain an estimate of the Lipschitz constant of the model error, which is valid with a given probability, in a region around the training data, providing a local, spatially-varying model error bound. We derive a trajectory tracking error bound for a contraction-based controller that is subjected to this model error, and then learn a controller that optimizes this tracking bound. With a given probability, we verify the correctness of the controller and tracking error bound in the trusted domain. We then use the trajectory error bound together with the trusted domain to guide a sampling-based planner to return trajectories that can be robustly tracked in execution. We show results on a 4D car, a 6D quadrotor, and a 22D deformable object manipulation task, showing our method plans safely with learned models of high-dimensional underactuated systems, while baselines that plan without considering the tracking error bound or the trusted domain can fail to stabilize the system and become unsafe. This chapter is based on the paper *Chou et al.* (2021c).

9.1 Introduction

Provably safe motion planning algorithms for unknown systems are critical for deploying robots in the real world. While planners are reliable when the system dynamics are known exactly, the dynamics often may be poorly modeled or unknown. To address this, data-driven methods (i.e., model-based reinforcement learning) learn the dynamics from data and plan with the learned model. However, such methods can be unsafe, in part because the planner can and will exploit errors in the learned dynamics to return trajectories which cannot actually be tracked on the real system, leading to unpredictable, unsafe behavior when executed. Thus, to guarantee safety, it is of major interest to establish a bound on the error that the true system may see

when attempting to track a trajectory planned with the learned dynamics, and to use it to guide the planning of robustly-trackable trajectories.

One key property of learned dynamics models is that they have varying error across the state space: they should be more accurate on the training data, and that accuracy should degrade when moving away from it. Thus, the model error that the system will see in execution will depend on the domain that it visits. This reachable domain also depends on the tracking controller; for instance, a poor controller may lead to the system visiting a larger set of possible states, and thus experiencing a larger possible model error. To analyze this, we need a bound on the trajectory tracking error for a given disturbance description (a tracking tube). In this chapter, we consider tracking controllers based on contraction theory. Introduced in *Lohmiller and Slotine* (1998) for autonomous systems and extended to the control-affine case in *Manchester and Slotine* (2017), control contraction theory studies the incremental stabilizability of a system, making it uniquely suited for obtaining trajectory tracking tubes under disturbance. In the past, tracking tubes have been derived for contraction-based controllers under simple uniform disturbance bounds *Singh et al.* (2019) (i.e., a UAV subject to wind with a known uniform upper bound). However, these assumptions are ill-suited for handling learned model error. Assuming a uniform disturbance bound over the space can be highly conservative, since the large model error far from the training data would yield enormous tracking tubes, rendering planning entirely infeasible. To complicate things further, obtaining an upper bound on the model error can be challenging, as we only know the value of the error on the training data.

To address this gap, we develop a method for safe contraction-based motion planning with learned dynamics models. In particular, our method is designed for high-dimensional neural network (NN) learned dynamics models, and provides probabilistic guarantees on safety and goal reachability for the true system. Our core insight is that we can derive a tracking error bound for a contraction-based controller under a spatially-varying model error description, and that we can use this error bound to bias planning towards regions in the state/control space where trajectories can be more robustly tracked. We summarize our contributions as:

- A trajectory tracking error bound for contraction-based controllers subjected to a spatially-varying, Lipschitz constant-based model error bound that accurately reflects the learned model error.
- A deep learning framework for joint learning of dynamics, control contraction metrics (CCMs), and contracting controllers that are approximately optimized for planning performance under this model error description.
- A sampling-based planner that returns plans which can be safely tracked under the learned dynamics/controller.
- Evaluation of our method on learned dynamics up to 22D, and demonstrating that it outperforms baselines.

9.2 Preliminaries and Problem Statement

9.2.1 System models, notation, and differential geometry

We consider deterministic unknown continuous-time nonlinear systems $\dot{x} = h(x, u)$, where $h : \mathcal{X} \times \mathcal{U} \to \mathcal{X}$, $\mathcal{X} \subseteq \mathbb{R}^{n_x}$, and $\mathcal{U} \subseteq \mathbb{R}^{n_u}$. We define $g : \mathcal{X} \times \mathcal{U} \to \mathcal{X}$ to be a control-affine approximation of the true dynamics:

$$g(x, u) = f(x) + B(x)u. \tag{9.1}$$

While we do not assume that the true dynamics are control-affine, we do assume that they are locally incrementally exponentially stabilizable, that is, there exists a β, $\lambda > 0$, and feedback controller such that $\|x^*(t) - x(t)\| \leq \beta e^{-\lambda t}\|x^*(0) - x(0)\|$ for all solutions $x(t)$ in a domain. Many underactuated systems satisfy this condition (for a subset of their trajectories), and it is much weaker than requiring $n_x = n_u$, as in *Knuth et al.* (2021a). Also, this only needs to hold in a task-relevant domain D, defined later.

For a function η, a Lipschitz constant over a domain $\mathcal{Z}$ is any L such that for all $z_1, z_2 \in \mathcal{Z}$, $\|\eta(z_1) - \eta(z_2)\| \leq L\|z_1 - z_2\|$. Norms $\|\cdot\|$ are always the 2-norm. We define L_{h-g} as the smallest Lipschitz constant of the error $h - g$. The argument of $h - g$ is a state-control pair (x, u) and its value is a state. We define a ball $\mathcal{B}_r(x)$ as $\{y \mid \|y - x\| < r\}$, also referred to as a r-ball about x. We suppose the state space $\mathcal{X}$ is partitioned into safe $\mathcal{X}_{\text{safe}}$ and unsafe $\mathcal{X}_{\text{unsafe}}$ sets (e.g., collision states). We denote $\widehat{Q} \doteq Q + Q^\top$ as a symmetrization operation on matrix Q, and $\bar{\lambda}(\hat{Q})$ and $\underline{\lambda}(\hat{Q})$ as its maximum and minimum eigenvalues, respectively. We overload notation when $Q(x)$ is a matrix-valued function, denoting $\bar{\lambda}_{\mathcal{Q}}(Q) \doteq \sup_{x\in\mathcal{Q}} \bar{\lambda}(Q(x))$ and $\underline{\lambda}_{\mathcal{Q}}(Q) \doteq \inf_{x\in\mathcal{Q}} \underline{\lambda}(Q(x))$. Let $\mathbf{I}_n$ be the identity matrix of size $n \times n$. Let $\mathbb{S}_n^{>0}$ denote the set of symmetric, positive definite $n \times n$ matrices. Let the Lie derivative of a matrix-valued function $Q(x) \in \mathbb{R}^{n\times n}$ along a vector $y \in \mathbb{R}^n$ be denoted as $\partial_y Q(x) \doteq \sum_{i=1}^n y^i \frac{\partial Q}{\partial x^i}$. Let x^i denote the ith element of vector x. Let the notation $Q_\perp(x)$ refer to a basis for the null-space of matrix $Q(x)$.

Finally, we introduce the needed terminology from differential geometry. For a smooth manifold $\mathcal{X}$, a Riemannian metric tensor $M : \mathcal{X} \to \mathbb{S}_{n_x}^{>0}$ equips the tangent space $T_x\mathcal{X}$ at each element x with an inner product $\delta_x^\top M(x)\delta_x$, providing a local length measure. Then, the length $l(c)$ of a curve $c : [0, 1] \to \mathcal{X}$ between points $c(0)$, $c(1)$ can be computed by integrating the local lengths along the curve: $l(c) \doteq \int_0^1 \sqrt{V(c(s), c_s(s))}ds$, where for brevity $V(c(s), c_s(s)) \doteq c_s(s)^\top M(c(s))c_s(s)$, and $c_s(s) \doteq \partial c(s)/\partial s$. Then, the Riemann distance between two points $p, q \in \mathcal{X}$ can be defined as $\text{dist}(p, q) \doteq \inf_{c\in\mathcal{C}(p,q)} l(c)$, where $\mathcal{C}(p, q)$ is the set of all smooth curves connecting p and q. Finally, we define the Riemann energy between p and q as $\mathcal{E}(p, q) \doteq \text{dist}^2(p, q)$.

9.2.2 Control contraction metrics (CCMs)

Contraction theory studies how the distance between trajectories of a system changes with time to infer properties on incremental stability. This can be formalized

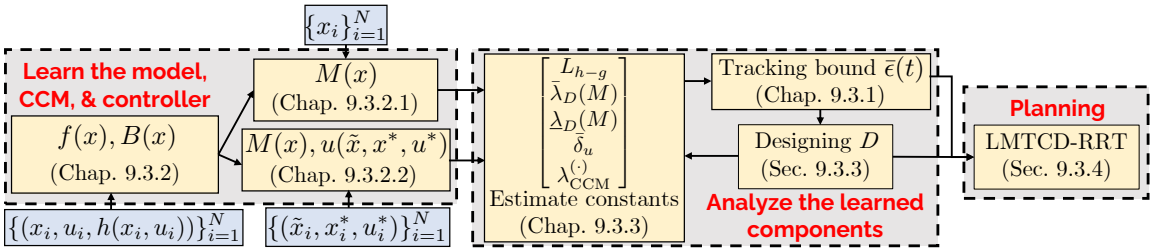


Figure 9.1: Method flowchart. **Left**: First, we learn a model of the dynamics using dataset $\mathcal{S}$ and obtain a contracting controller for this learned model (Prob. IX.1). **Center**: Next, within a trusted domain D, we verify (with a given probability) the correctness of the controller, bound the model error, and bound the trajectory tracking error under this model error (Prob. IX.2). **Right**: Finally, we use the error bounds to plan trajectories within D that can be safely tracked in execution (Prob. IX.3).

with a contraction metric $M(x) : \mathcal{X} \to \mathbb{S}_{n_x}^{>0}$ to measure if the differential distances between trajectories $V(x, \delta_x) = \delta_x^\top M(x)\delta_x$ shrink with time. Control contraction metrics (CCMs) adapt this analysis to control-affine systems (9.1). For dynamics of the form (9.1), the differential dynamics can be written as $\dot{\delta}_x = (\frac{\partial f}{\partial x} + \sum_{i=1}^{n_u} u^i \frac{\partial B^i}{\partial x})\delta_x + B(x)\delta_u$ *Singh et al.* (2019), where $B^i(x)$ is the ith column of $B(x)$. Then, we call $M(x) : \mathcal{X} \to \mathbb{S}_{n_x}^{>0}$ a CCM if there exists a differential controller δ_u such that the closed-loop system satisfies $\dot{V}(x, \delta_x) < 0$, for all x, δ_x.

How do we find a CCM $M(x)$ ensuring the existence of δ_u? First, define the dual metric $W(x) \doteq M^{-1}(x)$. Then, two sufficient conditions for contraction are (9.2)-(9.3) *Singh et al.* (2019); *Sun et al.* (2020):

$$B_\perp(x)^\top \Big(-\partial_f W(x) + \widehat{\frac{\partial f(x)}{\partial x} W(x)} + 2\lambda W(x) \Big) B_\perp(x) \preceq 0 \tag{9.2a}$$

$$B_\perp(x)^\top \Big(\partial_{B^j} W(x) - \widehat{\frac{\partial B^j(x)}{\partial x} W(x)} \Big) B_\perp(x) = 0, j = 1 \ldots n_u \tag{9.2b}$$

$$\dot{M}(x) + \widehat{M(x)(A(x) + B(x)K(\tilde{x}, x^*, u^*))} + 2\lambda M(x) \prec 0 \tag{9.3}$$

where $A \doteq \frac{\partial f}{\partial x} + \sum_{i=1}^{n_u} u^i \frac{\partial B^i}{\partial x}$ and $K = \frac{\partial u(\tilde{x}, x^*, u^*)}{\partial x}$, where $u : \mathcal{X} \times \mathcal{X} \times \mathcal{U} \to \mathcal{U}$ is a feedback controller which takes as input the tracking deviation $\tilde{x}(t) \doteq x(t) - x^*(t)$ from a nominal state $x^*(t)$, as well as a state/control $x^*(t)$, $u^*(t)$ on the nominal state/control trajectory that is being tracked $x^* : [0, T] \to \mathcal{X}$, $u^* : [0, T] \to \mathcal{U}$. We refer to the LHSs of (9.2a) and (9.3) as $C^s(x)$ and $C^w(\tilde{x}, x^*, u^*)$, respectively. Intuitively, (9.2a) is a contraction condition simplified by the orthogonality condition (9.2b), which together imply that all directions where the differential dynamics lack controllability must be naturally contracting at rate λ. The conditions (9.2) are stronger than (9.3), which does not make this orthogonality assumption.

How do we recover a tracking feedback controller $u(\tilde{x}, x^*, u^*)$ for (9.1) from (9.2) and (9.3)? For (9.2), the controller is implicit in the dual metric $W(x)$, and can be computed by solving a nonlinear optimization problem, which can be solved at runtime with pseudospectral methods *Leung and Manchester* (2017); *Singh et al.*

(2019). In (9.3), $u(\tilde{x}, x^*, u^*)$ is directly involved as the function defining K; as a consequence, $M(x)$ and $u(\tilde{x}, x^*, u^*)$ both need to be found. Thus for our purposes, the benefit of using (9.2) is that we have fewer parameters to learn. However, as some systems may not satisfy the properties needed to apply (9.2), we resort to using (9.3) in these cases (see Sec. 9.3.2.2). Finally, for a given CCM $M(x)$ and associated controller $u(\tilde{x}, x^*, u^*)$, for an unperturbed system tracking a nominal trajectory $x^*(t)$, the Riemannian energy $\mathcal{E}(x^*(t), x(t))$ satisfies $\|x(t) - x^*(t)\| \leq \beta \|x(0) - x^*(0)\| e^{-\lambda t}$ for an overshoot constant β, and thus the Euclidean distance also decays at this rate. If the system is subjected to bounded perturbations, it is instead guaranteed to remain in a tube around $x^*(t)$.

9.2.3 Problem statement

Our method has three major components. First, we learn a model (9.1), and then learn a contraction metric $M(x)$ and/or controller $u(\tilde{x}, x^*, u^*)$ for (9.1). Next, we analyze the learned (9.1), $M(x)$, and/or $u(\tilde{x}, x^*, u^*)$ to determine a trusted domain $D \subseteq \mathcal{X} \times \mathcal{U}$ where trajectories can be robustly tracked. Finally, we design a planner which computes a nominal state/control trajectory $x^*(t), u^*(t)$ (feasible for the *learned* dynamics), that steers between states in D, such that under the tracking controller $u(\tilde{x}, x^*, u^*)$, the system remains safe in execution and reaches the goal. In this chapter, we represent the approximate dynamics $g(x, u)$ with an NN, though our method is agnostic to the structure of the model and how it is derived. We note that due to this NN representation (where we use Lipschitz continuous activation functions), trajectories planned with the learned model will be continuous. Let $\mathcal{S} = \{(x_i, u_i, h(x_i, u_i))\}_{i=1}^N$ be the training data for g obtained by any means (i.e., random sampling, expert demonstrations, etc.), and let $\Psi = \{(x_j, u_j, h(x_j, u_j))\}_{j=1}^M$ be a set of independent and identically distributed (i.i.d.) samples collected near $\mathcal{S}$. Then, our method involves solving the following:

Problem IX.1 (Learning)**.** *Given $\mathcal{S}$, learn a control-affine model g, a contraction metric $M(x)$, and find a contraction-based controller $u(\tilde{x}, x^*, u^*)$ that satisfies* (9.2) *or* (9.3) *over $\mathcal{S}$.*

Problem IX.2 (Analysis)**.** *Given Ψ, g, $M(x)$, and $u(\tilde{x}, x^*, u^*)$, design a trusted domain D. In D, find a model error bound $\|h(x, u) - g(x, u)\| \leq e(x, u)$, for all $(x, u) \in D$, and verify, with high probability, if for all $x \in D$, M and u are valid, i.e., satisfying* (9.2)/(9.3)*.*

Problem IX.3 (Planning)**.** *Given g, $M(x)$, $u(\tilde{x}, x^*, u^*)$, start x_I, goal x_G, goal tolerance μ, maximum tracking error tolerance $\hat{\mu}$, trusted domain D, and $\mathcal{X}_{\text{safe}}$, plan a nominal trajectory $x^* : [0, T] \to \mathcal{X}$, $u^* : [0, T] \to \mathcal{U}$ under the learned dynamics g such that $x(0) = x_I$, $\dot{x} = g(x, u)$, $\|x(T) - x_G\| \leq \mu$, and $x(t)$, $u(t)$ remains in $D \cap \mathcal{X}_{\text{safe}}$ for all $t \in [0, T]$. Also, guarantee that in tracking $(x^*(t), u^*(t))$ under the true dynamics h with $u(\tilde{x}, x^*, u^*)$, the system remains in $D \cap \mathcal{X}_{\text{safe}}$ and reaches $\mathcal{B}_{\hat{\mu}+\mu}(x_G)$.*

9.3 Method

We first describe a spatially-varying, Lipschitz constant-based model error bound, and derive a trajectory tracking error bound for a CCM-based controller under this error description (Sec. 9.3.1). We then show how we can learn a dynamics model, CCM, and tracking controller to optimize the tracking error bound (Sec. 9.3.2). Then we show how we can design a trusted domain D and probabilistically verify the correctness of the model error bound, tracking error bound, and controller within D (Sec. 9.3.3). Finally, we show how this tracking bound can be embedded into a sampling-based planner to ensure that plans provably remain safe in execution and reach the goal (Sec. 9.3.4). We summarize our method in Fig. 9.1.

9.3.1 CCM-based tracking tubes under Lipschitz model error

We first establish a spatially-varying bound on model error within a trusted domain D which can be estimated from the model error evaluated at training points. For a single training point $(\bar{x}, \bar{u})$ and a novel point (x, u), we can bound the error between the true and learned dynamics at (x, u) using the triangle inequality and Lipschitz constant of the error L_{h-g}:

$$\begin{aligned} &\|h(x,u) - g(x,u)\| \\ &\quad \leq L_{h-g}\|(x,u) - (\bar{x},\bar{u})\| + \|h(\bar{x},\bar{u}) - g(\bar{x},\bar{u})\|. \end{aligned} \tag{9.4}$$

As this holds between the novel point and all training points, the following (possibly) tighter bound can be applied:

$$\begin{aligned} \|h(x,u) - g(x,u)\| \leq \min_{1\leq i\leq N} \Big\{ & L_{h-g}\|(x,u) - (x_i,u_i)\| \\ & + \|h(x_i,u_i) - g(x_i,u_i)\| \Big\}. \end{aligned} \tag{9.5}$$

To exploit higher model accuracy near the training data, we define D as the union of r-balls around $\mathcal{S}$, where $r < \infty$:

$$D = \bigcup_{i=1}^{N} \mathcal{B}_r(x_i, u_i). \tag{9.6}$$

We note that r should be chosen to be large enough, in order to make tracking tube containment checks feasible (described in Sec. 9.3.4). For these bounds to hold, L_{h-g} must be a valid Lipschitz constant over D. In Sec. 9.3.3, we discuss how to obtain a probabilistically-valid estimate of L_{h-g} and how to choose r. We now derive an upper bound $\bar{\epsilon}(t)$ on the Euclidean tracking error $\epsilon(t)$ around a nominal trajectory $(x^*(t), u^*(t)) \subseteq D$ for a given metric $M(x)$ and feedback controller $u(\tilde{x}, x^*, u^*)$, such that the executed and nominal trajectories $x(t)$ and $x^*(t)$ satisfy $\|x(t) - x^*(t)\| \leq \bar{\epsilon}(t)$, for all $t \in [0, T]$, when subject to the model error description (9.5). In Sec. 9.3.2, we discuss how M and u can be learned from data.

In *Singh et al.* (2019), it is shown that by using a controller which is contracting

with rate λ according to metric $M(x)$ for the *nominal* dynamics (9.1), the Riemannian energy $\mathcal{E}(t)$ of a *perturbed* control-affine system $\dot{x}(t) = f(x(t)) + B(x(t))u(t) + d(t)$ is bounded by the following differential inequality:

$$D^+\mathcal{E}(t) \leq -2\lambda\mathcal{E}(t) + 2\sqrt{\mathcal{E}(t)\bar{\lambda}_D(M)}\|d(t)\|, \tag{9.7}$$

where $\bar{\lambda}_D(M) = \sup_{x\in D} \bar{\lambda}(M(x))$ and $D^+(\cdot)$ is the upper Dini derivative of $(\cdot)$. Here, the energy $\mathcal{E}(t) = \mathcal{E}(x^*(t), x(t))$ is the squared trajectory tracking error according to the metric $M(x)$ at a given time t, and $d(t)$ is an external disturbance. Suppose that the only disturbance to the system comes from the discrepancy between the learned and true dynamics, i.e., $d(t) = h(x(t), u(t)) - g(x(t), u(t))$[1]. For short, let $e_i \doteq \|h(x_i, u_i) - g(x_i, u_i)\|$ be the training error of the ith data-point. In this case, we can use (9.5) to write:

$$\|d(t)\| \leq \min_{1\leq i\leq N} \left\{ L_{h-g} \left\| \begin{bmatrix} x(t) \\ u(t) \end{bmatrix} - \begin{bmatrix} x_i \\ u_i \end{bmatrix} \right\| + e_i \right\}. \tag{9.8}$$

As (9.8) is spatially-varying, it suggests that in solving Prob. IX.3, plans should stay near low-error regions to encourage low error in execution. However, (9.8) is only implicit in the plan, depending on the state visited and feedback control applied *in execution*: $x(t) = x^*(t) + \tilde{x}(t)$ and $u(\tilde{x}(t), x^*(t), u^*(t)) = u^*(t) + u_{\text{fb}}(t)$. To derive a tracking bound that can directly inform planning, we first introduce the following lemma:

Lemma IX.4. *The Riemannian energy $\mathcal{E}(t)$ of the perturbed system $\dot{x}(t) = f(x(t)) + B(x(t))u(t) + d(t)$, where $\|d(t)\|$ satisfies* (9.8)*, satisfies the differential inequality* (9.11)*, where $\underline{\lambda}_D(M) = \inf_{x\in D} \underline{\lambda}(M(x))$, $\bar{u}_{\text{fb}}(t)$ is a time-varying upper bound on the feedback control $\|u(t) - u^*(t)\|$, and $i^*(t)$ achieves the minimum in* (9.8)*.*

Proof sketch. We use the triangle inequality to simplify (9.8):

$$\begin{aligned} \|d(t)\| &\leq \min_{1\leq i\leq N} \left\{ L_{h-g} \left(\left\| \begin{bmatrix} x^*(t) \\ u^*(t) \end{bmatrix} - \begin{bmatrix} x_i \\ u_i \end{bmatrix} \right\| + \left\| \begin{bmatrix} \tilde{x}(t) \\ u_{\text{fb}}(t) \end{bmatrix} \right\| \right) + e_i \right\} \\ &\leq L_{h-g} \left\| \begin{bmatrix} \tilde{x}(t) \\ u_{\text{fb}}(t) \end{bmatrix} \right\| + \min_{1\leq i\leq N} \left\{ L_{h-g} \left\| \begin{bmatrix} x^*(t) \\ u^*(t) \end{bmatrix} - \begin{bmatrix} x_i \\ u_i \end{bmatrix} \right\| + e_i \right\}. \end{aligned}$$

Note that as $\|d(t)\|$ depends on $\tilde{x}(t)$, the disturbance bound itself depends on $\epsilon(t)$. To make this explicit, we use $\|\tilde{x}(t)\| = \epsilon(t)$ and $\|u_{\text{fb}}(t)\| \leq \bar{u}_{\text{fb}}(t)$ to obtain

$$\begin{aligned} \|d(t)\| \leq \;& L_{h-g}\big(\epsilon(t) + \bar{u}_{\text{fb}}(t)\big) + \\ & \min_{1\leq i\leq N} \left\{ L_{h-g} \left\| \begin{bmatrix} x^*(t) \\ u^*(t) \end{bmatrix} - \begin{bmatrix} x_i \\ u_i \end{bmatrix} \right\| + e_i \right\}. \end{aligned} \tag{9.9}$$

[1] In addition to model error, we can also handle *runtime* external disturbances with a known upper bound; we assume the training data is noiseless.

$$D^+\mathcal{E}(t) \leq -2\left(\lambda - L_{h-g}\sqrt{\frac{\bar{\lambda}_D(M)}{\underline{\lambda}_D(M)}}\right)\mathcal{E}(t) + 2\sqrt{\mathcal{E}(t)\bar{\lambda}_D(M)}\left(L_{h-g}\left(\left\| \begin{bmatrix} x^*(t) \\ u^*(t) \end{bmatrix} - \begin{bmatrix} x_{i^*(t)} \\ u_{i^*(t)} \end{bmatrix} \right\| + \bar{u}_{\text{fb}}(t)\right) + e_{i^*(t)}\right) \tag{9.11}$$

To obtain $\bar{u}_{\text{fb}}(t)$, if we use CCM conditions (9.2), we can use the optimization-based controller in *Singh et al.* (2019) (cf. Sec. 9.2.2), which admits the upper bound (*Singh et al.*, 2019, p.28):

$$\|u_{\text{fb}}(t)\| \leq \epsilon(t) \sup_{x \in D} \frac{\bar{\lambda}(L(x)^{-\top} F(x) L(x)^{-1})}{2\underline{\sigma}_{>0}(B^\top(x) L(x)^{-1})} \doteq \epsilon(t)\delta_u, \tag{9.10}$$

where $W(x) = L(x)^\top L(x)$, $F(x) = -\partial_f W(x) + \widehat{\frac{\partial f(x)}{\partial x} W(x)} + 2\lambda W(x)$, and $\underline{\sigma}_{>0}(\cdot)$ is the smallest positive singular value. If we instead use condition (9.3), we must estimate $\bar{u}_{\text{fb}}(t)$ for the learned controller (cf. Sec. 9.3.3).

To obtain the result, we plug (9.9) into (9.7) after relating $\epsilon(t)$ with $\mathcal{E}(t)$. Since $\mathcal{E}(x^*(t), x(t)) = \text{dist}^2(x^*(t), x(t)) \geq \underline{\lambda}_D(M)\|x^*(t) - x(t)\|^2$, we have that $\epsilon(t) \leq \sqrt{\mathcal{E}(t)/\underline{\lambda}_D(M)}$. Finally, we can plug all of these components into (9.7) to obtain (9.11), where $i^*(t)$ denotes a minimizer of (9.8). □

For intuition, let us interpret the spatially-varying disturbance bound (9.9). Note that (9.9) depends on several components. First, it depends on $\epsilon(t)$, which in turn relies on the disturbance magnitude: intuitively, this is because with better tracking performance, the system will visit a smaller set of states and thus experience lower worst-case model error. Second, it depends on $u_{\text{fb}}(t)$: if a large feedback is applied, the combined control input $u(t) = u^*(t) + u_{\text{fb}}(t)$ can be far from the control inputs that the learned model is trained on, possibly leading to high error. Finally, it is driven by the model error via closeness to the training data and the corresponding training error (minimization term of (9.9)). We can also gain some insight by comparing our derived tracking error bound (9.11) with the tracking bound for a uniform disturbance description (9.7). Notice that the "effective" contraction rate $\lambda - L_{h-g}\sqrt{\frac{\bar{\lambda}_D(M)}{\underline{\lambda}_D(M)}}$ shrinks with L_{h-g}, as the model error grows with tracking error. If the optimization-based controller *Singh et al.* (2019) is used, the $\epsilon(t)$ dependence of (9.10) reduces this rate to $\lambda - L_{h-g}\sqrt{\frac{\bar{\lambda}_D(M)}{\underline{\lambda}_D(M)}}(1 + \delta_u)$. Now, we are ready to obtain the tracking bound:

Theorem IX.5 (Tracking bound under (9.8))**.** *Let $\mathcal{E}_{RHS}$ denote the RHS of* (9.11)*. Assuming that the perturbed system $\dot{x}(t) = f(x(t)) + B(x(t))u(t) + d(t)$ satisfies $\mathcal{E}(t_1) \leq \mathcal{E}_{t_1}$ and $\|d(t)\|$ satisfies* (9.8)*. Then, $\bar{\epsilon}(t)$ is described at some $t_2 > t_1$ as:*

$$\bar{\epsilon}(t_2) = \sqrt{\left(\int_{\tau=t_1}^{t_2} \mathcal{E}_{\text{RHS}}(t)d\tau\right)/\underline{\lambda}_D(M)}, \quad \mathcal{E}(t_1) = \mathcal{E}_{t_1}. \tag{9.12}$$

Proof sketch. We apply the Comparison Lemma *Khalil* (2002) on (9.11). To use the Comparison Lemma, the right hand side of (9.11) must be Lipschitz continuous in $\mathcal{E}$ and continuous in t *Khalil* (2002). Lipschitz continuity in $\mathcal{E}$ holds for (9.11) since it only contains a linear and a square-root term involving $\mathcal{E}$, each with finite

coefficients; continuity in t follows by noting that the minimization in (9.9) is the pointwise minimum of N continuous functions of t, and thus is itself a continuous function of t. □

Note that Thm. IX.5 provides a Euclidean tracking error tube under the model error bound (9.8) that can be evaluated for *any* nominal trajectory computed with the learned dynamics. Note that at the cost of additional conservativeness, we have removed the explicit dependence on $u_{\text{fb}}(t)$ by replacing it with the upper bound $\bar{u}_{\text{fb}}(t)$. Moreover, as (9.12) can be integrated incrementally in time, it is well-suited to guide planning in an RRT (Rapidly-exploring Random Tree *LaValle and James J. Kuffner* (2001)); see Sec. 9.3.4 for more details.

9.3.2 Optimizing CCMs and controllers for the learned model

Having derived the tracking error bound, we discuss our solution to Prob. IX.1, i.e., how we learn the control-affine dynamics (9.1), a contraction metric $M(x)$, and (possibly) a stabilizing controller u in a way that optimizes the size of (9.12). In this chapter, we represent $f(x)$, $B(x)$, $M(x)$, and $u(\tilde{x}, x^*, u^*)$ as deep neural networks.

Ideally, we would learn the dynamics jointly with the contraction metric to minimize the size of the tracking tubes (9.12). However, we observe this leads to poor learning, generally converging to a valid CCM for highly inaccurate dynamics. Instead, we elect to use a simple two step procedure: we first learn g, and then fix g and learn $M(x)$ and $u(\tilde{x}, x^*, u^*)$ for that model. While this is sufficient for our examples, in general an alternation procedure may be helpful. Finally, we note that this learning procedure is not complete; even if a valid CCM and controller exist (within the chosen NN architectures) for the learned dynamics, the learning procedure is prone to local optima that can prevent convergence to that valid CCM and controller; this is in contrast to prior CCM synthesis methods *Manchester and Slotine* (2017); *Singh et al.* (2019) for polynomial systems that rely on tools like SoS programming.

Dynamics learning. Inspecting (9.11), we note that the model-error related terms are the Lipschitz constant L_{h-g} and training error e_i, $i = 1, \ldots, N$. Thus, in training the dynamics, we use a loss function penalizing the mean squared error and a batch-wise estimate of the Lipschitz constant:

$$L_{\text{dyn}} = \frac{1}{N_b} \sum_{i=1}^{N_b} e_i^2 + \alpha_1 \max_{1 \le i,j \le N_b} \left\{ \frac{\|e_i - e_j\|}{\|(x_i, u_i) - (x_j, u_j)\|} \right\}, \tag{9.13}$$

where $e_i = \|g(x_i, u_i) - h(x_i, u_i)\|$, $N_b \le N$ is the batch size, and α_1 trades off the objectives. Note that (9.13) promotes e_i to be small while remaining smooth over the training data, in order to encourage similar properties to hold over D. We also note that while there are existing approaches for providing trainable approximations for neural network Lipschitz constants *Huang et al.* (2021); *Pauli et al.* (2022), we apply the batch-wise estimate because we require the Lipschitz constant of the *error*, not the network itself.

CCM learning. We describe two variants of our learning approach, depending on if the stronger CCM conditions (9.2a) and (9.2b) or the weaker condition (9.3) is used.

9.3.2.1 Using (9.2a) and (9.2b)

We parameterize the dual metric as $W(x) = W_{\theta_w}(x)^\top W_{\theta_w}(x) + \underline{w}\mathbf{I}_{n\times n}$, where $W_{\theta_w}(x) \in \mathbb{R}^{n_x \times n_x}$, θ_w are the NN weights, and $\underline{w}$ is a minimum eigenvalue hyperparameter. This structure ensures that $W(x)$ is symmetric positive definite for all x. To enforce (9.2a), we follow *Sun et al.* (2020), relaxing the matrix inequality to an unconstrained penalty L^s_{NSD} over training data, where:

$$L^{(\cdot)}_{\text{NSD}} = \max_{1\le i\le N_b} \bar{\lambda}\big(C^{(\cdot)}(x_i)\big). \tag{9.14}$$

As we ultimately wish (9.2a) to hold everywhere in D, we can use the continuity in x of the maximum eigenvalue $\bar{\lambda}(\lambda^s(x))$ to verify (probabilistically) if (9.2a) holds over D (cf. Sec. 9.3.3). However, the equality constraints (9.2b) are problematic; by using unconstrained optimization, it is difficult to even satisfy (9.2b) on the training data, let alone on D. To address this, we follow *Singh et al.* (2020) by restricting the dynamics learning to sparse-structured $B(x)$ of the form, where θ_B are NN parameters:

$$B(x) = [\mathbf{0}^\top_{n_x - n_u \times n_u}, \quad B_{\theta_B}(x)^\top]^\top. \tag{9.15}$$

Restricting $B(x)$ to this form implies that to satisfy (9.2b), $W(x)$ must be a function of only the first $n_x - n_u$ states *Singh et al.* (2020), which can be satisfied by construction. When this structural assumption does not hold, we use the method in Sec. 9.3.2.2

In addition to the CCM feasibility conditions, we introduce novel losses to optimize the tracking tube size (9.12). As (9.12) depends on the nominal trajectory, it is hard to optimize a tight upper bound on the tracking error independent of the plan. Instead, we maximize the effective contraction rate,

$$L^s_{\text{opt}} = \alpha_2 \max_{1\le i\le N_b} \Big(\lambda - L_{h-g}\sqrt{\frac{\bar{\lambda}(M(x_i))}{\underline{\lambda}(M(x_i))}}(1+\delta_u(\tilde{x}_i))\Big), \tag{9.16}$$

where α_2 is a tuned parameter. Optimizing (9.16) while ensuring (9.2a) holds over the dataset is a challenging task for unconstrained NN optimizers. To ameliorate this, we use a linear penalty on constraint violation and switch to a logarithmic barrier *Keshavarz et al.* (2011) to maintain feasibility upon achieving it. Let the combination of the logarithmic barrier and the linear penalty be denoted $\texttt{logb}(\cdot)$. Then, the full loss function can be written as $\texttt{logb}(-L^s_{\text{NSD}}) + L^s_{\text{opt}}$.

9.3.2.2 Using (9.3)

For systems that do not satisfy (9.15), we must use the weaker contraction conditions (9.3). In this case, we cannot use the optimization-based controllers proposed in *Singh et al.* (2019), and we instead learn $u(\tilde{x}, x^*, u^*)$ in tandem with $M(x)$. As in (9.14), we enforce (9.3) by relaxing it to L^w_{NSD}. We represent $u(\tilde{x}, x^*, u^*)$ with the following structure:

$$u(\tilde{x}, x^*, u^*) = |\theta^u_1| \tanh\big(u_{\theta^u_2}(\tilde{x}, x^*)\tilde{x}\big) + u^*, \tag{9.17}$$

where θ_i^u are NN weights. The structure of (9.17) simplifies $\bar{u}_{\text{fb}}$ estimation, since $\|u(\tilde{x}, x^*, u^*) - u^*\| < |\theta_1^u|$ for all x, x^*, u^*. We define L^w_{opt} as in (9.16), without the δ_u term. Then, our full loss function is $\texttt{logb}(-L^w_{\text{NSD}}) + L^w_{\text{opt}} + \alpha_3|\theta_1^u|$.

We note that since the optimizer can reach a local minima, we may not find a valid CCM even if one exists. Some strategies we found to improve training reliability were the log-barrier and by gradually increasing α_2 and α_3 with the training epoch. If we can find a valid CCM on $\mathcal{S}$, we can verify in probability if it is also valid over D, as we now discuss.

9.3.3 Designing and probabilistically verifying the trusted domain

The validity of the tracking bound (9.12) depends on having overestimates of L_{h-g}, $\bar{\lambda}_D(M)$, and δ_u, an underestimate of $\underline{\lambda}_D(M)$, and the validity of (9.2a)/(9.3) over D. In this section, we describe our solution to Prob. IX.2, showing how to design a D and how to estimate these constants over D.

First, let us consider how to estimate the constants over a given D. To obtain probabilistic over/under-estimates of these constants that are each valid with a user-defined probability ρ, we use a stochastic approach from extreme value theory. For brevity, we describe the basics and refer to (*Knuth et al.*, 2021a, p.3), *Weng et al.* (2018) for details. This approach estimates the maximum of a function $\eta(z)$ over a domain $\mathcal{Z}$ by collecting N_s batches of i.i.d. samples of $z \in \mathcal{Z}$ of size N_b, and evaluating $\{s_j\}_{j=1}^{N_s} \doteq \{\max_{1\le i\le N_b} \eta(z_i^j)\}_{j=1}^{N_s}$ to obtain N_s samples of empirical maxima. If the true maximum is finite and the distribution of sampled s_j converges with increasing N_s, the Fisher-Tippett-Gnedenko (FTG) theorem *De Haan and Ferreira* (2007) dictates that the samples converge to a Weibull distribution. This can be empirically verified by fitting a Weibull distribution to the s_j, and validating the quality of the fit with a Kolmogorov-Smirnov (KS) goodness-of-fit test *DeGroot and Schervish* (2013). If this KS test passes, the location parameter of the fit Weibull distribution, adjusted with a confidence interval which scales in size with the value of the user-defined probability ρ, can serve as an estimate for the maximum which overestimates the true maximum with probability ρ. Finally, we note (as in Chapter VIII) that this probability holds in the limit of infinite samples N_s, since the FTG theorem only makes claims on asymptotic convergence to the Weibull distribution.

To estimate L_{h-g}, we follow *Knuth et al.* (2021a) to obtain a probabilistic overestimate of L_{h-g} by defining $\mathcal{Z} = D$ and η to be the slopes between pairs of points drawn i.i.d. from D. This approach can also be used as follows to estimate $\bar{\lambda}_D(M)$, $-\underline{\lambda}_D(M)$, and δ_u. Since the eigenvalues of a continuously parameterized matrix function are continuous in the parameter *Li and Zhang* (2019) (here, the parameter is x, since M is a function of x) and D is bounded, these constants are finite, so by the FTG theorem, we can expect the samples s_j to be Weibull. Hence, we can estimate these constants by defining $\mathcal{Z} = \text{proj}_x(D)$, where $\text{proj}_x(D) \doteq \bigcup_{\bar{x}\in\mathcal{S}} \mathcal{B}_r(\bar{x}) \supset \{x \mid \exists u, (x,u) \in D\}$, and by setting η appropriately for each constant. Finally, FTG can also probabilistically verify (9.2a) and (9.3), since the verification is equivalent to ensuring $\sup_{x\in\text{proj}_x(D)} \bar{\lambda}(C^{(\cdot)}(x)) \le \lambda^{(\cdot)}_{\text{CCM}}$ for some $\lambda^{(\cdot)}_{\text{CCM}} < 0$. λ^s_{CCM} can be estimated by setting $\mathcal{Z} = \text{proj}_x(D)$ and $\eta(x) = \bar{\lambda}(C^s(x))$. To estimate λ^w_{CCM}, we set $\mathcal{Z} = \mathcal{B}_{\epsilon_{\max}}(0) \times D$

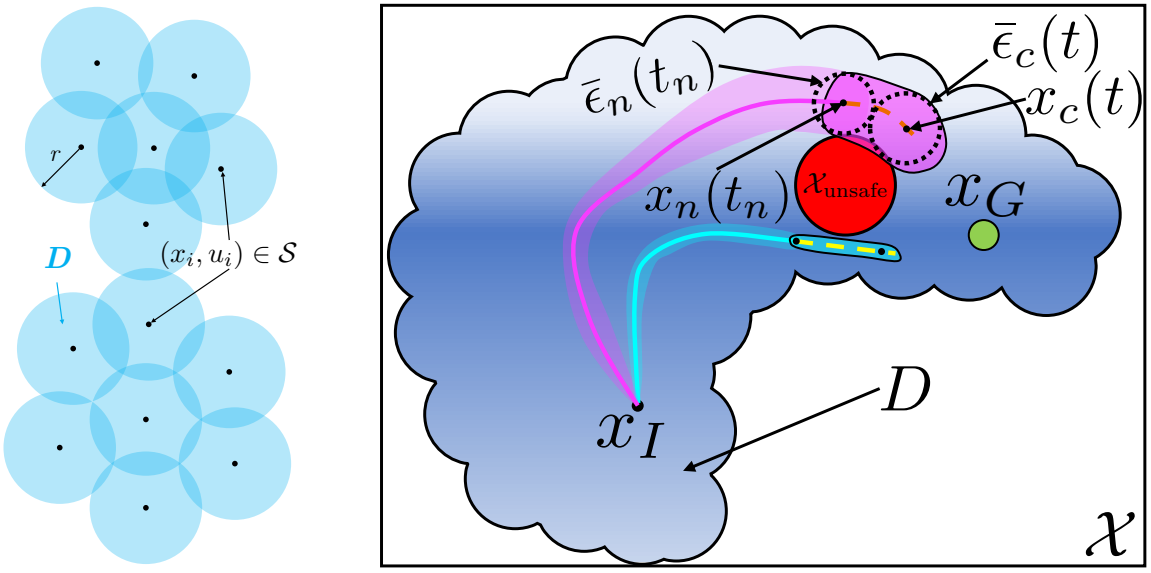


Figure 9.2: **Left**: an example of the trusted domain D, with the dataset $\mathcal{S}$ being shown as black dots. Note that a careful choice of r is needed; for a slightly smaller r than that shown in the figure, the upper and lower portions of D will become disconnected, leading to plan infeasibility. **Right**: An example of LMTCD-RRT in action. Regions in D that are shaded darker blue have smaller model error $\|d\|$; lighter shades have higher error $\|d\|$. Note that D is also a union of balls here; we have suppressed the boundaries of each individual ball to reduce clutter. The orange extension to the pink branch of the RRT is rejected, since the tube around that extension (dark magenta) exits D and intersects with the obstacle; the larger size of this tube results from the pink branch traveling through higher error regions. In contrast, the cyan branch (lower) accepts the yellow candidate extension, as its corresponding tube (dark cyan) remains inside D and collision-free; the smaller tube sizes reflect that the blue branch has traveled through lower-error regions. This type of behavior biases the planner to ultimately return a path that travels through lower-error regions.

and $\eta(\tilde{x}, x^*, u^*) = \bar{\lambda}(C^w(\tilde{x}, x^*, u^*))$, and sample $(x^*, u^*) \in D$ and $\tilde{x} \in \mathcal{B}_{\epsilon_{\max}}(0)$. Here, $\epsilon_{\max} \leq \hat{\mu}$ will upper-bound the allowable tracking tube size during planning (cf. Alg. IX.1, line 8); thus, to ensure that planning is minimally constrained, $\epsilon_{\max}$ should be selected to be as large as possible while maintaining $\lambda^w_{\text{CCM}} < 0$. As all samples are i.i.d., the probability of (9.12) holding, and thus the overall safety probability assured by our method, is the product of the user-selected ρ for each constants.

Before moving on, we note that other than for L_{h-g}, the estimation procedure does not affect data-efficiency, as it queries the *learned dynamics* and requires no new data of the form $(x, u, h(x, u))$. Moreover, some methods *Fazlyab et al.* (2019); *Jordan and Dimakis* (2020) deterministically give guaranteed upper bounds on the Lipschitz constant of NNs, and can be used to estimate all constants except L_{h-g}. We do not use these methods due to their scalability issues, but as further work is done in this area, these methods may also become applicable.

Finally, we discuss how to select r, which determines D (Fig. 9.2, left). A rea-

sonable choice of r is one that is maximally permissive for planning, during which we will need to ensure that the tracking tube around the planned trajectory remains entirely in D (cf. Sec. 9.3.4 for more details). However, finding this r is non-trivial to achieve and requires trading off many factors. For large r, planning may become easier since this increases the size of D; however, model error and L_{h-g} also degrades with increased r, which may make $\bar{\epsilon}(t)$ and $\bar{u}_{\text{fb}}(t)$ grow, which in turn grows the tracking tube size, making it harder to fit the tube within D. Also, (9.2a)/(9.3) may not be satisfied over D for large r. For small r, the model error and L_{h-g} remain smaller due to the closeness to $\mathcal{S}$, leading to smaller tubes, but planning can be challenging, as D may be too small to contain even these smaller tubes. In particular, planning between two states in D can become infeasible if D becomes disconnected.

Algorithm IX.1: LMTCD-RRT

Input: x_I, x_G, $\mathcal{S}$, $\{e_i\}_{i=1}^N$, estimated constants, μ, $\mathcal{E}_0$

$\mathcal{T} \leftarrow \{(x_I, \sqrt{\mathcal{E}_0/\underline{\lambda}_D(M)}, 0)\}; \mathcal{P} \leftarrow \{(\emptyset, \emptyset)\}$ `// node: state, energy, time; parent: previous control/dwell time`

while True **do**

  $(x_{\text{n}}, \bar{\epsilon}_{\text{n}}, t_{\text{n}}) \leftarrow$ `SampleNode(`$\mathcal{T}$`)` `// sample a node from the tree for expansion`

  $(u_{\text{c}}, t_{\text{c}}) \leftarrow$ `SampleCandidateControl ()` `// sample a control action and dwell time`

  $(x_{\text{c}}^*(t), u_{\text{c}}^*(t)) \leftarrow$ `IntegrateLearnedDyn` $(x_{\text{n}}, u_{\text{c}}, t_{\text{c}})$ `// apply control for dwell time; get candidate tree extension`

  $\bar{\epsilon}_c(t) \leftarrow$ `TrkErrBndEq12` $(\bar{\epsilon}_{\text{n}}, x_{\text{c}}^*(t), u_{\text{c}}^*(t), \mathcal{S}, \{e_i\}_{i=1}^N)$ `// compute tracking error tube for candidate tree extension`

  $D_{\text{chk}}^1 \leftarrow (x^*(t), u^*(t)) \in D_{\bar{\epsilon}_c(t) - \bar{u}_{\text{fb}}(t)}, \forall t \in [t_{\text{n}}, t_{\text{n}} + t_{\text{c}})$ `// check if tube around cand. extension remains within` D

  if *controller learned using* (9.3) **then** $D_{\text{chk}}^2 \leftarrow \bar{\epsilon}_c(t) \leq \epsilon_{\max}, \forall t \in [t_{\text{n}}, t_{\text{n}} + t_{\text{c}})$

  else $D_{\text{chk}}^2 \leftarrow$ True `// if using a controller satisfying (9.3), check if accumulated tracking error is below tolerance`

  $C \leftarrow$ `InCollision` $(x^*(t), u^*(t), \bar{\epsilon}_c(t))$ `// check if tracking error tube collides with obstacles`

  if $D_{\text{chk}}^1 \wedge D_{\text{chk}}^2 \wedge \neg C$ **then** $\mathcal{T} \leftarrow \mathcal{T} \cup \{(x^*(t_{\text{n}} + t_{\text{c}}), \bar{\epsilon}_c(t_{\text{n}} + t_{\text{c}}), t_{\text{c}})\}$; $\mathcal{P} \leftarrow \mathcal{P} \cup \{(u_{\text{c}}, t_{\text{c}})\}$

  else continue `// add node and corresponding parent if all checks pass`

  if $\exists t, x_c^*(t) \in \mathcal{B}_\mu(x_G)$ **then** break; return plan `// return path upon reaching goal`

To trade off these competing factors, we propose the following solution for selecting r. We first find a minimum r, r_{connect}, such that D has a single connected component. Depending on how $\mathcal{S}$ is collected, one may wish to first filter out outliers that lie far from the bulk of the data. We calculate the connected component by considering the dataset as a graph, where an edge between $(x_i, u_i), (x_j, u_j) \in \mathcal{S}$ exists if $\|(x_i, u_i) - (x_j, u_j)\| \leq r$. We then determine if the contraction condition (9.2a)/(9.3) is satisfied

for $r = r_{\text{connect}}$, using the FTG-based procedure. If it is not satisfied, we decrement r until (9.2a)/(9.3) holds, and select r as the largest value for which (9.2a)/(9.3) are satisfied. Since $r < r_{\text{connect}}$ in this case, planning can only be feasible between start and goal states within each connected component; to rectify this, more data should be collected to train the CCM/controller. If the contraction condition is satisfied at $r = r_{\text{connect}}$, we incrementally increase r, starting from r_{connect}. In each iteration, we first determine if the contraction condition (9.2a)/(9.3) is still satisfied for the current r, using the FTG-based procedure. If the contraction condition is satisfied, we evaluate an approximate measure of planning permissiveness under "worst-case" conditions[2]: $r - \bar{\epsilon}(t) - \bar{u}_{\text{fb}}(t)$, evaluated at a fixed time $t = T_{\text{query}}$, where $\bar{\epsilon}(t)$ and $\bar{u}_{\text{fb}}(t)$ are computed assuming that for all $t \in [0, T_{\text{query}}]$, $\|(x^*(t), u^*(t)) - (x_{i^*(t)}, u_{i^*(t)}))\| = \max_{1 \le i \le N} \min_{1 \le j \le N} \|(x_i, u_i) - (x_j, u_j)\|$, i.e., the dispersion of the training data, and experiences the worst training error (i.e., $e_{i^*(t)} = \max_{1 \le i \le N} e_i$, for all t). If the contraction condition is not satisfied, we terminate the search and select the r with the highest permissiveness, as measured by the aforementioned procedure.

9.3.4 Planning with the learned model and metric

Finally, we discuss our solution to safely planning with the learned dynamics (Prob. IX.3). We develop an incremental sampling-based planner akin to a kinodynamic RRT *LaValle and James J. Kuffner* (2001), growing a search tree $\mathcal{T}$ by forward-propagating sampled controls held for sampled dwell-times, until the goal is reached. To ensure the system remains within D in execution (where the contraction condition and (9.12) are valid), we impose additional constraints on where $\mathcal{T}$ is allowed to grow.

We denote (with a subscript) $D_q = D \ominus \mathcal{B}_q(0)$ as the state/controls which are at least distance q from the complement of D, where $\ominus$ refers to the Minkowski difference. Since (9.12) defines tracking error tubes for *any* given nominal trajectory, we can efficiently compute tracking tubes along any candidate edge of an RRT. Specifically, suppose that we wish to extend the RRT from a state on the planning tree $x^*_{\text{cand}}(t_1)$ with initial energy satisfying $\mathcal{E}_{\text{cand}}(t_1) \le \mathcal{E}_{t_1}$ to a candidate state $x^*_{\text{cand}}(t_2)$ by applying control u^*_{cand} over $[t_1, t_2)$. This information is supplied to (9.12), and we can obtain the tracking error $\bar{\epsilon}_{\text{cand}}(t)$, for all $t \in [t_1, t_2)$. Then, if we enforce that $(x^*(t), u^*(t)) \in D_{\bar{\epsilon}_{\text{cand}}(t) + \bar{u}_{\text{fb}}(t)}$ for all $t \in [t_1, t_2)$, we can ensure that the true system remains within D when tracked with a controller that satisfies $u_{\text{fb}}(t) \le \bar{u}_{\text{fb}}(t)$ in execution. Otherwise, the extension is rejected and the sampling continues. When using a learned $u(\tilde{x}, x^*, u^*)$ with (9.3), an extra check that $\bar{\epsilon}_{\text{cand}}(t) \le \epsilon_{\max}$ is needed to remain in $\mathcal{B}_{\epsilon_{\max}}(0) \times D$ (cf. Sec. 9.3.3). Since D is a union of balls, exactly checking $(x^*(t), u^*(t)) \in D_{\bar{\epsilon}(t) + \bar{u}_{\text{fb}}(t)}$ can be unwieldy. However, a conservative check can be efficiently performed by evaluating (9.18):

Theorem IX.6. *If* (9.18) *holds for some index* $1 \le i \le N$ *in* $\mathcal{S}$*,*

$$\|(x^*(t), u^*(t)) - (x_i, u_i)\| \le r - \bar{\epsilon}(t) - \bar{u}_{\text{fb}}(t), \tag{9.18}$$

[2]Roughly, this compares the size of D to the tracking error tube size and feedback control bound, cf. Sec. 9.3.4 and Thm. IX.6 for further justification.

then $(x^*(t), u^*(t)) \in D_{\bar{\epsilon}(t)+\bar{u}_{\text{fb}}(t)}$.

Proof. By (9.18), all $(\bar{x}, \bar{u}) \in \mathcal{B}_{\bar{\epsilon}(t)+\bar{u}_{\text{fb}}(t)}(x^*(t), u^*(t))$ satisfy $\|(x_i, u_i) - (\bar{x}, \bar{u})\| \leq r$ by the triangle inequality. Thus, $\mathcal{B}_{\bar{\epsilon}(t)+\bar{u}_{\text{fb}}(t)}(x^*(t), u^*(t)) \subset \mathcal{B}_r(x_i, u_i) \subset D$. As $\mathcal{B}_{\bar{\epsilon}(t)+\bar{u}_{\text{fb}}(t)}(x^*(t), u^*(t)) \subset D$, $(x^*(t), u^*(t))$ is at least $\bar{\epsilon}(t) + \bar{u}_{\text{fb}}(t)$ distance from the complement of D; thus, $(x^*(t), u^*(t)) \in D_{\bar{\epsilon}(t)+\bar{u}_{\text{fb}}(t)}$. □

We note that it can be quite conservative to rely on containment within a single r-ball; this can be mitigated by a pre-processing step where larger balls are "carved" out of unions of intersecting r-balls and are also used in the containment check; this can be potentially done with a method like *Deits and Tedrake* (2014). We perform collision checking between the tracking tubes and the obstacles, which we assume are expanded for the robot geometry; this is made easier since (9.12) defines a sphere for all time instants. We visualize our planner (Fig. 9.2, right), which we denote **L**earned **M**odels in **T**rusted **C**ontracting **D**omains (LMTCD-RRT), and summarize it in Alg. IX.1. We conclude with the following correctness result:

Theorem IX.7 (LMTCD-RRT correctness)**.** *Assume that the estimated* L_{f-g}, $\bar{\lambda}_D(M)$, $\bar{u}_{\text{fb}}(t)$, *and* $\lambda^{(\cdot)}_{\text{CCM}}$ *overapproximate their true values and the estimated* $\underline{\lambda}_D(M)$ *underapproximates its true value. Then, when using a controller* $u(\tilde{x}, x^*, u^*)$ *derived from* (9.2)*, Alg. IX.1 returns a trajectory* $(x^*(t), u^*(t))$ *that remains within* D *in execution on the true system. Moreover, when using a controller* $u(\tilde{x}, x^*, u^*)$ *derived from* (9.3)*, Alg. IX.1 returns a trajectory* $(x^*(t), u^*(t))$ *such that* $(\tilde{x}^*(t), x^*(t), u^*(t))$ *remains in* $\mathcal{B}_{\epsilon_{\max}}(0) \times D$ *in execution on the true system.*

Proof. First, we consider a controller that is derived from (9.2). By Thm. IX.6, Alg. IX.1 returns a plan $(x^*(t), u^*(t))$ such that for all $t \in [0, T]$, $(x^*(t), u^*(t)) \in D_{\epsilon(t)+\bar{u}_{\text{fb}}(t)}$. If the constants are estimated correctly, Thm. IX.5 holds, ensuring that the tracking error $\epsilon(t)$ in execution is less than $\bar{\epsilon}(t)$, for all $t \in [0, T]$. Furthermore, by the correct estimation of $\bar{u}_{\text{fb}}(t)$, the feedback control satisfies $\|u_{\text{fb}}(t)\| \leq \bar{u}_{\text{fb}}(t)$. Thus, for all $t \in [0, T]$, the state/control in execution $(x(t), u(t))$ remains in $\mathcal{B}_{\epsilon(t)+\bar{u}_{\text{fb}}(t)}(x^*(t), u^*(t))$. As $(x^*(t), u^*(t)) \in D_{\epsilon(t)+\bar{u}_{\text{fb}}(t)}$, $(x(t), u(t)) \in D$, for all $t \in [0, T]$. To apply this result for the learned NN controller derived using (9.3), we note that Alg. IX.1 further ensures that $\bar{\epsilon}(t) \leq \epsilon_{\max}$, for all $t \in [0, T]$. As the preceding discussion shows that $\epsilon(t) \leq \bar{\epsilon}(t)$ in execution, we have that $\epsilon(t) \leq \epsilon_{\max}$; therefore, $(\tilde{x}^*(t), x^*(t), u^*(t))$ remains within $\mathcal{B}_{\epsilon_{\max}}(0) \times D$ in execution. □

9.4 Results

To demonstrate LMTCD-RRT on a wide range of systems, we show our method on a 4D acrobot, 4D nonholonomic car, a 6D underactuated quadrotor, and a 22D rope manipulation task. Throughout, we will compare with four baselines to show the need to both use the bound (9.12) and to remain within D, where the bound is accurate: B1) planning inside D and assuming the model error is uniformly bounded by the average training error $\|d(t)\| \leq \sum_{i=1}^{N} e_i/N$ to compute $\bar{\epsilon}(t)$, B2) planning inside D and using the maximum training error $\|d(t)\| \leq \max_{1 \leq i \leq N} e_i$ as a uniform

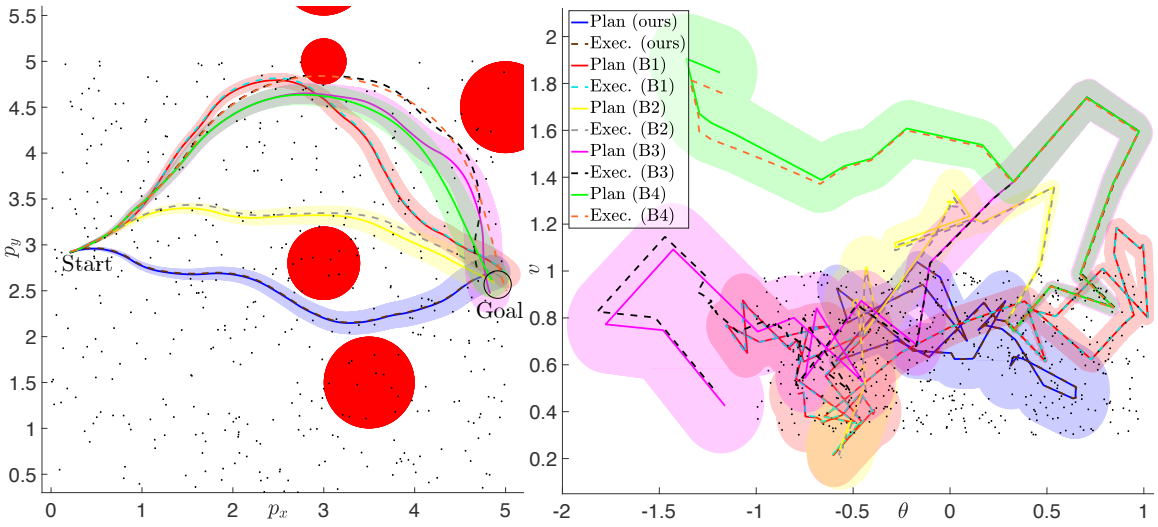


Figure 9.3: 4D car; planned (solid lines) and executed trajectories (dotted lines). The filled red circles are obstacles. Tracking tubes for all methods are drawn in the same color as the planned trajectory. To aid in visualizing D, the small black dots are a subsampling of $\mathcal{S}$. We plot state space projections of the trajectories: **Left**: projection onto the x, y coordinates; **Right**: projection onto the θ, v coordinates. For this example, LMTCD-RRT, B1, and B2 remain in D in execution, while B3 and B4 exit D, and also exit their respective tracking tubes, leading to crashes.

	Avg. trk. error (Car)	Goal error (Car)	Avg. trk. error (Quadrotor)	Goal error (Quadrotor)	Avg. trk. error (Rope)	Goal error (Rope)
LMTCD-RRT	0.008 ± 0.004 (0.024)	0.009 ± 0.004 (0.023)	0.0046 ± 0.0038 (0.0186)	0.0062 ± 0.0115 (0.0873)	0.0131 ± 0.0063 (0.0278)	0.0125 ± 0.0095 (0.0352)
B1: Mean, in D	0.019 ± 0.012 (0.054)	0.023 ± 0.016 (0.078)	0.0052 ± 0.0051 (0.0311)	0.0104 ± 0.0161 (0.0735)	18.681 ± 55.917 (167.79)	42.307 ± 126.81 (380.45)
B2: Max, in D	0.02 ± 0.01 (0.05) [19/50]	0.019 ± 0.012 (0.062) [19/50]	— [65/65]	— [65/65]	17.539 ± 52.380 (157.22)	21.595 ± 64.295 (193.05)
B3: Max, $\notin D$	0.457 ± 0.699 (3.640)	1.190 ± 1.479 (7.434)	0.1368 ± 0.2792 (1.5408)	0.8432 ± 1.3927 (9.0958)	111.86 ± 39.830 (170.96)	236.34 ± 72.622 (331.83)
B4: Lip., $\notin D$	0.704 ± 2.274 (13.313)	2.246 ± 8.254 (58.32)	0.4136 ± 0.4321 (1.9466)	1.8429 ± 1.5260 (6.9859)	17.301 ± 49.215 (148.43)	36.147 ± 52.092 (147.76)

Table 9.1: Statistics for the car, quadrotor, and rope. Mean ± standard deviation (worst case) [if nonzero, number of failed trials].

bound, B3) not remaining in D in planning and assuming a uniform bound on model error $\|d(t)\| \leq \max_{1 \leq i \leq N} e_i$ in computing $\bar{\epsilon}(t)$ for collision checking, and B4) not remaining in D and using our error bound (9.12). We note that B3-type assumptions are common in prior CCM work *Singh et al.* (2019); *Sun et al.* (2020). In baselines that leave D, the space is unconstrained: $\mathcal{X} = \mathbb{R}^{n_x}$, $\mathcal{U} = \mathbb{R}^{n_u}$. We set the FTG-based estimation probability $\rho = 0.975$ for each constant. Please see Table 9.1 for planning statistics and https://tinyurl.com/lmtcdrrt for a supplementary video which overviews the method and visualizes our results.

Acrobot (4D): We consider the Acrobot system (an underactuated double pendulum system with a single actuator at the "elbow"); the equations of motion can be found in *Stachowiak and Okada* (2006). The system state is four-dimensional, containing the angular positions and velocities: $x = [\theta_1, \theta_2, \dot{\theta}_1, \dot{\theta}_2]^\top$, and the control is one-dimensional: $u = u_1$. As this model satisfies (9.15), we use the stronger CCM conditions (9.2). We use 100000 training data-points, with states uniformly sampled from a tube of width 0.3 around a nominal swing-up trajectory obtained via trajectory optimization on the true dynamics, and with controls uniformly sampled between $[-3, 3]$. We note that the executed control at runtime is not explicitly constrained to fall in this range; however, at planning time, satisfaction of these constraints is

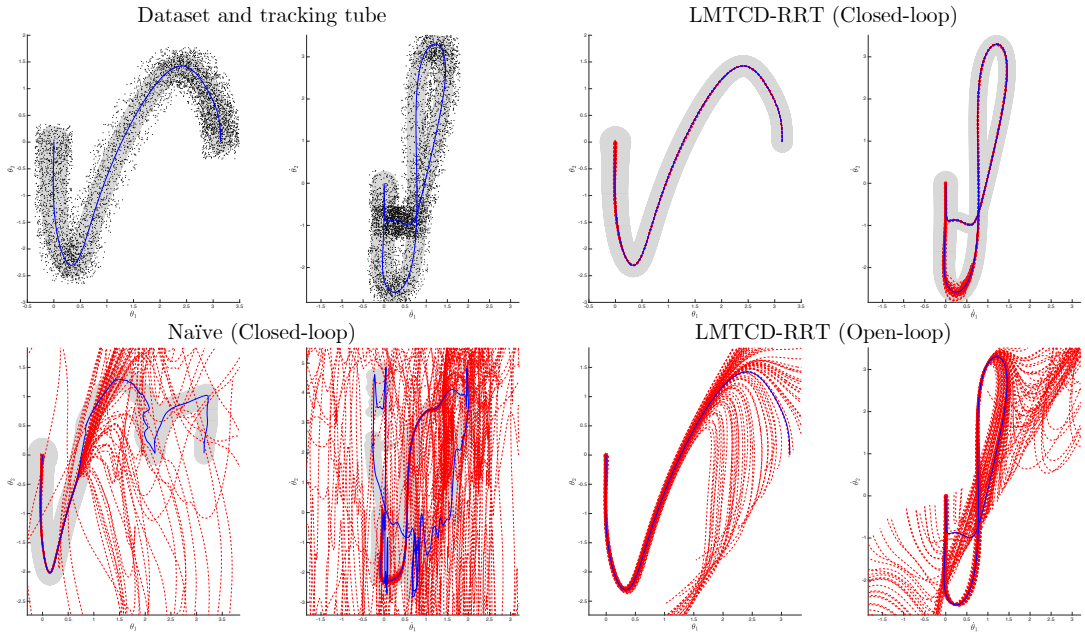


Figure 9.4: 4D underactuated double pendulum; planned (blue lines) and executed trajectories from various perturbed initial conditions (red lines). Tracking tubes are shown in grey. The small black dots are a subsampling of $\mathcal{S}$. We plot state space projections of the trajectories onto the θ_1, θ_2 coordinates, as well as the $\dot{\theta}_1$, $\dot{\theta}_2$ coordinates. **Top left**: the tracking tube computed by LMTCD-RRT, wrapped around a trajectory planned with the learned dynamics (blue), and overlaid by a subsampling of the training data. **Top right**: when using the CCM-based tracking controller, LMTCD-RRT remains within its tracking tube, and robustly converges to the upright equilibrium. **Bottom right**: executing the plan computed with LMTCD-RRT open-loop diverges, due to the chaotic nature of the system. **Bottom left**: a plan which exits D results in divergence at runtime, leading to failure to converge to the upright equilibrium.

ensured (probabilistically) by the estimated upper bound on the feedback control $\bar{u}_{\text{fb}}$, together with the check for containment in D. We use this data to train $f(x)$, $B(x)$, and $M(x)$, with the x needed to train $M(x)$ coming directly from the state data in the training set. We model f as an NN with five hidden layers, each of width 512, and B as a vector of four learnable parameters. We model $M(x)$ as an NN with three hidden layers, each of size 512. For simplicity, instead of representing D as a union of balls around the training data, we simply select D to be $\{x \mid \exists t, \|x - \xi_t\| \leq 0.3\} \times [-3, 3]$, where ξ_t is the state at time t on the nominal swing-up trajectory.

In Fig. 9.4, we demonstrate that our approach can robustly stabilize around a swing-up trajectory planned with the learned dynamics. Using LMTCD-RRT, we plan a trajectory using the learned dynamics (Fig. 9.4, top left, blue), starting from the downwards equilibrium $x_0 = [0, 0, 0, 0]^\top$, and swinging up to the upright equilibrium point $x_G = [\pi, 0, 0, 0]^\top$. To test the robustness of the CCM-based tracking controller around this plan, we additionally perturb the initial conditions such that the initial Riemannian energy is bounded by 0.03. The corresponding tracking tube

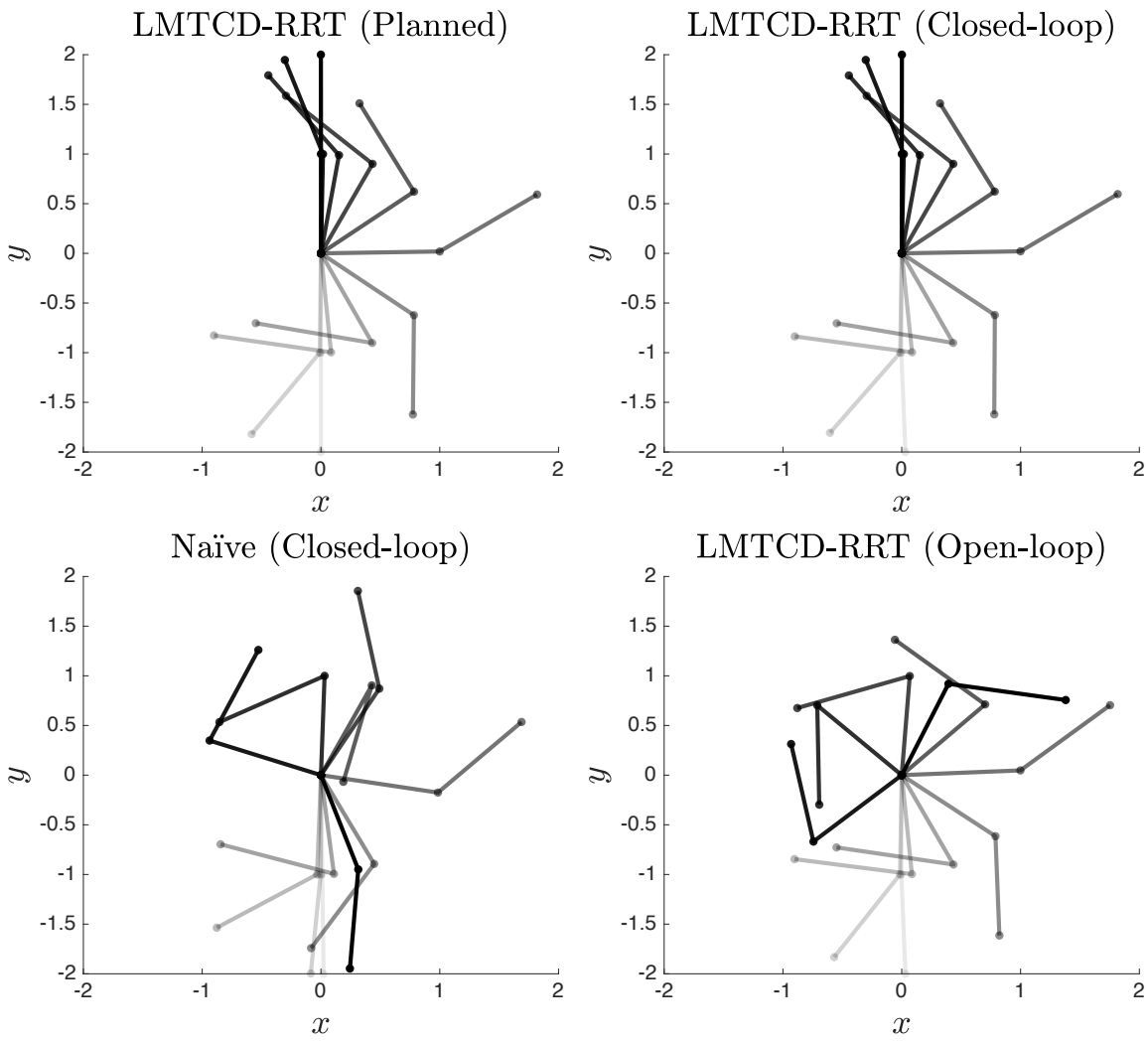


Figure 9.5: 4D underactuated double pendulum; time-lapse of planned trajectories and executed rollouts. Fainter (darker) lines correspond to configurations close to the beginning (end) of the rollout.

	Avg. trk. error (Acrobot)	Goal error (Acrobot)
LMTCD-RRT (closed-loop)	0.0161 ± 0.0105 (0.2394)	0.0048 ± 0.0050 (0.0255)
LMTCD-RRT (open-loop)	1.7611 ± 1.4001 (47.7217)	8.2124 ± 8.2605 (46.4244)
B3: Max, $\notin D$	7.2321 ± 3.7502 (138.3515)	31.8162 ± 22.3194 (110.4946)

Table 9.2: Statistics for the acrobot example. Mean ± standard deviation (worst case).

which accounts for the perturbation in initial condition and the model error is shown in grey in Fig. 9.4. Using the CCM-based controller around the plan generated by LMTCD-RRT successfully reaches the upright equilibrium, and all trajectories remain well within the computed tube (Fig. 9.4, top right). To demonstrate the need for using the CCM-based controller, we can see that for most of the perturbed initial conditions, executing the control trajectory planned by LMTCD-RRT in an open-loop fashion (Fig. 9.4, bottom right) leads to divergence and failure to reach the upright equilibrium; this is reflective of the chaotic nature of the double pendulum system. We also compare with a naïve planner, which is not constrained to remain within D (Fig. 9.4, bottom left). As expected, the executed rollouts exit the tubes, as the model error bound underapproximates the error outside of D, and diverge wildly due both due to 1) the inability to track the planned trajectory, which contains several sharp kinks which are artifacts of inaccuracies in the learned dynamics, and 2) the failure of the CCM-based controller, which may behave poorly outside of D. For intuition, we also plot time-lapses of the executed trajectory for one of the perturbed initial conditions in Fig. 9.5. Finally, numerical statistics on the average tracking error pointwise in time over the rollouts, as well as the average goal error, are presented in Table 9.2, and show that our method yields both the lowest average tracking error and goal error.

Nonholonomic car (4D): We consider the vehicle model

$$\begin{bmatrix} \dot{p}_x \\ \dot{p}_y \\ \dot{\theta} \\ \dot{v} \end{bmatrix} = \begin{bmatrix} v\cos(\theta) \\ v\sin(\theta) \\ 0 \\ 0 \end{bmatrix} + \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} \omega \\ a \end{bmatrix},$$

where $u = [\omega, a]^\top$. As this model satisfies (9.15), we use the stronger CCM conditions (9.2). We use 50000 training data-points uniformly sampled from $[0, 5] \times [-5, 5] \times [-1, 1] \times [0.3, 1]$ to train $f(x)$, $B(x)$, and $M(x)$, with the x needed to train $M(x)$ coming directly from the state data in $\mathcal{S}$. We model f and B as NNs, each with a single hidden layer of size 1024 and 16, respectively. We model $M(x)$ as an NN with two hidden layers, each of size 128. In training, we set $\underline{w} = 0.01$ and gradually increase α_1 and α_2 to 0.01 and 10, respectively. We select $r = 0.6$ by incrementally growing r as described in Sec. 9.3.3, collecting 5000 new datapoints for Ψ, giving us $\lambda = 0.09$, $L_{h-g} = 0.006$. $\delta_u = 1.01$, $\bar{\lambda}_D(M) = 0.258$, and $\underline{\lambda}_D(M) = 0.01$.

We plan for 50 different start/goal states in D, taking on average 6 mins, and compare against the four baselines. We visualize one trial in Fig. 9.3. Over the trials, LMTCD-RRT and B2 never violate their respective bounds in any of the trials, while B1, B3, and B4 violate their bounds in 6, 48, and 43 of the 50 trials, respectively, which could lead to a crash (indeed, B3 and B4 crash in Fig. 9.3). This occurs as the tracking error bounds for the baselines are invalid, as the baselines' model error bounds underestimate the true model error which could be seen in execution. Moreover, planning is infeasible in 19/50 of B2's trials, because the large uniform error bound can make it impossible to reach a goal while remaining in D. This suggests the utility of a fine-grained disturbance bound like (9.9), especially when

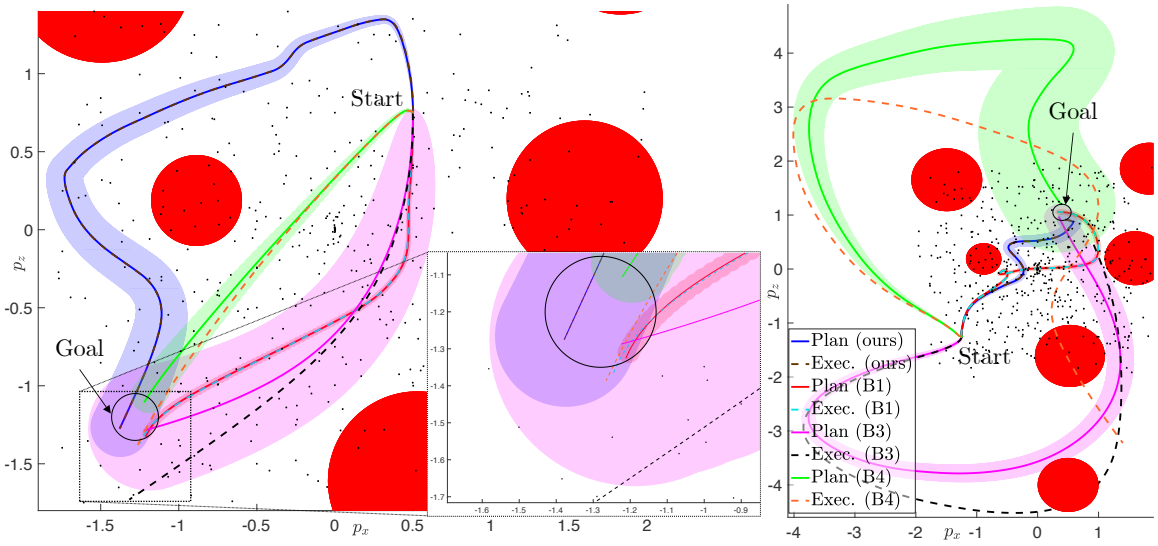


Figure 9.6: 6D planar (xz plane) quadrotor; planned (solid lines) and executed trajectories (dotted lines). The filled red circles are obstacles. Tracking tubes for all methods are drawn in the same color as the planned trajectory. The small black dots are a subsampling of $\mathcal{S}$. We plot state space projections of the trajectories onto the p_x, p_z coordinates. **Left**: for this example, LMTCD-RRT remains within its tracking tube, and all baselines violate their respective bounds near the end of execution (see inset). **Right**: for this example, LMTCD-RRT remains within its tracking tube, and B3 and B4 exit D and crash.

planning in D, which is quite constrained. Note that while B2 does not crash in this particular example, using the maximum training error can still be unsafe (as will be seen in later examples), as the true error can be higher in $D \setminus \mathcal{S}$. Finally, we note that the tracking accuracy difference between LMTCD-RRT and B1/B2 reflects that the spatially-varying error bound steers LMTCD-RRT towards lower error regions in D. Overall, this example suggests that (9.12) is accurate, while coarser disturbance bounds or exiting D can be unsafe.

Underactuated planar (2D) quadrotor (6D state space): We consider the quadrotor model in (*Singh et al.*, 2019, p.20) with six states and two inputs:

$$\begin{bmatrix} \dot{p}_x \\ \dot{p}_y \\ \dot{\phi} \\ \dot{v}_x \\ \dot{v}_z \\ \ddot{\phi} \end{bmatrix} = \begin{bmatrix} v_x \cos(\phi) - v_z \sin(\phi) \\ v_x \sin(\phi) + v_z \cos(\phi) \\ \dot{\phi} \\ v_z \dot{\phi} - g \sin(\phi) \\ -v_x \dot{\phi} - g \cos(\phi) \\ 0 \end{bmatrix} + \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 1/m & 1/m \\ l/J & -l/J \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \end{bmatrix},$$

where $x = [p_x, p_z, \phi, v_x, v_z, \dot{\phi}]$, modeling the linear/angular position and velocity, and $u = [u_1, u_2]$, modeling thrust. We use the parameters $m = 0.486$, $l = 0.25$, and $J = 0.125$. These dynamics also satisfy (9.15), so we use the stronger CCM conditions (9.2). We sample 245000 training points from $[-2, 2] \times [-2, 2] \times [-\pi/3, \pi/3] \times [-1, 1] \times [-1, 1] \times [-\pi/4, \pi/4]$ to train $f(x)$, $B(x)$, and $M(x)$, with the x data for training $M(x)$

being the state data in $\mathcal{S}$. We model f and B as NNs with a single hidden layer of size 1024 and 16, respectively. We model $M(x)$ as an NN with two hidden layers of size 128. In training, we set $\underline{w} = 0.01$ and gradually increase α_1 and α_2 to 0.001 and 0.33, respectively. We select $r = 1.0$ by incrementally growing r as in Sec. 9.3.3, resulting in 10000 new datapoints for Ψ. This gives us $\lambda = 0.09$, $L_{h-g} = 0.007$. $\delta_u = 1.9631$, $\bar{\lambda}_D(M) = 4.786$, and $\underline{\lambda}_D(M) = 0.0909$.

We plan for 65 different start/goal states within D, taking 1 min on average, and compare against the baselines. We visualize two trials in Fig. 9.6. Our attempts to run B2 failed, as the error bound was too large to feasibly plan within D in all 65 trials, again suggesting the need for a local model error bound. Over these trials, LMTCD-RRT never violates its computed bound in execution, while B1, B3, and B4 violate their bounds 14/65, 32/65, and 65/65 times, respectively. As for the car example, these bounds are violated because the model error descriptions assumed by these baselines can underestimate the true model error seen in execution. From Table 9.1, one can see that LMTCD-RRT obtains the lowest error, though it is closely matched by B1. However, LMTCD-RRT never violates the tracking tubes in execution, while B1 does (i.e., Fig. 9.6, left). B3-B4 perform poorly, with the controller failing to overcome the model error, causing crashes (Fig. 9.6, right). Overall, this example highlights the need for LMTCD-RRT's local error bounds while demonstrating our method's applicability to highly-underactuated systems.

10-link rope (22D): To demonstrate that our method scales to high-dimensional, non-polynomial systems well beyond the reach of SoS-based methods, we consider a planar rope manipulation task simulated in Mujoco *Todorov et al.* (2012). We consider a 10-link (11-node) rope approximation, where each link can stretch, and the head of the rope (see Fig. 9.7(d)) is velocity-controlled. The system has 22 states: the first two contain the xy position of the head, and the rest are the xy positions of the other nodes, relative to the head, and the system has two controls for the commanded xy head velocities. We wish to steer the *tail* of the rope to a given xy region while ensuring the rope does not collide in execution (cf. Fig. 9.7). This is a challenging task, as the tail of the rope is highly underactuated, and steering it to a goal requires modeling the complicated friction forces that the rope is subjected to. We collect three demonstrations to train the dynamics (see the supplementary video for a visualization): in the first, the rope begins horizontally and performs a counterclockwise elliptical loop; the second starts vertically and moves up, the third begins horizontally and moves right, giving a total of 20500 datapoints. As the rope dynamics do not satisfy (9.15), we learn both $M(x)$ and $u(\tilde{x}, x^*, u^*)$; to do so, we sample 20500 state/control perturbations around the demonstrations and evaluate the dynamics at these points, giving $|\mathcal{S}| = 41000$. We model f and B as three-layer NNs of size 512. $M(x)$ is modeled with two hidden layers of size 128, and $u(\tilde{x}, x^*, u^*)$ is modeled with a single hidden layer of size 128. In training, we set $\underline{w} = 1.0$ and gradually increase α_1 and α_3 to 0.005 and 0.56, respectively. To ensure that the CCM and controller are invariant to translations of the rope, we enforce $M(x)$ and $u(\tilde{x}, x^*, u^*)$ to not be a function of the head position. To simplify the dynamics learning, we note that as the head is velocity-controlled, it can be modeled as a single-integrator; we hardcode this structure and learn the dynamics for the other

20 states. We obtain $\epsilon_{\max} = 0.105$ and select $r = 0.5$ by incrementally growing r as in Sec. 9.3.3, resulting in $|\Psi| = 10000$, $\lambda = 0.0625$, $L_{h-g} = 0.023$, $\bar{u}_{\text{fb}} = 0.249$, $\bar{\lambda}_D(M) = 3.36$, and $\underline{\lambda}_D(M) = 1$.

We plan for 10 different start/goal states within D, taking 9 min on average, and compare against the baselines. As this example uses the learned controller (9.17), we adapt the baselines so that B1 and B2 remain in $\mathcal{B}_{\epsilon_{\max}}(0) \times D$, while B3 and B4 are unconstrained. We visualize one task in Fig. 9.7: the rope starts horizontally, with the head at $[0, 0]$, and needs to steer the tail to $[3, 0]$, within a 0.15 tolerance. LMTCD-RRT stays very close to the training data, reaching the goal with small tracking tubes. B1 and B2 also remain close to the training data as they plan in D, but as both the mean and maximum bounds may underestimate the true model error in D, they stray too close to the boundary of D, and the larger model error pushes them out of D, causing the system to become unstable as the learned u applies large inputs in an attempt to stabilize around the plan. B3 exploits errors in the model, planning a trajectory which is highly unrealistic. This is allowed to happen because the maximum error severely underestimates the model error outside of D, leading to a major underestimate of the tracking error that would be seen in execution. When executing, the system immediately goes unstable due to the large distance between the plan and the training data. The plan from B4 is forced to remain close to the training data at first, in order to move through the narrow passage. This is because the Lipschitz bound, while an underestimate outside of D, still grows quickly with distance from $\mathcal{S}$; attempting to plan a trajectory similar to B3 fails, since the tracking error tube grows so large in this case that it becomes impossible to reach the goal without the tube colliding with the obstacles. After getting through the narrow passage, B4 drifts from D and correspondingly fails to be tracked beyond this point. Over these 10 trials, LMTCD-RRT never violates the computed tracking bound, and B1, B2, B3, and B4 violate their bounds in 10, 6, 10, and 9 trials out of 10, respectively. Overall, this result suggests that contraction-based control can scale to very high-dimensional systems (i.e., deformable objects) if one finds where the model and controller are good and takes care to stay there during planning and execution.

9.5 Limitations and Future Directions

In this section, we discuss some limitations of the approach that we propose in this chapter, as well as ways that these limitations can be addressed.

- **Finding a valid CCM for the learned dynamics can be a challenging and unreliable process**. By representing the CCM as a neural network, we resort to using standard optimization techniques based on stochastic gradient descent to minimize violation of the CCM conditions (9.3) on the training set; however, this training procedure is quite prone to local optima. One particularly common local optimum is one where the CCM becomes nearly uniform across the training set, and all of its eigenvalues approach the preset minimum eigenvalue $\underline{w}$; this is a local optimum which leads to a smaller magnitude of

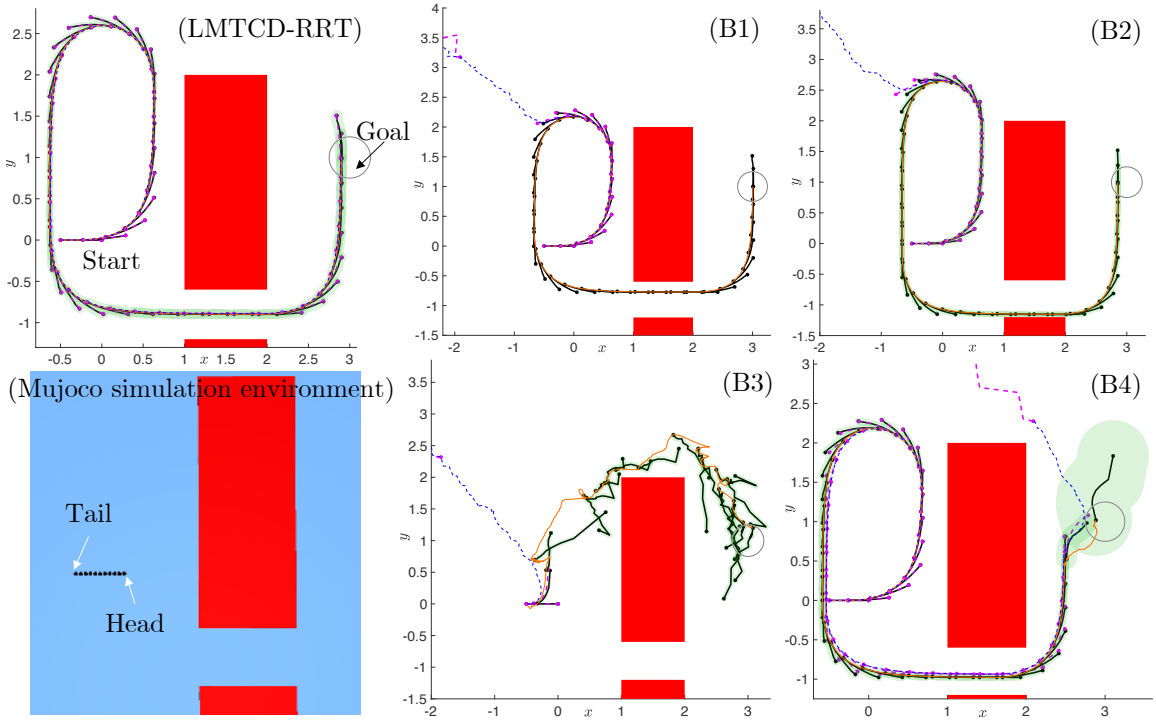


Figure 9.7: 22D planar rope dragging task. Snapshots of the planned trajectory are in black, snapshots of the executed trajectory are in magenta, and the tracking error tubes are in green. For further concreteness, for each snapshot, we mark the head of the rope with an asterisk, and we mark the tail of the rope with a solid dot. Additionally, the trajectory of the tail in the plan is plotted in orange, while the trajectory of the tail in execution is plotted in blue. Only LMTCD-RRT reaches the goal, while all baselines become unstable when attempting to track their respective plans. We also show the original Mujoco simulation environment in the bottom left.

violation of the CCM condition (9.3), but generally at this configuration, all training points tend to still violate (9.3) by some margin. Heuristics that we have explored which discourage convergence to this optimum are still not very reliable, and oftentimes many random weight initializations are required to find a valid CCM. More work is required to improve the robustness of CCM training, either in improving the loss function to be more robust to these local optima, or in improving the initialization step by reliably initializing the weights within the basin of attraction of a valid configuration.

- **Systems cannot be arbitrarily underactuated in D.** If the stabilizability properties of the system are not strong enough in the trusted domain D, a CCM may not exist, or a CCM that does exist is nearly unusable, e.g., $\frac{\bar{\lambda}_D(M)}{\underline{\lambda}_D(M)}$ becomes extremely large. Worsening this problem is the previous limitation: the CCM search procedure is far from complete – even if a CCM exists, we are not guaranteed to find one through our proposed procedure. This can make it challenging to determine if we need to adjust the domain D over which we are searching for a CCM, or if we should just restart the training procedure from a new initialization.

- **Training accurate dynamics models and valid CCMs is a data-hungry process**: in using standard multi-layer perceptrons to represent the dynamics and CCM, accurate generalization to unseen states is generally poor. Consequently, we require large datasets in order to learn dynamics models which are sufficiently accurate to induce tubes of a reasonable size at planning time. Similarly, to learn a valid CCM, we generally need to enforce (9.3) at many locations to be valid. To improve this, we can look into leveraging dynamics models with built-in invariances *Zhang et al.* (2021) for improved out-of-distribution generalization, and modifying our model error bound to account for this broader generalization.

- **Obtaining informative data can be challenging**. In this work, we assume that we have enough data to learn a sufficiently accurate dynamics model in the regions of the state space relevant for steering the system to the desired goal region. This is realistic if, for instance, demonstrations are provided to help guide the data collection (as is assumed in the acrobot and rope examples in Sec. 9.4). In general, without this guidance, it can be challenging to design a good policy for collecting task-relevant dynamics data. For instance, random trajectories which attempt to generate data for solving the acrobot swing-up task in Sec. 9.4 are likely to fail, as the system's underactuation makes it challenging to reach parts of the state space where data is needed to complete the task. An important direction for future work revolves around designing principled strategies for quickly identifying structure within the underactuated system to enable informed data collection.

- **Known state representations**. In this work, we assume that we know the state variables relevant for completing the task, i.e., the state representation

x is known, and we aim to learn a dynamics model within the limitations of that fixed state representation. In general, one may not know which state variables should be included to learn a sufficiently accurate model, especially when learning models from raw observations. Extending our approach to certify safety for learned latent space planning modules is a direction of our current research.

- **Sim-to-real gap**. It can be challenging to collect sufficiently large and accurately-annotated datasets on hardware platforms, so we may opt to collect data in a simulator to generate the data used to train our models. However, the simulator is inevitably an approximation of the real system, and this can increase the inaccuracy of the learned model relative to the true system. Future work involves bounding the discrepancy between the simulated and true systems using online data and using this to buffer our computed tracking tubes, and extending our approach to work with stochastic systems with noise corruption in the training dataset. One possible way to achieve this is by directly training a spatially-varying error bound (and then bounding the underapproximation error with a spatially-constant buffer, estimated via FTG), instead of parameterizing the model error with the Lipschitz constant, which is highly sensitive to noise.

- **Current tube-based planning is conservative and not adaptive**. Two of the largest bottlenecks in planning are 1) the conservativeness of the tracking tubes, as well as 2) the conservativeness of our check for containment in the trusted domain D. For the first shortcoming, a major reason for the conservativeness lies in the fact that we only leverage offline data for training the dynamics model. To mitigate this issue, we can look into an adaptive variant, where online data is used to refine the bound on the tracking error, or resort to shorter-horizon MPC-style planning, which can still admit safety guarantees if we compute an encompassing robust controlled-invariant set which we can keep the system within. To mitigate the second issue, we can resort to more computationally-expensive checks to determine if a query ball (the set of states that the system is guaranteed to remain within) is contained inside a union of balls (the trusted domain), or by computing large convex regions of free space (denoted $\mathcal{C}$) contained within D (as in *Deits and Tedrake* (2014)) and checking containment of the query ball inside $\mathcal{C}$, instead of individual balls in the trusted domain, as is done in this chapter.

9.6 Conclusion

We present a method for safe feedback motion planning with unknown dynamics. To achieve this, we jointly learn a dynamics model, a contraction metric, and contracting controller, and analyze the learned model error and trajectory tracking bounds under that model error description, all within a trusted domain. We then use these tracking bounds together with the trusted domain to guide the planning of probabilistically-safe trajectories; our results demonstrate that ignoring either com-

ponent can lead to plan infeasibility or unsafe behavior. In the next chapter, we will discuss how we can build upon the ideas presented in this chapter in order to guarantee safety and robust goal reachability when controlling from image observations, in the face of uncertain dynamics and potentially inaccurate learned perception modules.

CHAPTER X

Safe Output Feedback Motion Planning from Images via Learned Perception Modules and Contraction Theory

In this chapter, we present a motion planning algorithm for a class of uncertain control-affine nonlinear systems which guarantees runtime safety and goal reachability when using high-dimensional sensor measurements (e.g., RGB-D images) and a learned perception module in the feedback control loop. First, given a dataset of states and observations, we train a perception system that seeks to invert a subset of the state from an observation, and estimate an upper bound on the perception error which is valid with high probability in a trusted domain near the data. Next, we use contraction theory to design a stabilizing state feedback controller and a convergent dynamic state observer which uses the learned perception system to update its state estimate. We derive a bound on the trajectory tracking error when this controller is subjected to errors in the dynamics and incorrect state estimates. Finally, we integrate this bound into a sampling-based motion planner, guiding it to return trajectories that can be safely tracked at runtime using sensor data. We demonstrate our approach in simulation on a 4D car, a 6D planar quadrotor, and a 17D manipulation task with RGB(-D) sensor measurements, demonstrating that our method safely and reliably steers the system to the goal, while baselines that fail to consider the trusted domain or state estimation errors can be unsafe. This chapter is based off the paper *Chou et al.* (2022a).

10.1 Introduction

Safely and reliably deploying an autonomous robot requires a systematic analysis of the uncertainties that it may face across its perception, planning, and feedback control modules. State-of-the-art methods largely analyze each module separately; e.g., by first certifying perception *Yang et al.* (2021), finding a safe plan under a nominal dynamics model *LaValle* (2006), and then using a stable tracking controller *Singh et al.* (2019). However, this ignores how the errors in each module can propagate. Inaccuracies in the dynamics and perception can destabilize the downstream

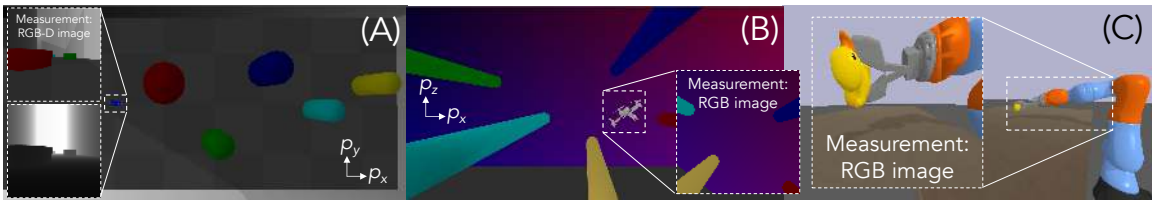


Figure 10.1: For a 4D car, a 6D quadrotor, and a 14D arm, we compute plans that can be safely stabilized to reach goals at runtime using rich sensor observations in the form of RGB(-D) images.

feedback controller and lead to failure, revealing a need to unify perception, planning, and control to guarantee safety for the end-to-end autonomy pipeline.

To address this gap, we consider one such unified approach: the Output Feedback Motion Planning problem (OFMP) *Renganathan et al.* (2020), which jointly plans nominal trajectories and designs feedback controllers which safely stabilize the system to some goal when using imperfect state information (i.e., output feedback). A concrete way to solve the OFMP is to bound the set of states that the system may reach while tracking a plan using output feedback, that is, a closed-loop output feedback trajectory tracking tube, and ensure it is collision-free. Practical robots present challenges in solving the OFMP:

1. The tracking tubes should be efficiently computable for arbitrary trajectories so that they can be used in the planning loop to restrict the set of states that can be safely visited. However, solving this reachability problem is computationally demanding.

2. Processing rich sensor data (e.g., images, depth maps, etc.) at runtime is often done via deep learning-based perception modules, which are powerful but error-prone. Bounding this error and bounding its effect on trajectory tracking error is difficult.

To address the first challenge, we use contraction theory, which is of specific interest for the OFMP as it enables the 1) design of stabilizing feedback controllers *Manchester and Slotine* (2017) and convergent state estimators *Dani et al.* (2015) and 2) fast computation of tracking/estimation tubes, given a bound on the disturbances that the controller and observer are subjected to *Manchester and Slotine* (2014). Estimating this bound is central to our solution of the second challenge, where we use data to 1) estimate a bound on the error of a learned perception module which is valid with high probability and 2) bound the level to which incorrect state estimates can destabilize the controller. Combining these solutions provides accurate tubes that can be used in planning. In summary, we develop a contraction-based output feedback motion planning algorithm for control-affine systems stabilized from image observations, which retains probabilistic guarantees on safety and goal reachability. Our specific contributions are:

- A learning-based framework for integrating high-dimensional observations into

contraction-based control and estimation that can generalize across environments

- A trajectory tracking error bound for contraction-based feedback controllers in output feedback, subjected to a disturbance that accurately reflects the perception error
- A sampling-based planner which solves the OFMP, returning plans that can be safely tracked and that reliably reach the goal at runtime using image observations
- Validation in simulation on a 4D nonholonomic car, a 6D planar quadrotor, and a 17D manipulation task, maintaining safety whereas baseline approaches fail

10.2 Preliminaries and Problem Statement

We consider uncertain continuous-time control-affine nonlinear systems (which include many common mechanical systems of interest *LaValle* (2006)) with output observations

$$\dot{x}(t) = f(x(t)) + Bu(t) + B_w(t)w_x(t) \tag{10.1a}$$
$$y(t) = h(x(t), \theta) + B_y w_y(t) \tag{10.1b}$$

where $f : \mathcal{X} \to \mathcal{X}$, $\mathcal{X} \subseteq \mathbb{R}^{n_x}$, $B \in \mathbb{R}^{n_x \times n_u}$, $B_w : [0, \infty) \to \mathbb{R}^{n_x \times n_{w_x}}$, $B_y \in \mathbb{R}^{n_y \times n_{w_y}}$, $\mathcal{U} \subseteq \mathbb{R}^{n_u}$, and $w_x \in \mathbb{R}^{n_{w_x}}$ is a possibly stochastic state disturbance where $\|w_x(t)\| \leq \bar{w}_x$, for all t. Without loss of generality, we assume $\|B_w(t)\| \leq 1$, for all t. Norms $\|\cdot\|$ without subscript are the (induced) 2-norm. We obtain high-dimensional observations $y \in \mathcal{Y} \subseteq \mathbb{R}^{n_y}$ (e.g., $N \times N$-pixel RGB-D images, leading to $n_y = 4N^2$), generated by a deterministic, nonlinear function $h(x, \theta) : \mathcal{X} \times \Theta \to \mathcal{Y}$ which is unknown to the robot; here, $\theta \in \mathbb{R}^{n_p}$ are external parameters (e.g., location of obstacles, map of environment, etc.). The observations may be corrupted by (possibly stochastic) sensor noise $w_y(t) \in \mathbb{R}^{n_{w_y}}$, where $\|w_y(t)\| \leq \bar{w}_y$, for all t. We note that our results also apply to time-varying $B(t)$ under some conditions on its null-space.

We assume that (10.1a) is locally incrementally exponentially stabilizable (IES) in domain $D_c \subseteq \mathcal{X}$, that is, there exists an α, $\lambda > 0$, and some feedback controller such that for any nominal trajectory $x^*(t) \subseteq D_c$, $\|x^*(t) - x(t)\| \leq \alpha e^{-\lambda t}\|x^*(0) - x(0)\|$ for all t. While stronger than asymptotic stability, many underactuated systems are IES *Manchester et al.* (2015). We also assume that (10.1) is locally universally detectable *Manchester and Slotine* (2014), which ensures that any two trajectories $x_1(t)$ and $x_2(t)$ in a domain $D_e \subseteq \mathcal{X}$ that yield identical observations $y_1(t)$ and $y_2(t)$ for all t converge to each other as $t \to \infty$, i.e., $x_1(t) \to x_2(t)$. Similar assumptions are common in the estimation literature *Maybeck* (1979) to ensure estimator convergence, and do not require the full state to be observable instantaneously, e.g., as in *Dean et al.* (2020b).

Definitions: We assume $\mathcal{X}$ is partitioned into (un)safe ($\mathcal{X}_{\text{unsafe}}$) $\mathcal{X}_{\text{safe}}$ sets (e.g., obstacles). Let ($\mathbb{S}_n^{>0}$) $\mathbb{S}_n$ be the set of (positive definite) symmetric $n \times n$ matrices. For

$Q \in \mathbb{S}_n$, denote $\bar{\lambda}(Q)$ and $\underline{\lambda}(Q)$ as its maximum and minimum eigenvalues. If $Q(x)$ is a matrix-valued function over a domain D, we denote $\bar{\lambda}_D(Q) \doteq \sup_{x\in D} \bar{\lambda}(Q(x))$ and $\underline{\lambda}_D(Q) \doteq \inf_{x\in D} \underline{\lambda}(Q(x))$. Let the Lie derivative of a matrix-valued function $Q(x) \in \mathbb{R}^{n\times n}$ along a vector $y \in \mathbb{R}^n$ be denoted as $\partial_y Q(x) \doteq \sum_{i=1}^n y^i \frac{\partial Q}{\partial x^i}$, where x^i is the ith element of vector x. For a smooth manifold $\mathcal{X}$, a Riemannian metric tensor $M : \mathcal{X} \to \mathbb{S}^{>0}_{n_x}$ provides the tangent space $T_x\mathcal{X}$ with an inner product $\delta_x^\top M(x)\delta_x$, where $\delta_x \in T_x\mathcal{X}$. The length $l(c)$ of a curve $c : [0,1] \to \mathcal{X}$ between $c(0)$, $c(1)$ is $l(c) \doteq \int_0^1 \sqrt{V(c(s), c_s(s))}ds$, where $V(c(s), c_s(s)) \doteq c_s(s)^\top M(c(s))c_s(s)$, and $c_s(s) \doteq \partial c(s)/\partial s$. The Riemannian distance between $p, q \in \mathcal{X}$ is $d(p, q) \doteq \inf_{c\in\mathcal{C}(p,q)} l(c)$, where $\mathcal{C}(p, q)$ contains all smooth curves between p and q; a curve $\gamma(p, q)$ achieving the argmin is called a geodesic.

10.2.1 Problem statement

We formally state the output feedback motion planning problem (OFMP) as follows:

OFMP: Given start x_I, external parameter $\theta \in D_\theta$, goal region $\mathcal{G} \subseteq D_x$ (D_θ, D_x are defined in the next paragraph), and safe set $\mathcal{X}_{\text{safe}}$, we want to plan a state-control trajectory $x^* : [0, T] \to \mathcal{X}$, $u^* : [0, T] \to \mathcal{U}$, $x^*(0) = x_I$, under the nominal dynamics $\dot{x}(t) = f(x(t)) + Bu(t)$ such that in execution on the true system (10.1a), $x(t) \in \mathcal{X}_{\text{safe}}$ for all $t \in [0, T]$ and $x(T) \in \mathcal{G}$. At runtime, we do not observe $x(t)$; we are only given observations $y(t)$ generated by (10.1b), and must track x^* using a (dynamic) output feedback controller that we must also design. We assume f, B, B_w, and B_y are known; h is unknown; w_x, w_y are not measurable but $\bar{w}_x$ and $\bar{w}_y$ are known. If $n_r \leq n_x$ of the states can be inferred directly from y, we denote these indices as the reduced observation $y_r = C_r x \in \mathbb{R}^{n_r}$, where $C_r \in \{0, 1\}^{n_r\times n_x}$ is a boolean matrix that selects the observable dimensions of x. We assume that we are given C_r. Let $x(t)$ be the executed trajectory of (10.1a), and let $\hat{x}(t)$ be the trajectory of the state estimate. We are given upper bounds $\bar{d}_c(0)$, $\bar{d}_e(0)$on the Riemannian distance between the true and estimated initial state $d_e(x(0), \hat{x}(0))$ and between the true/planned initial state $d_c(x^*(0), x(0))$; $d_e(\cdot, \cdot)$ and $d_e(\cdot, \cdot)$ are defined with respect to (w.r.t.) metrics M_c and M_e, defined in Sec. 10.2.2.

To help solve the OFMP, we are given two datasets. The first is $\mathcal{S} = \{h(x_i, \theta_i), x_i, \theta_i\}_{i=1}^{N_{\text{data}}}$, a dataset of noiseless (cf. Sec. 10.5 for some possibilities on how to relax this assumption) observation-state-parameter triplets, where $x_i \in D_p \subseteq \mathcal{X}$, $\theta_i \in D_\theta \subseteq \Theta$ are collected by any means (sampling, demonstrations, etc.). We assume D_p and D_θ (the domains where $\mathcal{S}$ is drawn from) are known, though this can be relaxed by estimating these sets as in *Chou et al.* (2021c); *Knuth et al.* (2021a). We are also given a validation dataset $\mathcal{V} = \{h(x_i, \theta_i), x_i, \theta_i\}_{i=1}^{N_{\text{val}}}$ collected i.i.d. in $D_p \times D_\theta$. In the context of (10.1b), $h(x, \theta)$ may be a simulated image, and $B_y w_y(t)$ is the sensor noise at runtime. We also define a "trusted domain" for planning, $D = D_x \times D_\theta \subseteq \mathcal{X} \times \Theta$, where $D_x = D_r \cap D_c \cap D_e$ and D_r is defined as follows: for ease, suppose C_r selects the first n_r indices of x, then $D_r = (C_r D_p) \times \mathbb{R}^{n_x - n_r}$. D_r is defined similarly if C_r selects other indices (cf. Fig. C.1). Ultimately, D_x is a set where a stabilizing controller (in D_c) and state estimator (in D_e) exist, and where the perception is valid (in D_r).

10.2.2 Control/observer contraction metrics (CCMs/OCMs)

As our approach builds on contraction theory, we provide an overview here. Control contraction theory *Manchester and Slotine* (2017) studies incremental stabilizability by measuring the distances between trajectories w.r.t. a Riemannian metric $M_c : \mathcal{X} \to \mathbb{S}^{>0}_{n_x}$. For (10.1a) if $w_x \equiv 0$, a sufficient condition *Singh et al.* (2019) for M_c to be called a control contraction metric (CCM) is:

$$B_\perp^\top \Big(-\partial_f W_c(x) + A(x)W_c(x) + W_c(x)A(x)^\top + 2\lambda_c W_c(x) \Big) B_\perp \preceq 0 \tag{10.2a}$$

$$B_\perp^\top \Big(\partial_{B^j} W_c(x) \Big) B_\perp = 0, \quad j = 1...n_u, \tag{10.2b}$$

for all $x \in D_c$, where $W_c(x) \doteq M_c^{-1}(x)$, $A(x) = \frac{\partial f(x)}{\partial x}$, and $B_\perp$ is a basis for the nullspace of B. The CCM also defines a controller $u : \mathcal{X} \times \mathcal{X} \times \mathcal{U} \to \mathcal{U}$, which takes the current state $x(t)$ and a state/control $x^*(t)$, $u^*(t)$ on the nominal state/control trajectory being tracked $x^* : [0, T] \to \mathcal{X}$, $u^* : [0, T] \to \mathcal{U}$, and returns a u that contracts x towards x^* at rate $\lambda_c > 0$. The controller $u(x, x^*, u^*)$ can be computed directly via $W_c(x)$ (cf. Sec. 10.3.2). If $w_x \equiv 0$, for any nominal $x^*(t)$, applying $u(x, x^*, u^*)$ renders the system closed-loop IES, i.e., $\|x(t) - x^*(t)\| \leq \alpha_c \|x(0) - x^*(0)\| e^{-\lambda_c t}$ for $\alpha_c > 0$. For bounded w_x, (10.1a) remains in a tube around $x^*(t)$; we exploit this in Sec. 10.3.2. Contraction also analyzes the convergence of state observers *Dani et al.* (2015); *Manchester and Slotine* (2014), i.e., whether a state estimate $\hat{x}(t)$ approaches the true state $x(t)$. Consider the nominal closed-loop system $\dot{x} = f(x) + Bu(\hat{x}, x^*, u^*)$ with noiseless observations $y = h(x, \theta)$ and a nominal observer

$$\dot{\hat{x}} = f(\hat{x}) + Bu(\hat{x}, x^*, u^*) + \tfrac{1}{2}\rho(\hat{x}) M_e(\hat{x}) C(\hat{x})^\top (y - h(\hat{x}, \theta)) \tag{10.3}$$

for the nominal system, where $C(x) = \frac{\partial h(x,\theta)}{\partial x}$, $\rho(x) \geq 0$ is a multiplier term, and $M_e : \mathcal{X} \to \mathbb{S}^{>0}_{n_x}$ is called an observer contraction metric (OCM), which should satisfy

$$\partial_{f+Bu} W_e(\hat{x}) + W_e(\hat{x})A(\hat{x}) + A(\hat{x})^\top W_e(\hat{x}) - \rho(\hat{x}) C(\hat{x})^\top C(\hat{x}) \leq -2\lambda_e W_e(\hat{x}) \tag{10.4}$$

for all $\hat{x} \in D_e \subseteq \mathcal{X}, u \in \mathcal{U}$. Here, $W_e(\hat{x}) = M_e^{-1}(\hat{x})$. To show that the estimated and true trajectories $\hat{x}(t)$ and $x(t)$ converge, we can analyze a nominal "meta-level" virtual system with state q *Tsukamoto and Chung* (2021), which recovers the nominal $x(t)$ and $\hat{x}(t)$ when integrated from initial conditions $q(0) = x(0)$ and $q(0) = \hat{x}(0)$:

$$\dot{q} = f(q) + Bu(\hat{x}, x^*, u^*) + \tfrac{1}{2}\rho(\hat{x}) M_e(\hat{x}) C(\hat{x})^\top (y - h(q, \theta)). \tag{10.5}$$

By setting $q = \hat{x}$, we recover the estimator dynamics (10.3); if we set $q = x$, we recover $\dot{x} = f(x) + Bu(\hat{x}, x^*, u^*)$. We can then analyze the convergence of $\hat{x}$ to x via (10.5), and *Tsukamoto and Chung* (2021) shows that if (10.4) holds, then $\hat{x}(t)$ contracts at some rate $\gamma \in (0, \lambda_e]$ towards $x(t)$. If $M_e(x)$ and $C(x)$ are constant, one can show that this holds for $\gamma = \lambda_e$, and in our experiments, we will rely on constant $M_e(x)$ and $C(x)$. In particular, $\|x(t) - \hat{x}(t)\| \leq \alpha_e \|x(0) - \hat{x}(0)\| e^{-\lambda_e t}$ for $\alpha_e > 0$, and $\hat{x}(t)$ remains in a tube around $x(t)$ if (10.3) is perturbed. For polynomial systems of moderate dimension ($n_x \lesssim 12$) with polynomial observation maps, CCMs and

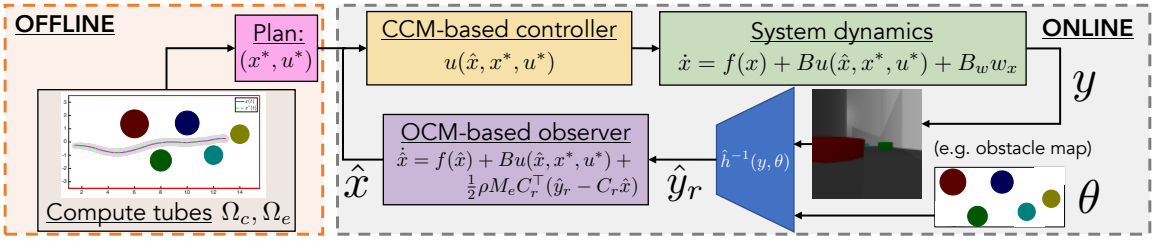


Figure 10.2: Our method. **Offline**: After learning a perception system $\hat{h}^{-1}$ (Sec. 10.3.1), we bound its error to derive tracking tubes under imperfect perception (Sec. 10.3.2). We use these tubes to find safely-trackable plans (Sec. 10.3.4). **Online**: We design a CCM/OCM-based controller/observer (Sec. 10.3.3) to track the plan/perform state estimation at runtime, using $\hat{h}^{-1}$ to process rich observations y.

OCMs can be found via convex Sum of Squares (SoS) programs *Singh et al.* (2019). CCMs/OCMs can also be found for high-dimensional non-polynomial systems via learning-based methods (e.g., *Chou et al.* (2021c); *Sun et al.* (2020)).

10.3 Method

We describe our solution to the OFMP (cf. Fig. 10.2). Using dataset $\mathcal{S}$, we first train a perception system that returns a reduced-order observation that simplifies the search for the contraction metrics (Sec. 10.3.1). Second, we bound the error of the learned perception module, and propagate this perception error bound through the system to derive bounds on the tracking and estimation error when using a CCM-/OCM-based controller/estimator (Sec. 10.3.2). Third, we obtain a CCM and OCM which optimizes this bound via SoS programming (Sec. 10.3.3). Finally, we use these bounds to constrain a planner to return trajectories that enable safe runtime tracking and robust goal reachability from observations (Sec. 10.3.4). For space, all proofs for the theoretical results are in App. C.3.

10.3.1 Learning a perception module for contraction-based estimation

Let us reconsider the observer (10.3), which updates its estimate directly using $y - h(\hat{x}, \theta)$ in the rich observation space. To implement (10.3), one can use $\mathcal{S}$ to train a deep approximation of h, denoted $\hat{h}$, design an OCM satisfying (10.4) for $C(\hat{x}) = \frac{\partial \hat{h}(\hat{x}, \theta)}{\partial x}$, and plug $\hat{h}$ and the OCM into (10.3). This naïve solution is flawed: 1) as n_y is large, learning an accurate $\hat{h}$ can be difficult; 2) the $C(\hat{x})$ in (10.4) becomes the Jacobian of a (non-polynomial) deep network, complicating OCM synthesis by precluding the use of SoS programming.

We can take a more structured approach if we know which states can be directly inferred from y; this is reasonable if the states have semantic meaning (e.g., poses, velocities). Recall C_r (Sec. 10.2.1) defines this reduced observation as $y_r = C_r x \in \mathbb{R}^{n_r}$. We can then learn an approximate inverse $\hat{h}^{-1}(y, \theta) : \mathbb{R}^{n_y} \times \mathbb{R}^{n_p} \to \mathbb{R}^{n_r}$ which maps a y and θ to the reduced observation. Note that if each unique y corresponds to a unique y_r, this inverse is well-defined and does not require the full state to be invertible

from a single y. Concretely, consider a car with position, orientation, and velocity states $[p_x, p_y, \phi, v]$ and RGB-D data from an onboard camera (Fig. 10.1.A) driving in several obstacle fields. In this case, $y_r = [p_x, p_y, \phi]^\top$ and θ could be the obstacle locations. We model $\hat{h}^{-1}$ as a neural network and train it via the mean squared error between $\hat{h}^{-1}(y_i, \theta_i)$ and $C_r x_i$ for all $i \in 1, \ldots, N_{\text{data}}$. Note that as the nominal reduced observations are roughly linear, i.e., $\hat{h}^{-1}(h(x, \theta), \theta) \approx C_r x$, this simplifies the nominal observer (10.3) to $\dot{\hat{x}} = f(\hat{x}) + Bu(\hat{x}, x^*, u^*) + \frac{1}{2}\rho(\hat{x})M_e(\hat{x})C_r^\top C_r(x - \hat{x})$, and simplifies OCM synthesis: as $C_r^\top C_r$ is constant, (10.4) is SoS-representable, despite $\hat{h}^{-1}$ being non-polynomial. Compared to the nominal reduced observer, the true observer we use,

$$\dot{\hat{x}} = f(\hat{x}) + Bu(\hat{x}, x^*, u^*) + \tfrac{1}{2}\rho(\hat{x})M_e(\hat{x})C_r^\top(\hat{h}^{-1}(h(x, \theta) + B_y w_y, \theta) - C_r\hat{x}), \quad (10.6)$$

experiences disturbance from model error $B_w w_x$, sensor noise $B_y w_y$, and learning error $\|\hat{h}^{-1}(h(x, \theta), \theta) - C_r x\|$. Quantifying these errors for our vision-based observer (10.6) is one of our core contributions and is key in deriving tracking bounds useful for planning.

10.3.2 Bounding tracking error and state estimation error for planning

To begin, assume we have a CCM M_c and an OCM M_e that are valid in $D_c \subseteq \mathcal{X}$ and $D_e \subseteq \mathcal{X}$ and which contract at rate λ_c and λ_e, respectively. We discuss CCM/OCM synthesis in Sec. 10.3.3. Define the nominal closed-loop state and virtual dynamics as:

$$\dot{x}(t) = f(x(t)) + Bu(x(t), x^*(t), u^*(t)) \quad (10.7a)$$
$$\dot{q}(t) = f(q(t)) + Bu(\hat{x}(t), x^*(t), u^*(t)) + \tfrac{1}{2}\rho(q(t))M_e(q(t))C_r^\top C_r(x(t) - q(t)) \quad (10.7b)$$

Factor the CCM/dual OCM as $M_c(x) = R_c(x)^\top R_c(x)$ and $W_e(x) = R_e(x)^\top R_e(x)$. Let $\gamma_c^t(s)$, $s \in [0, 1]$ be the geodesic between $x^*(t)$ and $x(t)$ w.r.t. M_c, and $\gamma_e^t(s)$, $s \in [0, 1]$ be the geodesic between $\hat{x}(t)$ and $x(t)$ w.r.t. W_e. *Manchester and Slotine* (2014) shows if $\gamma_c^t(s) \subseteq D_c$ for all t, s and (10.7a) is perturbed by $w_c(t)$, i.e., $\dot{x} = f(x) + Bu(x, x^*, u^*) + w_c$, the Riemannian distance w.r.t. M_c between the true and nominal state, $d_c(t) = d_c(x^*(t), x(t))$, satisfies:

$$\dot{d}_c(t) \leq -\lambda_c d_c(t) + \textstyle\int_0^1 \|R_c(\gamma_c^t(s))w_c(t)\|ds. \quad (10.8)$$

If $\gamma_e^t(s) \subseteq D_e$ for all t, s and (10.7b) is perturbed by additive $w_q(t)$ *Tsukamoto and Chung* (2021), the Riemannian distance w.r.t. W_e between the true and estimated state, $d_e(t) = d_e(x(t), \hat{x}(t))$, satisfies

$$\dot{d}_e(t) \leq -\lambda_e d_e(t) + \textstyle\int_0^1 \|R_e(\gamma_e^t(s))w_q(t)\|ds. \quad (10.9)$$

We will use (10.8) and (10.9) to obtain upper bounds on the tracking/estimation Riemannian distances, denoted as $\bar{d}_c(t)$ and $\bar{d}_e(t)$, respectively. These upper bounds

define tracking and state estimation tubes, i.e., a bound on where x and $\hat{x}$ can be, which we denote as $\Omega_c(t) = \{x \mid d_c(x^*(t), x) \leq \bar{d}_c(t)\}$ and $\Omega_e(t) = \{\hat{x} \mid d_e(x(t), \hat{x}) \leq \bar{d}_e(t)\}$, respectively. These tubes are crucial in informing where the planner can safely visit, since tracking any Ω_c-buffered candidate trajectory within D_x which remains in $\mathcal{X}_{\text{safe}}$ is guaranteed to remain safe. However, for these tubes to be usable in a planner, we need explicit bounds on the integral terms in (10.8) and (10.9).

In this section, we first present the final derived bounds on the integrals (Lemmas X.1 and X.2), describe the ideas behind the derivations, and postpone the full mathematical details to App. C.2.

Lemma X.1 ($\dot{d}_c(t)$)**.** *The integral term in* (10.8) *can be bounded as*

$$\int_0^1 \|R_c(\gamma_c^t(s))w_c(t)\|ds \leq \sqrt{\bar{\lambda}_{D_c}(M_c)}\bar{w}_x + L_{\Delta k}d_e. \tag{10.10}$$

In the second term, $L_{\Delta k}$ is the Lipschitz constant of the controller error (to be described later) which, together with state estimate error d_e, bounds the destabilizing effect of using incorrect state estimates in feedback control. This term, which can be explicitly estimated and thus concretely informs tube size in planning, is the key novelty of Lemma X.1. Overall, (10.10) states that tracking degrades with larger dynamics and estimation error.

Lemma X.2 ($\dot{d}_e(t)$)**.** *Let $\bar{\sigma}(B_y)$ denote the maximum singular value of B_y. For constant ρ and M_e, the integral in* (10.9) *simplifies to $\|R_e w_q(t)\|$ and can be bounded as:*

$$\|R_e w_q(t)\| \leq \sqrt{\bar{\lambda}(W_e)}\bar{w}_x + \tfrac{1}{2}\rho\bar{\lambda}(M_e)^{1/2}\big(L_{\hat{h}^{-1}}\sqrt{\bar{\sigma}(B_y)}\bar{w}_y + \bar{\epsilon}_{\{1,2,3\}}(x^*, \theta)\big) \tag{10.11}$$

We write Lemma X.2 for constant ρ and M_e, as this is the representation used in Sec. 10.4. Here, $L_{\hat{h}^{-1}}$ is the local Lipschitz constant of $\hat{h}^{-1}$, and $\bar{\epsilon}_{\{1,2,3\}}(x^*, \theta)$ are (spatially-varying) bounds on its error $\|\hat{h}^{-1}(h(x, \theta), \theta) - C_r x\|$, each with different strengths/weaknesses (cf. Fig. 10.4 for a visual overview). Relative to prior work, Lemma X.2 is novel as it bounds high-dimensional measurement error and learned perception module error. Overall, (10.11) states that estimation accuracy degrades with larger dynamics error, measurement error, and learned perception module error.

10.3.2.1 Bounding tracking error:

We explain more details behind Lemma X.1. As Lemma X.1 relies on a bound for $w_c(t)$, we first break down the components that make up $w_c(t)$. Relative to the nominal closed-loop dynamics (10.7a), our true closed-loop system

$$\dot{x}(t) = f(x(t)) + Bu(\hat{x}(t), x^*(t), u^*(t)) + B_w(t)w_x(t) \tag{10.12}$$

is subject to two disturbances. The first is the dynamics error $B_w(t)w_x(t)$. The second is imperfect state feedback: we apply $u(\hat{x}, x^*, u^*)$ instead of $u(x, x^*, u^*)$, which unlike the latter, may not stabilize (10.7a) at rate λ_c. Naïvely, one can bound this error by rewriting (10.12) as $\dot{x} = f(x) + Bu(\hat{x}, x^*, u^*) - Bu(x, x^*, u^*) + Bu(x, x^*, u^*) +$

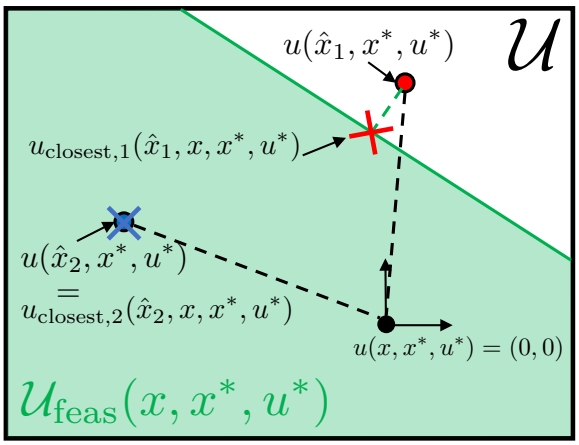


Figure 10.3: u_{closest} can be much closer to $u(\hat{x}, x^*, u^*)$ than $u(x, x^*, u^*)$: we show this for two different state estimates $\hat{x}_1$ and $\hat{x}_2$.

$B_w w_x$, where the difference between output/perfect state feedback is in red. While $\|Bu(\hat{x}, x^*, u^*) - Bu(x, x^*, u^*)\|$ is a valid disturbance bound, we can obtain a tighter bound by exploiting the structure of $u(x, x^*, u^*)$. In general, many $u_{\text{fb}} \in \mathcal{U}$ can make $\dot{x} = f(x) + B(u^* + u_{\text{fb}})$ contract at rate λ_c towards x^*, w.r.t. M_c. Define $\dot{x}^* = f(x^*) + Bu^*$; then, the contracting u_{fb} *Singh et al.* (2019) are defined by a linear inequality constraint,

$$\mathcal{U}_{\text{feas}}(x, x^*, u^*) = \{u_{\text{fb}} \mid \gamma_{c,s}^\top(1) M_c(x)\dot{x} - \gamma_{c,s}^\top(0) M_c(x^*)\dot{x}^* \leq -\lambda_c d_c(x^*, x)^2\}, \tag{10.13}$$

where $\gamma_{c,s}(\cdot) = \frac{\partial \gamma_c(\cdot)}{\partial s}$. As in *Singh et al.* (2019), we select the minimum-norm feasible control to be $u(\hat{x}, x^*, u^*)$, i.e., $u(\hat{x}, x^*, u^*) = \arg\min_{u \in \mathcal{U}_{\text{feas}}(\hat{x}, x^*, u^*)} \|u\|$. Then, using $\mathcal{U}_{\text{feas}}$, we can rewrite (10.12) as $\dot{x} = f(x) + B(u(\hat{x}, x^*, u^*) - u_{\text{closest}} + u_{\text{closest}}) + B_w w_x$, where $u_{\text{closest}}(\hat{x}, x, x^*, u^*) \doteq \arg\min_{u \in \mathcal{U}_{\text{feas}}(x, x^*, u^*)} \|u - u(\hat{x}, x^*, u^*)\|$ is the closest control input to $u(\hat{x}, x^*, u^*)$ that contracts the nominal dynamics at x. Bounding the imperfect state feedback as $\|Bu(\hat{x}, x^*, u^*) - Bu_{\text{closest}}\|$ instead of $\|Bu(\hat{x}, x^*, u^*) - Bu(x, x^*, u^*)\|$ can be far tighter, as $u(\hat{x}, x^*, u^*)$ may still contract the system at rate λ_c (Fig. 10.3: $\hat{x}_2$ case), or there can be a contracting u closer to $u(\hat{x}, x^*, u^*)$ than $u(x, x^*, u^*)$ (Fig. 10.3: $\hat{x}_1$ case). Combining with the dynamics error, we can write w_c:

$$w_c(t) \doteq Bu(\hat{x}(t), x^*(t), u^*(t)) - Bu_{\text{closest}}(t) + B_w(t) w_x(t) \tag{10.14}$$

As (10.14) still depends on x and $\hat{x}$, which are unknown at planning time, extra steps must be taken to obtain a useful bound that is independent of x and $\hat{x}$; we achieve this by bounding the first two terms of (10.14) via a Lipschitz constant. Define $\Delta k(\hat{x}, x, x^*, u^*) = \max_{s \in [0,1]} \|R_c(\gamma_c^t(s)) B(u(\hat{x}, x^*, u^*) - u_{\text{closest}})\|$, and $L_{\Delta k}$ as its local Lipschitz constant in the first argument, i.e., for all $x^* \in D$, $u^* \in \mathcal{U}$, $\{x \mid d_c(x^*, x) \leq \bar{c}\}$, and $\{\hat{x} \mid d_e(x, \hat{x}) \leq \bar{e}\}$ for predetermined $\bar{c}, \bar{e} > 0$ (adjustable based on the expected error),

$$|\Delta k(\hat{x}_1, x, x^*, u^*) - \Delta k(\hat{x}_2, x, x^*, u^*))| \leq L_{\Delta k} d_e(\hat{x}_1, \hat{x}_2). \tag{10.15}$$

See Rem. X.4 for details on estimating $L_{\Delta k}$. In estimating $L_{\Delta k}$, we measure input

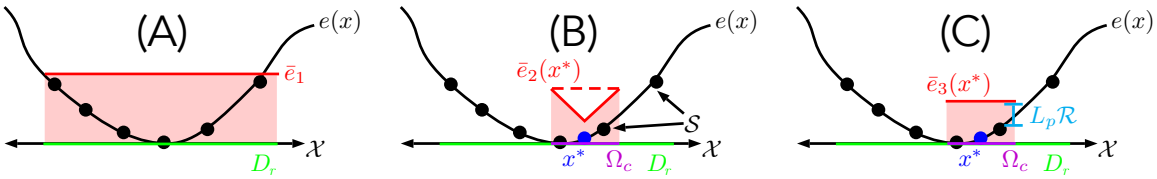


Figure 10.4: Our perception error bounds. (A) $\bar{\epsilon}_1$ is simple but conservative. B) $\bar{\epsilon}_2(x^*)$ is tighter, as it only seeks to be valid over the tube Ω_c. However, it scales linearly with the size of Ω_c. C) $\bar{\epsilon}_3(x^*)$ can be tighter for larger Ω_c by adding a Lipschitz-based buffer to the largest training error in Ω_c.

distances w.r.t. W_e; this reduces conservativeness due to the form of our estimation error bound. Combining (10.14)-(10.15) yields Lemma X.1; see App. C.3 for the detailed proof.

10.3.2.2 Bounding estimation error:

Now, we provide more details behind Lemma X.2. To bound $\int_0^1 \|R_e(\gamma_e^t(s))w_q(t)\|ds$, we first note that $\|w_q\|$ is bounded by the sum of the disturbance magnitudes when $q = x$ and when $q = \hat{x}$ *Tsukamoto and Chung* (2021). If $q = x$, (10.7b) becomes $\dot{x} = f(x) + Bu(\hat{x}, x^*, u^*)$; relative to the true closed-loop dynamics (10.12), the disturbance is $B_w(t)w_x(t)$. If instead $q = \hat{x}$, (10.7b) becomes $\dot{\hat{x}} = f(\hat{x}) + Bu(\hat{x}, x^*, u^*) + \frac{1}{2}\rho(\hat{x})M_e(\hat{x})C_r^\top C_r(x - \hat{x})$; relative to the true observer (10.6), the disturbance is

$$w_e(t) \doteq \tfrac{1}{2}\rho(\hat{x}(t))M_e(\hat{x}(t))C_r^\top \big(\hat{h}^{-1}\big(h(x(t),\theta) + B_y w_y(t), \theta\big) - C_r x(t)\big). \tag{10.16}$$

Two errors drive $w_e(t)$: the perception error $\hat{h}^{-1}(h(x,\theta),\theta) - C_r x$, and the runtime observation noise $B_y w_y$. Combining with the dynamics error gives $w_q(t) \doteq B_w(t)w_x(t) + w_e(t)$. $B_w(t)w_x(t)$ can be bounded as in Lemma X.1, but $w_e(t)$ is harder to bound. Let $y_p = h(x,\theta)$ and $y = h(x,\theta) + B_y w_y$. We rewrite the norm of the red term in (10.16) as:

$$\begin{aligned} \|\hat{h}^{-1}\big(y,\theta\big) - C_r x\| &= \|\hat{h}^{-1}\big(y,\theta\big) - \hat{h}^{-1}\big(y_p,\theta\big) + \hat{h}^{-1}\big(y_p,\theta\big) - C_r x\| \\ &\leq \underbrace{L_{\hat{h}^{-1}}\|B_y w_y\|}_{\text{from measurement noise}} + \underbrace{\|\hat{h}^{-1}\big(y_p,\theta\big) - C_r x\|}_{\text{from learning error} \doteq \epsilon(x,\theta)}. \end{aligned} \tag{10.17}$$

Here, $L_{\hat{h}^{-1}}$ is the local Lipschitz constant of the learned inverse function in y, i.e.,

$$\|\hat{h}^{-1}(\tilde{y},\theta) - \hat{h}^{-1}(\check{y},\theta)\| \leq L_{\hat{h}^{-1}}\|\tilde{y} - \check{y}\|, \quad \forall \tilde{y}, \check{y} \in D_y \oplus \mathcal{Y}_d, \quad \forall \theta \in D_\theta, \tag{10.18}$$

where $D_y = h(D_r, D_\theta)$ is the image of the training data domains, $\oplus$ is the Minkowski sum, and $\mathcal{Y}_d = \{B_y w_y \mid \|w_y\| \leq \bar{w}_y\}$ is the set of feasible measurement noise. The first braced term in (10.17) bounds the effect of measurement error on the reduced observation and is valid for all $(x,\theta) \in D_r \times D_\theta$ and observation noise satisfying $\|w_y\| \leq \bar{w}_y$.

Now, consider the second braced term in (10.17). How can we bound the learned

perception module error $\epsilon(x,\theta) \doteq \|\hat{h}^{-1}\big(h(x,\theta),\theta\big) - C_r x\|$ over $D_r \times D_\theta$? We describe three options (Fig. 10.4) at a high level, highlight their strengths/drawbacks, and provide the details in App. C.2. The first bound, denoted $\bar{\epsilon}_1$, is a constant bound on $\epsilon(x,\theta)$ globally over $D_r \times D_\theta$ (Fig. 10.4.A). This works well if the error is consistent, but is loose if there are any error spikes. The second bound (Fig. 10.4.B), denoted $\bar{\epsilon}_2(x^*,\theta)$, bounds the error only in the tube Ω_c around a nominal x^*, using the Lipschitz constant of $\epsilon(x,\theta)$ (denoted L_p). Due to its locality, $\bar{\epsilon}_2(x^*,\theta)$ can be tighter than $\bar{\epsilon}_1$; however, it scales linearly with the size of Ω_c, even if $\epsilon(x,\theta)$ remains constant. The third bound, $\bar{\epsilon}_3(x^*,\theta)$ (Fig. 10.4.C), also bounds the error in the tube but avoids the linear scaling by taking the worst training error in Ω_c and buffering it with a *constant* value, which depends on L_p and the dataset dispersion $\mathcal{R}$. Each of these bounds $\bar{\epsilon}_{\{1,2,3\}}$ on $\epsilon(x,\theta)$ can be plugged into Lemma X.2 to upper bound $\epsilon(x,\theta)$; see App. C.2 for details.

10.3.2.3 Integrating the differential inequalities:

Now that we can bound the RHSs of the differential inequalities (10.8) and (10.9) via Lemmas X.1 and X.2, we show how these bounds on $\dot{d}_c$ and $\dot{d}_e$ bound the values of d_c and d_e, thereby providing the desired tubes. By grouping terms in (10.8)-(10.9), we have the following affine vector-valued differential inequality,

$$\begin{bmatrix}\dot{d}_c\\ \dot{d}_e\end{bmatrix} \leq \begin{bmatrix}-\lambda_c & L_{\Delta k}\\ (*) & -\lambda_e\end{bmatrix}\begin{bmatrix}d_c\\ d_e\end{bmatrix} + \begin{bmatrix}\sqrt{\bar{\lambda}_{D_c}(M_c)}\bar{w}_x\\ \sqrt{\bar{\lambda}(W_e)}\bar{w}_x + \frac{\rho}{2}\bar{\lambda}(M_e)^{1/2}\big(L_{\hat{h}^{-1}}\bar{w}_y + \bar{\epsilon}_{\{1,\tilde{2},3\}}(x^*,\theta)\big)\end{bmatrix}, \tag{10.19}$$

where we regroup the terms for $\bar{\epsilon}_2(x^*,\theta)$ as $\bar{\epsilon}_{\tilde{2}}(x^*,\theta) \doteq \bar{\epsilon}_2(x^*,\theta) - L_p d_c/\sqrt{\underline{\lambda}_{D_c}(M_c)}$, and $(*) = 0.5 L_p \rho \sqrt{\bar{\lambda}(M_e)/\underline{\lambda}_{D_c}(M_c)}$ if using $\bar{\epsilon}_2$ and 0 else. Then, we have this result:

Theorem X.3 (From derivative to value)**.** *Let RHS denote the right hand side of* (10.19)*. Given bounds on the Riemannian distances at $t = 0$: $d_c(0) \leq \bar{d}_c(0)$ and $d_e(0) \leq \bar{d}_e(0)$, upper bounds $\bar{d}_c(t) \geq d_c(t)$ and $\bar{d}_e(t) \geq d_e(t)$ for all $t \in [0,T]$ can be written as*

$$\begin{bmatrix}d_c(t)\\ d_e(t)\end{bmatrix} \leq \int_{\tau=0}^{t} RHS\Big(\tau, \begin{bmatrix}d_c\\ d_e\end{bmatrix}\Big)d\tau \doteq \begin{bmatrix}\bar{d}_c(t)\\ \bar{d}_e(t)\end{bmatrix}, \quad d_c(0) = \bar{d}_c(0),\ d_e(0) = \bar{d}_e(0). \tag{10.20}$$

Evaluating the integral in (10.20) is efficient as RHS is affine, so $\bar{d}_c$ and $\bar{d}_e$ can be readily used in planning (cf. Sec. 10.3.4). However, note that these tubes are only *locally valid*, e.g., evaluating the tubes outside of D_x will give incorrect values. We detail a set of validity conditions in Sec. 10.3.4, prove their sufficiency in Thm. X.5, use them in our planner, and show in Sec. 10.4 that a baseline that ignores these conditions is unsafe. Finally, we close with a remark on how we estimate the constants in the bounds.

Remark X.4 (Estimating constants from data)*.* The derived bounds depend on several constants that are unknown *a priori*, such as $L_{\Delta k}$ and $L_{\hat{h}^{-1}}$, and if $\bar{\epsilon}_1$, $\bar{\epsilon}_2$, or $\bar{\epsilon}_3$

is being used, $\bar{\epsilon}_1$, L_p, and $\{L_p, \mathcal{R}\}$ also need to be estimated, respectively. As over-approximating each constant also yields valid (and looser) bounds, we use the i.i.d. validation set $\mathcal{V}$ to overestimate each constant via a sampling-based approach based on extreme value theory *Chou et al.* (2021c). This returns a value which overestimates the true constant with a user-desired probability δ, where δ holds in the limit of infinite samples. See *Chou et al.* (2021c); *Knuth et al.* (2021a); *Weng et al.* (2018) for details.

10.3.3 Optimizing CCMs and OCMs for output feedback

We briefly discuss how we obtain the CCM/OCM that define the controller/observer; for space, we detail our method in App. C.4. We write two SoS programs to independently synthesize the CCM/OCM, which are approximately optimized to minimize their tube sizes. We search over polynomial CCMs and constant OCMs. For polynomial dual CCMs $W_c(x)$, we also find a constant metric $\bar{W}_c \succeq W_c(x)$, for all x, in order to simplify constraint checking in Sec. 10.3.4. For linear systems, these SoS programs simplify to a standard semidefinite program (SDP), which scale to higher-dimensional systems.

10.3.4 Solving the OFMP

Algorithm X.1: **C**ontraction-based **O**utput feedback **RRT** (CORRT)

Input: x_I, $\mathcal{G}$, θ, $\mathcal{S}$, training error $\{e_i\}_{i=1}^{N_{\text{data}}}$, estimated constants, $\bar{d}_c(0)$, $\bar{d}_e(0)$, $\bar{c}$, $\bar{e}$

$\mathcal{T} \leftarrow \{(x_I, \bar{d}_c(0), \bar{d}_e(0), 0)\}$ `// node: state, CCM/OCM Riem. dist. bound, time`

$\mathcal{P} \leftarrow \{(\emptyset, \emptyset)\}$ `// parent: previous control/dwell time`

while True **do**

- $(x_{\text{n}}, \bar{d}_c^n, \bar{d}_e^n, t_{\text{n}}) \leftarrow$ `SampleNode(`$\mathcal{T}$`)` `// sample node from tree`
- $(u_{\text{p}}, t_{\text{p}}) \leftarrow$ `SampleProposedControl ()` `// sample ctrl/dwell time`
- $(x_{\text{p}}^*(t), u_{\text{p}}^*(t)), t \in [t_n, t_n + t_p) \leftarrow$ `IntegrateDyn` $(x_{\text{n}}, u_{\text{p}}, t_{\text{p}})$ `// get extension`
- $(\bar{d}_c^n(t), \bar{d}_e^n(t)), t \in [t_n, t_n + t_p) \leftarrow$ `ErrBnd` $(\bar{d}_c^n, \bar{d}_e^n, x_{\text{p}}^*(t), u_{\text{p}}^*(t), \mathcal{S}, \{e_i\}, \theta)$ `// new tube`
- $(b_L^c, b_L^e) \leftarrow (\bar{d}_c^n(t) \leq \bar{c}, \bar{d}_e^n(t) \leq \bar{e}), \forall t \in [t_n, t_n + t_p)$ `// check upper bound`
- $b_c \leftarrow \Omega_c^n(t) \subseteq (D_c \cap D_r \cap \mathcal{X}_{\text{safe}}), \forall t \in [t_{\text{n}}, t_{\text{n}} + t_{\text{p}})$ `// check tracking tube`
- $b_e \leftarrow \Omega_c^n(t) \oplus (\Omega_e^n(t) \ominus \{x(t)\}) \subseteq (D_e \cap D_c), \forall t \in [t_{\text{n}}, t_{\text{n}} + t_{\text{p}})$ `// chk. estimator tube`
- **if** $b_L^c \wedge b_L^e \wedge b_c \wedge b_e$ **then** $\mathcal{T} \leftarrow \mathcal{T} \cup \{(x_c^*(t_{\text{n}} + t_{\text{p}}), \bar{d}_c^n(t_{\text{n}} + t_{\text{p}}), \bar{d}_e^n(t_{\text{n}} + t_{\text{p}}), t_{\text{p}})\}$; $\mathcal{P} \leftarrow \mathcal{P} \cup \{(u_{\text{p}}, t_{\text{n}} + t_{\text{p}})\}$
- **else** continue `// add extension if all checks pass`
- **if** $\exists t, \Omega_c^n(t) \subseteq \mathcal{G}$ **then** break; return plan `// return if in` $\mathcal{G}$

Given the CCM, OCM, and the ability to compute tracking tubes, we can now solve the OFMP. Our solution builds upon a kinodynamic RRT *LaValle* (2006),

though we note that the tubes derived in Sec. 10.3.2 are planner-agnostic. We grow a search tree $\mathcal{T}$ by integrating sampled controls held for sampled dwell-times until $\mathcal{G}$ is reached. To ensure we stay in $\mathcal{X}_{\text{safe}}$ at runtime, we impose extra constraints on each candidate transition, which are informed by the tubes; this translates to a restriction on where $\mathcal{T}$ can grow (cf. Fig. 10.5).

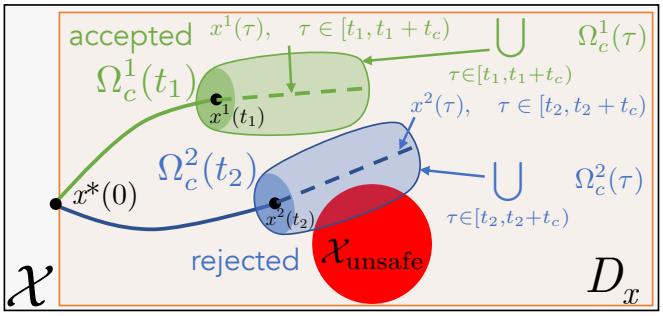


Figure 10.5: Visualization of Alg. X.1.

To use the Riemannian distance bounds $\bar{d}_c(t)$ and $\bar{d}_e(t)$ from (10.20) in planning, recall that these bounds define sets centered around $x^*(t)$ and $x(t)$, $\Omega_c(t)$ and $\Omega_e(t)$, which x and $\hat{x}$ are guaranteed to remain within. We can use these sets for collision and constraint checking. If the metric defining $\Omega(t)$ is constant, each $\Omega(t)$ defines an ellipsoid, i.e., $\Omega_c(t) = \{x(t) \mid (x(t)-x^*(t))^\top M_c(x(t)-x^*(t)) \leq \bar{d}_c(t)^2\}$ and $\Omega_e(t) = \{\hat{x}(t) \mid (\hat{x}(t)-x(t))^\top W_e(\hat{x}(t)-x(t)) \leq \bar{d}_e(t)^2\}$. If the metric is state-dependent (as is the case for some CCMs we use), we can use $\bar{W}_c$ (see Sec. 10.3.3) to obtain an ellipsoidal outer approximation of $\Omega_c(t)$: $\Omega_c(t) \subseteq \{x(t) \mid (x(t)-x^*(t))^\top (\bar{W}_c)^{-1}(x(t)-x^*(t)) \leq \bar{d}_c(t)^2\} \doteq \tilde{\Omega}_c(t)$ that can ease constraint checking. Thus, we can guarantee *at planning time* that *in execution*, $x(t) \in \tilde{\Omega}_c(t)$, and $\hat{x}(t) \in \tilde{\Omega}_c(t) \oplus (\Omega_e(t) \ominus \{x(t)\})$, where $A \ominus B \doteq \{x-y \mid x \in A, y \in B\}$. As (10.20) defines Ω for *any* nominal trajectory, we can quickly compute tubes along all edges in $\mathcal{T}$. For instance, suppose we wish to extend from some node in $\mathcal{T}$, $x_n^*(t_n)$, which satisfies $d_c^n(t_n) \leq \bar{d}_c^n(t_n)$ and $d_e^n(t_n) \leq \bar{d}_e^n(t_n)$, to a candidate state $x_n^*(t_n+t_p)$ by applying control u over $[t_n, t_n+t_p)$. Then, using (10.20), we can obtain $\bar{d}_c^n(t)$ and $\bar{d}_e^n(t)$, for all $t \in [t_n, t_n+t_p)$, and to remain collision-free in execution, we require the induced $\tilde{\Omega}_c(t) \subseteq \mathcal{X}_{\text{safe}}$; we check this in line 9 of our planner, Alg. X.1. Here, we assume obstacles are inflated to account for robot geometry.

To remain collision-free at runtime, we must add extra constraints on $\mathcal{T}$ to ensure the tubes are valid, as discussed in Sec. 10.3.2. We describe these constraints now, and prove they are sufficient in Thm. X.5. At a high level, the estimated constants, CCM, and OCM must be valid for any x and $\hat{x}$ that can be reached at runtime. Thus, in line 8, we ensure $d_c(t)$ and $d_e(t)$ remain less than $\bar{c}$ and $\bar{e}$ for all time, so that $L_{\Delta k}$ (10.15) is valid. In line 9, we ensure that $\Omega_c(t) \subseteq D_c \cap D_r$, i.e., the system remains where the controller can contract x towards x^*, and $\bar{\epsilon}_i$ is valid. In line 10, we ensure $\hat{x}$ remains in $D_e \cap D_c$; this ensures that (10.6) contracts towards the true state x via (10.2), and that a feasible feedback control exists in (10.13); ensuring this at planning time (when we only know $x^*(t)$) requires a Minkowski sum of Ω_c and $\Omega_e \ominus \{x(t)\}$. Constraint-satisfying candidate extensions are added to $\mathcal{T}$ (line 11); else, they are rejected (line 12). This continues until the goal is reached (line 13). We visualize our planner (Fig. 10.5), **C**ontraction-based **O**utput feedback **RRT** (CORRT), detailed in Alg. X.1. Finally, Thm. X.5 shows our method ensures safety and goal reachability if all estimated constants are valid; as our estimates are probabilistically-valid, the

	CORRT trk. err.	CORRT est. err.	B1 trk. err.	B1 est. err.	B2 trk. err.	B2 est. err.	B3 trk. err.	B3 est. err.
Car	0.175 ± 0.117	0.032 ± 0.022	17.49 ± 79.86	143.4 ± 1202	1.520 ± 6.306	3.597 ± 19.90	—	—
Quad	0.151 ± 0.187	0.029 ± 0.028	39.30 ± 142.1	52.64 ± 185.9	40.56 ± 302.1	63.53 ± 424.1	—	—
Arm	2.0e-4 ± 1.3e-5	0.053 ± 0.039	2.0e-4 ± 1.4e-5	0.145 ± 0.239	—	—	0.000 ± 0.000	0.316 ± 0.249

Table 10.1: Statistics on the tracking/estimation error reduction across all experimental results. "Trk. err." $= \|x^*(T) - x(T)\|/\|x^*(0) - x(0)\|$. "Est. err." $= \|\hat{x}(T) - x(T)\|/\|\hat{x}(0) - x(0)\|$. In each cell: average error ± standard deviation over all trials.

overall guarantees are probabilistic (cf. Rem. C.8):

Theorem X.5 (CORRT correctness)**.** *Assume that $L_{\Delta k}$, $L_{\hat{h}^{-1}}$, and the estimated constants in $\bar{\epsilon}_{\{1,2,3\}}$ are valid over their computed domains. Then Alg. X.1 returns a trajectory $(x^*(t), u^*(t))$, which when tracked on the true system* (10.1a) *using $u(\hat{x}, x^*, u^*)$ with state estimates $\hat{x}$ generated by* (10.6)*, reaches $\mathcal{G}$ while satisfying $x(t) \in \mathcal{X}_{safe}$, for all $t \in [0, T]$.*

10.4 Results

We evaluate CORRT on a 4D car with RGB-D observations, a 6D quadrotor with RGB observations, and a 14D acceleration-controlled 7DOF arm with RGB observations. All observations are rendered in PyBullet. We compare with three baselines; two are shared across experiments, so we overview them here. To show the need to plan where the CCM/OCM are valid and the error bounds are accurate, Baseline 1 (B1) plans using the tracking tubes from (10.20) inside Alg. X.1 but is *not constrained* to stay within D, i.e., the checks in line 8-10 of Alg. X.1 are relaxed. To show the need to consider estimation error in planning, Baseline 2 (B2) assumes perfect state knowledge in computing its tubes, i.e., $d_e(t) \equiv 0$. All baselines execute with the same CCM/OCM as our method. See Table 10.1 for error statistics and the video `http://tinyurl.com/wafr22corrt` for visualizations.

10.4.0.1 4D nonholonomic car

We consider a ground vehicle in an obstacle field (Fig. 10.1.A), governed by (C.15). The observations are given by 48x48 RGB-D images taken from a front-facing onboard camera (Fig. 10.1.A, inset); this makes $y \in \mathbb{R}^{9216}$. Three states can be directly inferred from a single image: p_x, p_y, and ϕ. For this example, $\theta \in \mathbb{R}^5$ parameterizes the p_y-translation of each of the five obstacles. We are given $N_{\text{data}} = 250000$ datapoints to train the perception system $\hat{h}^{-1}$, sampled uniformly from $C_r D_p = [0, 13.5] \times [-2.5, -2.5] \times [-\pi/3, \pi/3]$ and $D_\theta = [0.5, 1.5] \times [-1.5, -0.5] \times [0.5, 1.5] \times [-1, 0] \times [0, 1]$. We model $\hat{h}^{-1}$ as a fully-connected neural network, with five hidden layers of width 1024 and softplus activations. We use the method of Sec. 10.3.3 to obtain a constant CCM M_c with $\bar{\lambda}(M_c) = 1$, $\underline{\lambda}(M_c) = 0.07$, and $\lambda_c = 2.5$, and a constant OCM M_e with $\bar{\lambda}(W_e) = 5.44$, $\underline{\lambda}(W_e) = 0.05$, and $\lambda_o = 0.6$, where $D_c = (-\infty, \infty)^2 \times [-\pi/3, \pi/3] \times [2, 5] = D_e$. To compute our tubes in CORRT, we use $\bar{\epsilon}_3(x^*, \theta)$, since for this example Ω_c may be large. The constants are estimated to be $L_{\Delta k} = 3.28$, $L_{\hat{h}^{-1}} = 0.05$, $L_p = 0.024$, and $\mathcal{R} = 0.69$. In computing our tubes, we assume $\|w_x\| \leq 0.05$, $\bar{d}_c(0) = 0.2$, $\bar{d}_e(0) = 0.1$, and $w_y \in \mathbb{R}^{n_y}$ satisfies $\|w_y\| \leq 0.25$.

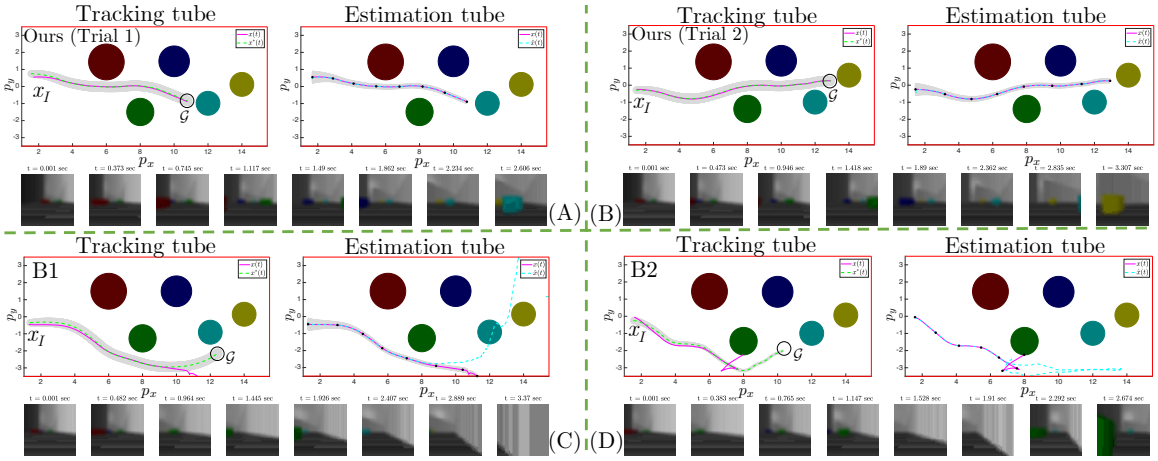


Figure 10.6: 4D car. Planned, executed, and estimated trajectories, overlaid with corresponding tracking and estimation tubes $\Omega_c(t)$ and $\Omega_e(t)$. For eight timesteps corresponding to the black dots on the Ω_e plot, we also show RGB component of the observations seen at runtime (bottom). A) and B): two examples of CORRT, which safely reach the goal. C) and D): B1 and B2: both crash.

To simulate noisy depth images, B_y is set to be a diagonal $n_y \times n_y$ matrix, with 0 diagonal entries for RGB indices and 1 for the depth indices.

We plan for 150 start/goals in D; our unoptimized implementation takes 2.5 minutes on average. This is done offline; the tracking controller is computed at real-time rates following Sec. 10.3.2.1 and *Singh et al.* (2019). For each trial, the obstacle map θ is selected uniformly within D_θ. See Table 10.1 for error statistics. Over all trials, our method ensures $x(t)$ and $x^*(t)$ always remain within the CORRT-computed $\Omega_c(t)$ and $\Omega_e(t)$, respectively, and reduces the initial tracking/estimation error by a factor of > 5 and 30, respectively. In contrast, B1 violates its $\Omega_c(t)$ and $\Omega_e(t)$ in 90/150 and 101/150 trials, respectively, fails to reduce tracking/estimation error, and can crash. For instance, in Fig. 10.6.C, the plan leaves D_r, causing observation error to increase (here, $\hat{h}$ is inaccurate, since it is not trained outside of D_r), destabilizing $\hat{x}$ (Fig. 10.6.C, right), in turn destabilizing x, leading to the crash. Similarly, B2 violates its computed Ω_c in 60/150 trials (no $\Omega_e(t)$ is computed for B2, as it assumes perfect state information), fails to shrink tracking/estimation errors, leading to crashes (see Fig. 10.6). As in B1, this crash also arises from observation error. Overall, this experiment suggests that CORRT enables safe goal-reaching for nonholonomic systems using RGB-D data, and that it generalizes to different environments (i.e., obstacle layouts), while baselines are unsafe.

10.4.0.2 6D quadrotor

We consider a planar quadrotor in an obstacle field (Fig. 10.1.B), governed by (C.16). The observations are given by 48x48 RGB images taken from a front-facing onboard camera (Fig. 10.1.B, inset); this makes $y \in \mathbb{R}^{6912}$. Three states can be directly inferred from an image: p_x, p_z, and ϕ. Here, we consider a single set of map configurations, i.e., θ is a singleton. We are given $N_{\text{data}} = 140000$ datapoints to train

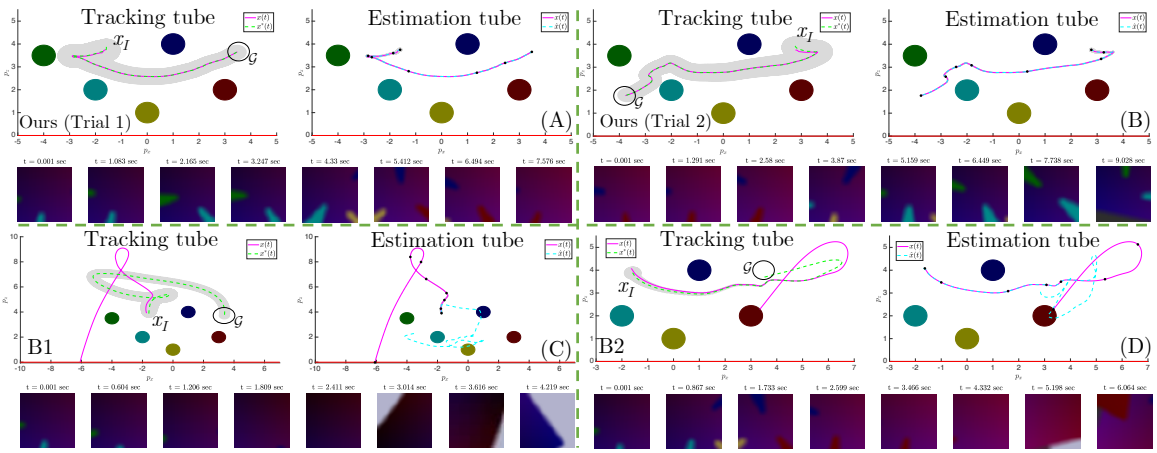


Figure 10.7: 6D quadrotor. Planned, executed, and estimated trajectories, overlaid with $\Omega_c(t)$ and $\Omega_e(t)$. Snapshots of the runtime observations are shown (bottom). A) and B): two examples of CORRT, which safely reach the goal. C) and D): B1 and B2: both crash.

$\hat{h}^{-1}$, sampled uniformly from $C_r D_p = [-4.5, 4.5] \times [0.5, 4.5] \times [-\pi/4, \pi/4]$. We model $\hat{h}^{-1}$ as a fully-connected neural network, with five hidden layers of width 1024 and ReLU activations. Using the method of Sec. 10.3.3, we obtain a polynomial CCM M_c with $\bar{\lambda}_{D_c}(M_c) = 6.55$, $\underline{\lambda}_{D_c}(M_c) = 0.22$, and $\lambda_c = 0.8$, and a constant OCM M_e with $\bar{\lambda}(W_e) = 8.13$, $\underline{\lambda}(W_e) = 0.1$, and $\lambda_e = 0.7$, where $D_c = (-\infty, \infty)^2 \times [-\pi/3, \pi/3] \times [-4.5, 4.5] \times [-1, 1] \times [-2, 2]$ and $D_e = (-\infty, \infty)^2 \times [-\pi/4, \pi/4] \times [-5, 5] \times [-2.5, 2.5] \times [-2.5, 2.5]$. To update our tracking tubes in CORRT, we found it sufficient to use the first error bound $\bar{\epsilon}_1$, which we estimate to be 0.008, and $L_{\Delta k} = 3.6$. In computing our tubes, we assume $\|w_x\| \leq 0.0125$, $\bar{d}_c(0) = 0.15$, $\bar{d}_e(0) = 0.1$, and noiseless images $\|w_y\| = 0$.

We plan for 150 start/goals in D, taking 6 minutes on average (see Table 10.1 for statistics). Across all trials, CORRT ensures $x(t)$ and $\hat{x}(t)$ stay inside the CORRT-computed tubes $\Omega_c(t)$ and $\Omega_e(t)$, respectively, and reduces the initial tracking/estimation error by a factor of > 6 and 34. In contrast, B1 violates its computed $\Omega_c(t)$ and $\Omega_e(t)$ in 61/150 and 76/150 trials, respectively, fails to reduce error, and can be unsafe (see Fig. 10.7). Similarly, B2 violates its Ω_c in 142/150 trials. We show concrete examples of this in Fig. 10.7.C-.D; the plans in both cases exit D_r, moving to p_x and p_z values outside of the $[-4.5, 4.5] \times [0.5, 4.5]$ training range, leading to high $\hat{h}^{-1}$ error. The plans also take overly-aggressive turns that bring the velocities outside of D_e and D_c; this further destabilizes the system, causing crashes in both cases. Overall, this experiment suggests the need to ensure that $\hat{h}^{-1}$, the CCM, and the OCM are correct, and that CORRT enables this to guarantee safety with high probability for underactuated systems via RGB observations.

10.4.0.3 17D manipulation task

We consider an acceleration-controlled 7DOF Kuka arm, where each joint follows double integrator dynamics (C.18), which is grasping an object (a rubber duck) with

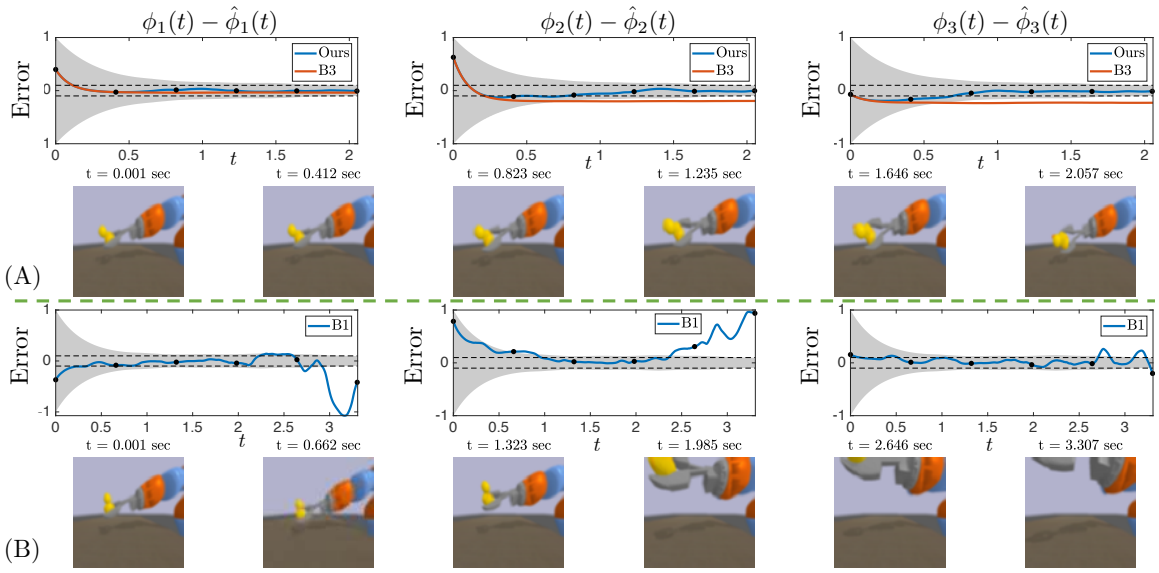


Figure 10.8: 7DOF arm. State estimate error, overlaid with $\Omega_e(t)$ (in gray). Runtime observations are shown (bottom). A): when using CORRT, the state estimate error remains in $\Omega_e(t)$ and achieves $|\hat{\phi}_i(T) - \phi_i(T)| \leq 0.1$. B3 fails to meet this requirement. B) B1 also fails the 0.1 requirement.

an unknown orientation relative to the end effector. We assume slight noise in the dynamics (C.18), $\bar{w}_x = 0.0125$, due to the weight of the object. Our goal is to estimate the unknown orientation, represented as three Euler angles $\{\phi_i\}_{i=1}^3$, using our observer (10.6), given 80x80 RGB images (Fig. 10.1.C) of the arm and grasped object (see Fig. 10.1.C, inset); this makes $y \in \mathbb{R}^{19200}$. We may also plan motions for the arm to improve the quality of the observations/state estimates, though in doing so, we also need to counteract the dynamics error. Our overall goal is to guarantee with high probability that our final estimate of the relative orientation satisfies $|\phi_i(T) - \hat{\phi}_i(T)| \leq 0.1$, $i = 1, 2, 3$.

We assume that the joint angles and velocities can be perfectly estimated (i.e., directly measured), given the accuracy of the Kuka joint encoders, focusing instead on estimating the unknown $\{\phi_i\}_{i=1}^3$ and controlling j and $\dot{j}$ (the joint angles and velocities) using our method. We assume the object is rigidly attached to the gripper, such that its relative orientation is constant over time. Combining $\{\phi_i\}_{i=1}^3$ and the 14D model, the full state of the system is 17D (C.17), i.e., $x = [\phi_1, \phi_2, \phi_3, j_1, \ldots, j_7, \dot{j}_1, \ldots \dot{j}_7]^\top$. To train $\hat{h}^{-1}$, we note that $\{\phi_i\}_{i=1}^3$ can all be estimated from the image. For this example, since j is known and affects the generated y, we design $\hat{h}^{-1}$ to take as input $y \in \mathbb{R}^{19200}$ and $j \in \mathbb{R}^7$ (i.e., j plays the role of θ) and to output $\{\phi_i\}_{i=1}^3$. We are given $N_{\text{data}} = 62500$ datapoints to train $\hat{h}^{-1}$, where $\{\phi_i\}_{i=1}^3$ are sampled uniformly from $[-\pi/3, \pi/3]^3$ and j is sampled uniformly from $[-0.05, 0] \times [0, 0.05] \times [0.15, 0.32] \times [-1.83, -1.69] \times [-0.05, 0.05]^2 \times [-\pi/3, \pi/3]$. We model $\hat{h}^{-1}$ as a fully-connected neural network, with five hidden layers of width 1024 and softplus activations. We compute a constant CCM for the 14D subsystem: CCM synthesis for the full 17D system fails, as the $\{\phi_i\}_{i=1}^3$ are not controllable due to the rigid attachment. Since the arm dynamics are linear, the CCM optimization

simplifies to a standard semidefinite program that can be quickly solved. We compute a constant OCM for the full 17D system, to enable estimation of $\{\phi_i\}_{i=1}^3$. Using the method of Sec. 10.3.3, we obtain a CCM M_c with $\bar{\lambda}(M_c) = 100$, $\underline{\lambda}(M_c) = 2.81$, and $\lambda_c = 2.89$, and a constant OCM M_e with $\bar{\lambda}(W_e) = \underline{\lambda}(W_e) = 0.1$, and $\lambda_e = 9.5$. As the dynamics are linear, a constant CCM/OCM holds globally, i.e., $D_e = D_c = \mathcal{X}$. To update the tubes in CORRT, we use $\bar{\epsilon}_2(x^*, j)$, where we estimate $L_p = 2.45$. Since j and $\dot{j}$ are known, no error arises from incorrect state estimates; thus, $L_{\Delta k}$ does not need to be estimated. We assume $\bar{d}_c(0) = 10^{-3}$, $\bar{d}_e(0) = 0.32$, and noiseless images $\|w_y\| = 0$.

We plan 100 trajectories in D from various initial j, $\dot{j}$, and orientation estimates, taking 45 seconds on average. We summarize the error statistics in Table 10.1. Across all trials, when planning with CORRT, x and $\hat{x}$ always remain within the computed tubes $\Omega_c(t)$ and $\Omega_e(t)$; the CCM keeps the tracking error very small, and the OCM shrinks the error by a factor of > 18. Crucially, if a plan is found where $\Omega_e(T)$ satisfies the estimation accuracy threshold, we can ensure our true state estimate satisfies $|\phi_i(T) - \hat{\phi}_i(T)| \leq 0.1$, $i = 1, 2, 3$. We are able to find plans that achieve this threshold for 100/100 trials. We compare with two baselines in this example: B1 (as described before), and B3, which keeps the arm stationary and runs (10.6) for the same duration as the plan computed using CORRT. The purpose of B3 is to show that the actions taken by the CORRT plan help to reduce estimation error. In contrast to CORRT, B1 violates its computed $\Omega_e(t)$ in 44/100 trials and can fail to achieve the required estimation accuracy, only satisfying the 0.1 threshold in 79/100 trials (see Fig. 10.8). One failure example is shown in Fig. 10.8.B: the arm moves too close to the camera (outside of D_r), causing the duck to fall out of frame. This causes a sharp increase in $\hat{h}^{-1}$ error, since ϕ_i cannot be observed; this destabilizes (10.6), leading to a failure to satisfy the 0.1 threshold. Note that B1 does not violate Ω_c; this is because the controller is not a function of the incorrect ϕ_i estimates. Similarly, B3 often fails to satisfy the 0.1-estimation accuracy threshold, only satisfying it in 7/100 trials (see Fig. 10.8.A for a failure example). This shows that passively estimating ϕ_i without moving the arm cannot achieve the needed estimation accuracy; instead, the arm must be moved towards regions with smaller perception error. Overall, this experiment suggests the applicability of our approach on high-dimensional systems, that it can design actions that improve state estimates, and that our approach can plan paths that with high probability, guarantee a desired level of state estimation accuracy.

10.5 Discussion and Conclusion

We present a motion planning algorithm for control-affine systems that enables safe tracking at runtime using an output feedback controller with image observations as input. To achieve this, we learn a perception system and use it in an OCM and CCM-based output feedback control loop. We derive tracking tubes for the closed-loop system and use them within an RRT-based planner to compute plans that theoretically guarantee, with high probability, safe goal-reaching at runtime.

Our results empirically validate this probabilistic safety guarantee, and show that ignoring the effects of state estimation error and the local validity of the perception system/estimator/controller can lead to unsafe behavior.

Our method has some weaknesses which reveal directions for future work. While the large dataset $\mathcal{S}$ used to train $\hat{h}^{-1}$ is easy to gather in simulation, sim-to-real is then needed for $\hat{h}^{-1}$ to transfer to the real world. Thus, in future work, we will combine synthetic, domain-randomized perception data with a small real-world labeled dataset to train generalizable perception modules that have calibrated estimates of the sim-to-real error. Our method also assumes noiseless training data, to ensure L_p is finite; in the future, we wish to relax this. Investigating Lipschitz constant estimation methods robust to input noise *Calliess* (2014) may help achieve this; another possibility is to use a different representation of the model error bound that does not rely on the Lipschitz constant (e.g., using a learned, spatially-varying bound and a constant buffer, as described in the conclusion of Chapter IX). Another drawback is the conservativeness of using worst-case disturbances; to mitigate this, we will integrate stochastic contraction *Kawano and Hosoe* (2021) into our method. Finally, we require θ to be known; in future work, we will aim to jointly estimate θ and x with similar convergence guarantees.

CHAPTER XI

Conclusion and Outlook

In this chapter, we will conclude this thesis by summarizing our key contributions (Sec. 11.1), and by outlining plans for future work (Sec. 11.2).

11.1 Summary

We make core contributions in two primary areas: 1) learning constrained task specifications for safe motion planning, and 2) safe planning and control with learned dynamics and perception modules.

In the first category, we discuss how approximately globally-optimal demonstrations can be used to learn the unknown safety constraints (Chapter III); we use this global optimality assumption to synthetically generate lower-cost trajectories which must violate the unknown constraint, and use this information to synthesize a consistent constraint. In Chapter IV, we relax the global optimality assumption on the demonstrations to one of local optimality; that is, that there exists no local perturbation to the demonstration which enables a decrease in cost without the violation of some constraint. This is formalized using the Karush-Kuhn-Tucker (KKT) conditions from constrained optimization, which are necessary conditions for a candidate solution to an optimization problem to be locally-optimal. This insight enables the learning of constraints with weaker assumptions on the demonstrator. In Chapter V, we address a major drawback of the method presented in Chapter IV – the requirement of a known constraint parameterization. We do this by instead representing the unknown constraint as a nonparametric Gaussian Process, which enables us to avoid prespecifying a set of features that span the set of all possible constraints that we may encounter. In Chapter VI, we move from the time-invariant constraints considered previously to multi-stage, temporally extended tasks. We achieve this by showing how similar notions of optimality can be used to learn linear temporal logic (LTL) formulas from suboptimal demonstrations. Temporal logic provides a means to specify complex temporally-extended tasks with time-varying, history-dependent constraints. Specifically, we use the KKT conditions together with a counterexample-guided falsification approach to learn the atomic propositions (defining low-level state space constraint regions) and logical structure of the unknown LTL formula (determining the high-level flow of the task), respectively. Finally, in Chapter VII, we tackle the

ill-posedness of the constraint learning problem – specifically, the fact that there may be (infinitely) many constraints which are consistent with the optimality conditions of the demonstrations. This is done by obtaining a "belief" over constraints which is driven by the feasible set of consistent constraints in the constraint inference problem; then, we leverage this belief in a chance-constrained optimization framework to plan probabilistically-safe trajectories under this belief.

In the second category, we will discuss how we can plan with complex learned models of high-dimensional systems while guaranteeing safety and goal reachability in execution, when controlling from high-dimensional observations (e.g., images) generated by the system at runtime. The method is first developed for the state-feedback case for systems with the same number of control inputs as states (Chapter VIII). The method works by defining a "trusted domain" around the training data of the dynamics model, and estimating an upper bound on the model error that may be experienced within this domain. Estimating this model error enables us to derive a tracking error bound (that is, how far the system may deviate from a planned trajectory at runtime due to model error), which gives us the means to guarantee safety and robust goal reachability at runtime. This method is then extended to a class of underactuated systems (Chapter IX) by leveraging contraction theory. Finally, the ideas are extended to the output-feedback setting (for planning and control from images) in Chapter X, by estimating upper bounds on the perception error in a domain around the training data.

11.2 Future work

11.2.1 Safe Planning from Pixels with Data-Driven Model Error Bounds

One interesting future direction which builds off of Chapters VIII-X is to relax the assumption of being given a dataset of state transitions to train the dynamics model. This is especially useful in situations where a state representation of the system is difficult to obtain a priori, e.g., a deformable object, like a rope. More generally, it is useful to assume only access to high-dimensional observations when collecting data, as it can in general be difficult to collect data for each *a priori* engineered state in the dynamics.

This is the setting of planning from pixels, or learning latent dynamics models for planning *Hafner et al.* (2019); *Ichter and Pavone* (2019); *Watter et al.* (2015). In this body of work, a dataset of observation-control-next observation tuples is provided to the learner. In general, these observations are images of the system taken at a particular state, and are high-dimensional representations of the underlying system state. Instead of learning dynamics directly in the image space, which would require a huge amount of data and necessitate a complex dynamics model function class, the core idea of latent space planning is to jointly learn a mapping from the observation space to a lower-dimensional latent space (often called an encoder), together with a dynamics model within that latent space. A decoder (which maps from the learned latent space back to observation space) is often also learned (by adding a loss that aims to reconstruct the original image from the low-dimensional latent state), but is

not strictly necessary.

The key improvement that this approach would provide relative to our existing safe perception-based control algorithm in Chapter X is that it eases the data collection process: while the method of Chapter X explicitly requires data of the form $(x, h(x))$, i.e., image observations paired directly with ground-truth states, the latent space approach does not need any ground-truth state information, which makes it substantially easier to implement on real systems, with real images.

Moreover, since the methods that we have developed in Chapters VIII-X have been designed to be integrated within a deep learning pipeline, existing parts of our pipeline should be compatible with additional deep learning components, e.g., an encoder which maps from the high-dimensional observation space to the low-dimensional latent space, where the planning is to occur. Nevertheless, there are some additional challenges that need to be addressed in this setting:

- Estimating the Lipschitz constant of the error in the learned latent space: in Chapters VIII and IX, we could estimate the Lipschitz constant of the error by directly sampling pairs (x, u) and evaluating the discrepancy between the true dynamics and the learned dynamics at those points. In the latent space setting, we cannot simply sample a (z, u) pair and evaluate the error, as each latent state z has no meaning independent of a corresponding high-dimensional observation y. This then necessitates the sampling of (y, u) pairs to estimate the Lipschitz constant, which can complicate the construction of the trusted domain D, which should remain a subset of $\mathcal{X} \times \mathcal{U}$. We have some intial results for accomplishing this through sampling-based approaches, specifically kernel density estimation.

- Another challenge that we will face will arise from the difficulty of jointly learning more components than in prior work. In Chapter IX, we already needed to learn a control contraction metric, a contracting controller, and a dynamics function, where there were multiple losses that depended on each one of these learned components. In addition to these components, we will also need to learn an encoder and possibly a decoder to make the learned latent space well-posed; this will involve more losses. The joint learning of so many components with different competing losses has the potential to make the learning problem much more challenging. To alleviate this, we are currently utilizing a different strategy where we search for a contraction metric and contracting controller by solving a semidefinite program (which can be done if the contraction metric is restricted to be flat, and the contracting controller is restricted to be linear in the differential state). Then, by differentiating through the semidefinite program, we can more directly search for and optimize the recovered contraction metric and controller, which dramatically improves the learning of the other components (e.g., the encoder/decoder).

11.2.2 Learning Dynamics Models from Demonstrations

In Chapters III - VI, we show that global and local notions of optimality alike can be very useful in learning inequality constraints with a small amount of data. It may be possible that a similar framework may be useful in learning dynamics models from demonstrations (or equality constraints more generally). There are some key challenges that we may face here:

- By directly utilizing the "unions of offsets" parameterization developed in Chapter IV, we will be restricted to a very small, simple hypothesis space for the dynamics. We will need to determine a way to relax these restrictions to learn useful dynamics.

- In Chapter V, we are able to obtain the gradients of the unknown inequality constraint by determining if only one unique gradient can make the KKT conditions satisfied. For the examples we considered, most of the recovered gradients were unique up to a scaling. It remains to be seen if the gradients for the dynamics functions will also tend to be unique.

- As detailed in Chapter III, global optimality is currently enforced using sampled lower-cost trajectories within a mixed integer program. This will be incompatible with trying to fit higher-capacity dynamics models. To make our method scalable to modern machine learning-based approaches for dynamics learning, we will ideally determine a way to (approximately) enforce global optimality within the context of a neural network training loop. In particular, it might be possible to utilize the lower-cost trajectories by embedding them in a contrastive loss (*Yan et al.*, 2020) for learning the dynamics.

11.2.3 Safe Planning with Models Learned Online

A core weakness with the approaches presented in Chapters VIII and IX is the requirement of a large, offline-collected dataset of transitions to train the dynamics model. It is often the case that we may need to obtain data online to train the dynamics model. In general, it may be challenging to say much about the safety of the system without additional assumptions or some confidence on an initial dynamics model. However, one setting that might be an interesting point of investigation is the case where locally-linear models of the dynamics are fit online. Linear models can be fit quickly and reliably to new data, unlike neural networks (which can suffer from catastrophic forgetting, etc.), and have been employed in a variety of model-based reinforcement learning applications *Fu et al.* (2016); *Levine and Abbeel* (2014). It would be interesting to investigate if a similar approach based on estimating an error bound in a domain around the training data can allow us to determine when exactly a local model is "accurate enough" for planning and reliable execution, and how this might propagate to tracking error, etc., so that we can guarantee safety for the closed-loop system.

APPENDICES

APPENDIX A

Appendix for Chapter 3: Learning Constraints from Globally-Optimal Demonstrations

A.1 Appendix: Chapter III: Analysis

We review the most important results in this section:

- Theorem A.2 shows that all states that can be guaranteed unsafe must lie within some distance to the boundary of the unsafe set. Corollary A.4 shows that the set of guaranteed unsafe states shrinks to a subset of the boundary of the unsafe set when using a continuous demonstration directly to learn the constraint.
- Corollary A.9 shows that under assumptions on the alignment of the grid and unsafe set for the discrete time case, the guaranteed learned unsafe set is a guaranteed underapproximation of the true unsafe set.
- For continuous trajectories that are then discretized, Theorem A.11 shows us that the guaranteed unsafe set can be made to contain states on the interior of the unsafe set, but at the cost of potentially labeling states within some distance outside of the unsafe set as unsafe as well.
- Theorem III.19 shows that for the parametric case, all states that can be guaranteed unsafe must be implied unsafe by the states within some distance to the boundary of the unsafe set and the parameterization.
- Theorem A.21 shows that for the discrete time case, the guaranteed safe and guaranteed unsafe sets are inner approximations of the true safe and unsafe sets, respectively. For the continuous time case, the recovered sets are inner approximations of a padded version of the true sets.

For convenience, we repeat the definitions and include some illustrations for the sake of visualization. For clarity, the numbers of the definitions, theorems, and corollaries in the appendix match with those in the main body.

A.1.1 Learnability

In this section, we will provide analysis on the learnability of unsafe sets, given the known constraints and cost function. Most of the analysis will be based off unsafe sets defined over the state space, i.e. $\mathcal{A} \subseteq \mathcal{X}$, but we will extend it to the feature space in Corollary A.15. If a state x can be learned to be guaranteed unsafe, then we denote that $x \in {\mathcal{G}^z_{\neg s}}^*$, where ${\mathcal{G}^z_{\neg s}}^*$ is the set of all states that can be learned guaranteed unsafe.

We begin our analysis with some notation.

Definition A.1 (Signed distance)**.** Signed distance from point $p \in \mathbb{R}^m$ to set $\mathcal{S} \subseteq \mathbb{R}^m$, $\mathsf{sd}(p, \mathcal{S}) = -\inf_{y \in \partial \mathcal{S}} \|p - y\|$ if $p \in \mathcal{S}$; $\inf_{y \in \partial \mathcal{S}} \|p - y\|$ if $p \in \mathcal{S}^c$.

The following theorem describes the nature of ${\mathcal{G}^z_{\neg s}}^*$:

Theorem A.2 (Learnability (discrete time))**.** *For trajectories generated by a discrete time dynamical system satisfying $\|x_{t+1} - x_t\| \leq \Delta x$ for all t, the set of learnable guaranteed unsafe states is a subset of the outermost Δx shell of the unsafe set: ${\mathcal{G}^z_{\neg s}}^* \subseteq \{x \in \mathcal{A} \mid -\Delta x \leq \mathit{sd}(x, \mathcal{A}) \leq 0\}$.*

Proof. Consider the case of a length T unsafe trajectory $\xi = \{x_1, \ldots, x_N\}$, $x_1 \in \mathcal{A} \vee \ldots \vee x_T \in \mathcal{A}$. For a state to be learned guaranteed unsafe, $T-1$ states in ξ must be learned safe. This implies that regardless of where that unsafe state is located in the trajectory, it must be reachable from some safe state within one time-step. This is because if multiple states in ξ differ from the original safe trajectory ξ^*, to learn that one state is unsafe with certainty means that the others should be learned safe from some other demonstration. Say that $x_1, \ldots, x_{i-1}, x_{i+1}, \ldots, x_T \in \mathcal{S}$, i.e. they are learned safe. Since $(\|x_{i+1} - x_i\| \leq \Delta x) \wedge (\|x_i - x_{i-1}\| \leq \Delta x)$ and $x_{i-1}, x_{i+1} \in \mathcal{S}$, x_i must be within Δx of the boundary of the unsafe set: $-\min_{y \in \partial \mathcal{A}} \|x_i - y\| \geq \Delta x$, implying $-\Delta x \leq \mathsf{sd}(x_i) \leq 0$.

□

Remark A.3. *For linear dynamics, Δx can be found via*

$$\underset{x \in \mathcal{X}, u \in \mathcal{U}}{\textit{maximize}} \quad \|Ax + Bu - x\| \tag{A.1}$$

In the case of general dynamics, an upper bound on Δx can be found via

$$\Delta x \leq \sup_{x \in \mathcal{X}, u \in \mathcal{U}, t \in \{t_0, t_0+1, \ldots, T\}} \|f(x, u, t) - x\| \tag{A.2}$$

Corollary A.4 (Learnability (continuous time))**.** *For continuous trajectories $\xi(\cdot) : [0, T] \to \mathcal{X}$, the set of learnable guaranteed unsafe states shrinks to the boundary of the unsafe set: ${\mathcal{G}^z_{\neg s}}^* \subseteq \{x \in \mathcal{A} \mid \mathit{sd}(x, \mathcal{A}) = 0\}$.*

Proof. The output trajectory of a continuous time system can be seen as the output of a discrete time system in the limit as the time-step is taken to 0. In this case, as long as the dynamics are locally Lipschitz continuous, $\Delta x \doteq \lim_{\Delta t \to 0} \|x(t+\Delta t) - x(t)\| \to 0$ (*Khalil* (2002)), and via Theorem A.2, the corollary is proved. □

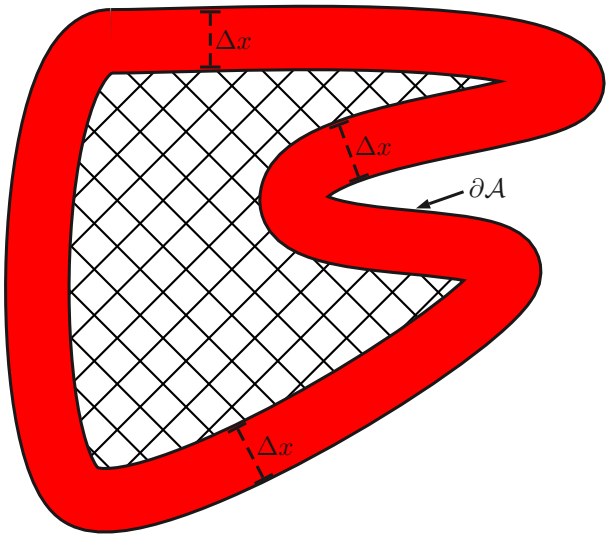


Figure A.1: Illustration of the outermost Δx shell (shown in red) of the unsafe set $\mathcal{A}$. The hatched area cannot be learned guaranteed safe.

Depending on the cost function, ${\mathcal{G}^z_{\neg_s}}^*$ can become arbitrarily small: some cost functions are not very informative for recovering a constraint. For example, the path length cost function used in many of the experiments (which was chosen due to its common use in the motion planning community), prevents any lower-cost sub-trajectories from being sampled from straight sub-trajectories. The overall control authority that we have on the system also impacts learnability: the more controllable the system, the more of the Δx shell is reachable. In particular, a necessary condition for any unsafe states to be learnable from a demonstration of length $T+1$ starting from x_0 and ending at x_T is for there to be more than one trajectory which steers from x_0 to x_T in $T+1$ steps while satisfying the dynamics and control constraints.

A.1.2 Conservativeness

For the analysis in this section, we will assume that the unsafe set has a Lipschitz boundary; informally, this means that $\partial\mathcal{A}$ can be locally described by the graph of a Lipschitz continuous function. A formal definition can be found in *Dacorogna* (2015). We define some notation:

Definition A.5 (Normal vectors)**.** Denote the outward-pointing normal vector at a point $p \in \partial\mathcal{A}$ as $\hat{n}(p)$. Furthermore, at non-differentiable points on $\partial\mathcal{A}$, $\hat{n}(p)$ is replaced by the set of normal vectors for the sub-gradient of the Lipschitz function describing $\partial\mathcal{A}$ at that point (*Allaire et al.* (2016)).

Definition A.6 (γ-offset padding)**.** Define the γ-offset padding $\partial\mathcal{A}_\gamma$ as: $\partial\mathcal{A}_\gamma = \{x \in \mathcal{X} \mid x = y + d\hat{n}(y), d \in [0,\gamma], y \in \partial\mathcal{A}\}$.

Definition A.7 (γ-padded set)**.** We define the γ-padded set of the unsafe set $\mathcal{A}$, $\mathcal{A}(\gamma)$, as the union of the γ-offset padding and $\mathcal{A}$: $\mathcal{A}(\gamma) \doteq \partial\mathcal{A}_\gamma \cup \mathcal{A}$.

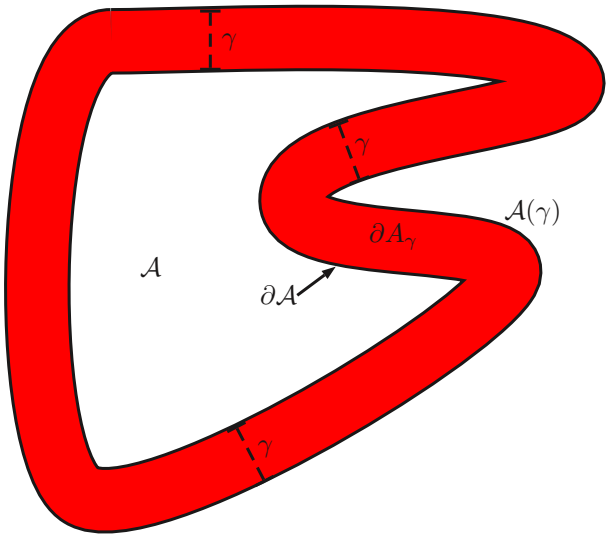


Figure A.2: Illustration of the γ-padded set $\mathcal{A}(\gamma)$, which is the union of the red and white regions. The γ-offset padding is displayed in red. The original set $\mathcal{A}$ is shown in white.

Definition A.8 (Maximum grid size)**.** Let $R(z_i)$ be the radius of the smallest ball which contains grid cell z_i: $R(z_i) \doteq \min_r \min_{x_i} r$, subject to $z_i \subseteq B_r(x_i)$, for some optimal center x_i.

Furthermore, let R^* be the radius of the smallest ball which contains each grid cell $z_i, i = 1, \ldots, G$: $R^* = \max(R(z_1), \ldots, R(z_G))$.

We introduce the following assumption, which is illustrated in Figure A.3 for clarity:

Assumption 1: The unsafe set $\mathcal{A}$ is aligned with the grid (i.e. there does not exist a grid cell z containing both safe and unsafe states in its interior).

Theorem A.9 (Discrete time conservative recovery of unsafe set)**.** *For a discrete-time system, if Assumption 1 holds,* $\mathcal{G}^z_{\neg s} \subseteq \mathcal{A}$*. If Assumption 1 does not hold, then* $\mathcal{G}^z_{\neg s} \subseteq \mathcal{A}(R^*)$*.*

Proof. In discrete-time, we know that each trajectory sampled using Algorithm III.1 starting from an optimal demonstration contains at least one truly unsafe state, i.e. for all $\xi_j, j \in \{1, \ldots, N_{\neg s}\}$, there exists $x \in \xi_j, x \in \mathcal{A}$. Then, if Assumption 1 holds, enforcing $z_i \ni x$ to be unsafe can never also enforce that some safe state $y \in \mathcal{S}$ is unsafe. If Assumption 1 does not hold, suppose that there exists $x \in \partial\mathcal{A}$ which is learned guaranteed unsafe, and that $x \in z_i$, where $(z_i \cap \mathcal{A}) \subseteq \partial\mathcal{A}$ (i.e. the grid cell only touches the boundary of the unsafe set). Then, $\mathcal{G}^z_{\neg s} \subseteq \mathcal{A}(R(z_i)) \subseteq \mathcal{A}(R^*)$. □

Note that if we deal with continuous trajectories directly, the guaranteed learnable set shrinks to a subset of the boundary of the unsafe set, $\partial\mathcal{A}$. However, if we discretize these trajectories, we can learn unsafe states lying in the interior, at the cost of conservativeness guarantees holding only for a padded unsafe set.

The following results hold for continuous time trajectories. We begin the discussion with an intermediate result we will need for Theorem A.11:

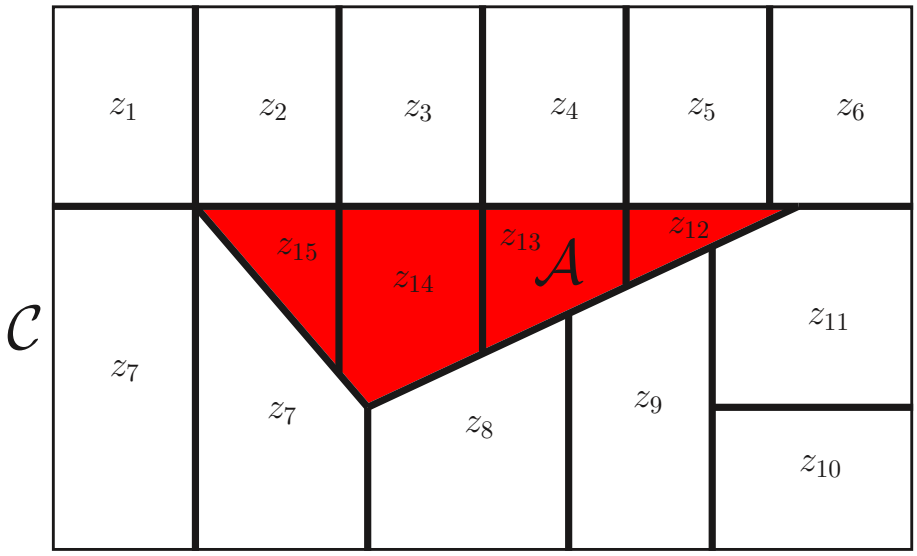


Figure A.3: Illustration of Assumption 1 - all grid cells are either fully contained by $\mathcal{A}$ or $\mathcal{A}^c$.

Lemma A.10 (Maximum distance)**.** *Consider a continuous time trajectory $\xi : [0, T] \to \mathcal{X}$. Suppose it is known that in some time interval $[a, b], a \leq b, a, b \in [0, T]$, ξ is unsafe; denote this sub-segment as $\xi([a, b])$. Consider any $t \in [a, b]$. Then, the signed distance from $\xi(t)$ to the unsafe set, $\mathsf{sd}(\xi(t), \mathcal{A})$, is bounded by $D_\xi([a, b]) \doteq \sup_{t_1 \in [a,b], t_2 \in [t_1,b]} \|\xi(t_1) - \xi(t_2)\|_2$.*

Proof. Since there exists $\tilde{t} \in [a, b]$ such that $\xi(\tilde{t}) \in \mathcal{A}$,

$$\sup_{t \in [a,b]} \mathsf{sd}(\xi(t), \mathcal{A}) = \sup_{t \in [a,b]} \mathsf{sd}(\xi(t), \xi(\tilde{t})) \leq \sup_{t_1 \in [a,b], t_2 \in [t_1,b]} \|\xi(t_1) - \xi(t_2)\|_2$$

. □

We introduce another assumption, which is illustrated in Figure A.4 for clarity:
Assumption 2: The time discretization of the unsafe trajectory $\xi : [0, T] \to \mathcal{X}$, $\{t_1, \ldots, t_N\}, t_i \in [0, T]$, for all i, is chosen such that there exists at least one discretization point in the interior of each cell that the continuous trajectory passes through (i.e. if $\exists t \in [0, T]$ such that $\xi(t) \in z$, then $\exists t_i \in \{t_1, \ldots, t_N\}$ such that $\xi(t_i) \in z$).

We also introduce a convention for tie-breaking in Problems III.2, III.4, and III.5. Suppose there exists an unsafe trajectory ξ for which a safe cell z is incorrectly learned guaranteed unsafe due to time discretization. If a demonstration is added to the optimization problem which marks cell z as safe, to avoid infeasibility, we remove the unsafe trajectory ξ from the optimization problem.

Theorem A.11 (Continuous-to-discrete time conservativeness)**.** *The following results hold for continuous time systems:*

1. *Suppose that both Assumptions 1 and 2 hold. Then, the learned guaranteed unsafe set $\mathcal{G}^z_{\neg_s}$, defined in Section 3.3.4.1, is contained within the true unsafe set $\mathcal{A}$.*

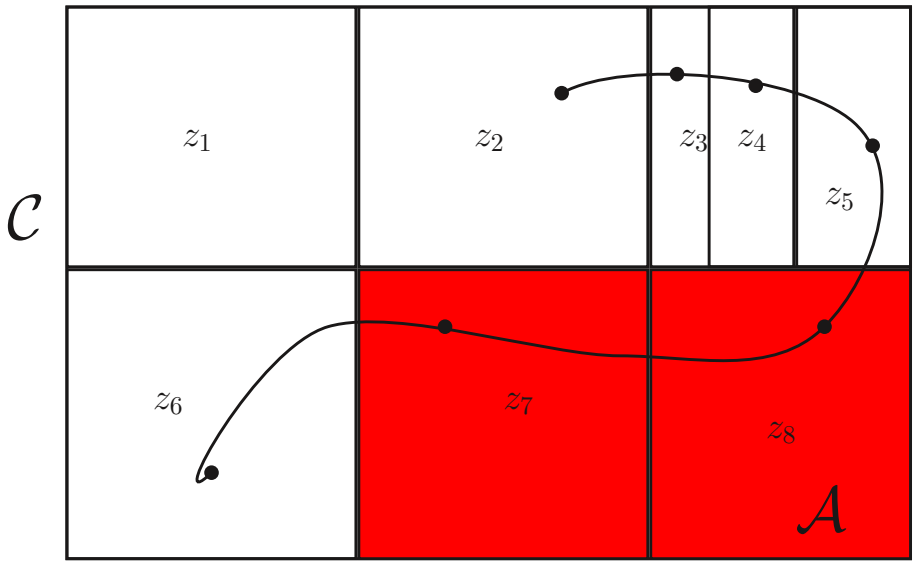


Figure A.4: Illustration of Assumption 2: each cell z that the trajectory passes through must have a time discretization point (shown as a dot).

2. *Suppose that only Assumption 2 holds. Then, the learned guaranteed unsafe set* $\mathcal{G}^z_{\neg_s}$ *is contained within the* R^**-padded unsafe set,* $\mathcal{A}(R^*)$*.*

3. *Suppose that neither Assumption 1 nor Assumption 2 holds. Furthermore, suppose that Problems III.2, III.4, and III.5 are using* M *sub-trajectories sampled with Algorithm III.1 as unsafe trajectories, and that each sub-trajectory is defined over the time interval* $[a_i, b_i], i = 1, \ldots, M$*. Denote* $D_\xi([a,b]) \doteq \sup_{t_1 \in [a,b], t_2 \in [t_1,b]} \|\xi(t_1) - \xi(t_2)\|_2$*, for some trajectory* ξ*. Denote* $D^* \doteq \max_{i \in \{1,\ldots,M\}} D^*_{\xi_i}([a_i, b_i])$*. Then, the learned guaranteed unsafe set* $\mathcal{G}^z_{\neg_s}$ *is contained within the* $D^* + R^*$*-padded unsafe set,* $\mathcal{A}(D^* + R^*)$*.*

Proof. Let's prove the case where both Assumptions 1 and 2 hold. By Assumption 1, all cells z which contain unsafe states $x \in \mathcal{A}$ must be fully contained in the unsafe set: $z \in \mathcal{A}$. Now, suppose there exists a trajectory $\xi : [0, T] \to \mathcal{X}$ which is unsafe (i.e it satisfies the known constraints and has lower cost than a demonstration). Then, there exists at least one $t \in [0, T]$ such that $\xi(t) \in \mathcal{A}$. By Assumption 2, there exists a discretization point $t_i \in [0, T]$ such that $\xi(t_i)$ lies within some cell z, and $z \in \mathcal{A}$ by Assumption 1. Hence, we will only learn grid cells within $\mathcal{A}$ to be unsafe: $\mathcal{G}^z_{\neg_s} \subseteq \mathcal{A}$.

If only Assumption 2 holds, $\mathcal{G}^z_{\neg_s} \subseteq \mathcal{A}(R^*)$ due to the gridding: suppose there exists a cell z_k containing both safe and unsafe states which is learned guaranteed unsafe. Then, by padding the unsafe set to contain any grid cell $z_1, \ldots, z_G$, z_k is fully contained, and hence the algorithm returns a conservative estimate of the $D^* + R_k \leq D^* + R^*$-padded unsafe set.

Let's prove the case where neither assumption holds. Suppose in this case, there exists a cell $z \not\subseteq \mathcal{A}$ which is truly safe, but for which we have no demonstration that says cell z is safe. Now, suppose there exists an unsafe trajectory $\xi_j([a_j, b_j])$ passing through z which violates Assumption 2. Suppose that $\xi_j(t_i) \in z$, and $\{t_1, \ldots, t_N\}$ is chosen such that for all $j \in \{1, \ldots, N\} \setminus \{i\}$, $\xi_j(t_i)$ belongs to a known safe cell.

Then, we may incorrectly learn that $z \in \mathcal{G}^z_{\neg s}$, as we force at least one point in the sampled trajectory to be unsafe. Via Lemma A.10, we know that $\xi_j(t_i)$ is at most $D_{\xi_j}([a_j, b_j])$ signed distance away from $\mathcal{A}$. Hence, for this trajectory, any learned guaranteed unsafe state must be contained in the $D_{\xi_j}([a_j, b_j])$-padded unsafe set. For this to hold for all unsafe trajectories sampled with Algorithm III.1, we must pad the unsafe set by D^*. Lastly, to account for the gridding, suppose that $\xi_{j^*}(t_i)$ is contained in cell z_k, which is then marked unsafe. Then, by padding the set to contain z_k, the algorithm returns a conservative estimate of the $D^* + R_k \leq D^* + R^*$-padded unsafe set.

□

Remark A.12. *In practice, we observe that the bound in Theorem A.11 when using only Assumption 1 is quite conservative, and as more demonstrations are added to the optimization, using the tie-breaking rule described previously removes the overapproximations described by Theorem A.11. Furthermore, though the experiments are implemented using only Assumption 1, ensuring Assumption 2 also holds is straightforward as long as the grid cells are large enough such that finding a sufficiently fine time-discretization is efficient.*

Remark A.13. *Note that for the cases where Assumption 1 does not hold, safe states can be incorrectly forced to be unsafe; thus, the constraint recovery program can become infeasible. In these situations, we use the tie-breaking rule described before the statement of Theorem A.11 to keep the program feasible.*

Remark A.14. *If some state x on a demonstration lies directly on the boundary between two grid cells z_i and z_j, neither z_i nor z_j is enforced to be safe unless either of z_i or z_j is learned to be unsafe; then the other grid cell can be labeled safe. Furthermore, the demonstrations that appear to lie on the boundary of the unsafe set actually lie in the interior of the safe set and very close to the boundary due to the solvers' numerical tolerance; hence we do not actually have any demonstrations lying exactly on the boundary of any grid cells in the experiments.*

Corollary A.15 (Continuous-to-discrete feature space conservativeness)**.** *Let the feature mapping $\phi(x)$ from the state space to the constraint space be Lipschitz continuous with Lipschitz constant L. Then, the following results hold:*

1. *Suppose both Assumptions 1 and 2 (used in Theorem III.16) hold. Then, our method ensures $\mathcal{G}^z_{\neg s} \subseteq \mathcal{A}$.*
2. *Suppose only Assumption 2 holds. Then, our method recovers a guaranteed subset of the LR^*-padded unsafe set, $\mathcal{A}(LR^*)$, in the feature space.*
3. *Suppose neither Assumption 1 nor Assumption 2 holds. Then, our method recovers a guaranteed subset of the $L(D^* + R^*)$-padded unsafe set, $\mathcal{A}(L(D^* + R^*))$, where D^* is as defined in Theorem III.16.*

Proof. Under Assumptions 1 and 2, the result follows directly from the logic in Theorem A.11. Now, consider the case where only Assumption 2 holds. From the definition

of Lipschitz continuity, $\|\phi(x)-\phi(y)\| \leq L\|x-y\|$. From Theorem A.11, the unsafe set estimate is a subset of the R^*-padded estimate in the continuous space case. Using Lipschitz continuity, the value of the feature can at most change by LR^* from the boundary of the true constraint set to the boundary of the padded set; hence, the statement holds. Analogous reasoning holds for the case where neither assumption holds. □

A.1.3 Learnability: Parametric

In this section, we develop results for learnability of the unsafe set in the parametric case. We begin with the following notation:

Definition A.16 (Implied unsafe set)**.** For some set $\mathcal{B} \subseteq \Theta$, denote

$$I(\mathcal{B}) \doteq \bigcap_{\theta \in \mathcal{B}} \{x \mid g(x, \theta) \leq 0\} \tag{A.3}$$

as the set of states that are implied unsafe by restricting the parameter set to $\mathcal{B}$. In words, $I(\mathcal{B})$ is the set of states for which all $\theta \in \mathcal{B}$ mark as unsafe.

Lemma A.17. *Suppose $\mathcal{B} \subseteq \hat{\mathcal{B}}$, for some other set $\hat{\mathcal{B}}$. Then, $I(\hat{\mathcal{B}}) \subseteq I(\mathcal{B})$.*

Proof. By definition,

$$\begin{aligned} I(\hat{\mathcal{B}}) &= \bigcap_{\theta \in \hat{\mathcal{B}}} \{x \mid g(x, \theta) \leq 0\} \\ &= \bigcap_{\theta \in \left(\mathcal{B} \cup (\hat{\mathcal{B}} \setminus \mathcal{B})\right)} \{x \mid g(x, \theta) \leq 0\} \\ &\subseteq \bigcap_{\theta \in \mathcal{B}} \{x \mid g(x, \theta) \leq 0\} \\ &= I(\mathcal{B}). \end{aligned}$$

□

Lemma A.18. *Denote the Δx-shell of $\mathcal{A}$ as $\mathcal{A}_{\Delta x}$, where Δx is as defined in Theorem A.2. Then, each unsafe trajectory ξ_j with start and goal states in the safe set contains at least one state in $\mathcal{A}_{\Delta x}$: $\forall j \in \{1, \ldots, N_{\neg s}\}, \exists x \in \xi_j, x \in \mathcal{A}_{\Delta x}$.*

Proof. For each unsafe trajectory ξ_j with start and goal states in the safe set, there exists $x \in \xi_j, x \in \mathcal{A}$. Further, if there exists $x \in \xi_j \in (\mathcal{A} \setminus \mathcal{A}_{\Delta x})$, then there also exists $x \in \xi_j \in \mathcal{A}_{\Delta x}$. For contradiction, suppose there exists a time $\hat{t} \in \{1, \ldots, T_j\}$ for which $\xi_j(\hat{t}) \in (\mathcal{A} \setminus \mathcal{A}_{\Delta x})$ and $\nexists t \in \{1, \ldots, T_j\}$ for which $\xi_j(t) \in \mathcal{A}_{\Delta x}$. But this implies $\exists t < \hat{t}, \|\xi(t) - \xi(t+1)\| > \Delta x$ or $\exists t > \hat{t}, \|\xi(t) - \xi(t-1)\| > \Delta x$, i.e. to skip deeper than Δx into the unsafe set without first entering the Δx shell, the state must have changed by more than Δx in a single time-step. Contradiction. An analogous argument holds for the continuous-time case. □

Denote as $\mathcal{G}^*_{\neg s}$ the learnable set of unsafe states. Further denote as $\mathcal{F}_{\Delta x}$ the set of parameters that sets all states in $\mathcal{A}_{\Delta x}$ as unsafe and all states on safe trajectories as safe. Last, denote as $I(\mathcal{F}_{\Delta x})$ the set of states that are implied as unsafe by restricting the parameter set to $\mathcal{F}_{\Delta x}$. The following result states that in discrete time, $\mathcal{G}^*_{\neg s}$ is contained by $I(\mathcal{F}_{\Delta x})$. Furthermore, in continuous time, the same holds, except the Δx shell is replaced by the boundary of the unsafe set, $\partial\mathcal{A}$.

Theorem A.19 (Discrete time learnability for parametric constraints)**.** *For trajectories generated by discrete time systems, $\mathcal{G}_{\neg s} \subseteq \mathcal{G}^*_{\neg s} \subseteq I(\mathcal{F}_{\Delta x})$, where*

$$\begin{aligned}\mathcal{F}_{\Delta x} = \{\theta \mid \forall i \in \{1,\dots,N_s\}, \forall x \in \xi_i^*, g(x,\theta) > 0,\\ \forall x \in \mathcal{A}_{\Delta x}, g(x,\theta) \leq 0\}\end{aligned}$$

Proof. Recall that $\mathcal{G}_{\neg s} \doteq \bigcap_{\theta\in\mathcal{F}}\{x \mid g(x,\theta) \leq 0\}$, where $\mathcal{F}$ is the feasible set of Problem III.3:

$$\begin{aligned}\mathcal{F} = \{\theta \mid \forall i \in \{1,\dots,N_s\}, \forall x \in \xi_i^*, g(x,\theta) > 0,\\ \forall j \in \{1,\dots,N_{\neg s}\}, \exists x \in \xi_j, g(x,\theta) \leq 0\}\end{aligned}$$

We can then show that $\mathcal{F}_{\Delta x} \subseteq \mathcal{F}$, since enforcing that $g(x,\theta) \leq 0$ for all $x \in \mathcal{A}_{\Delta x}$ implies that there exists $x \in \xi_j$, for all $j \in \{1,\dots,N_{\neg s}\}$ such that $g(x,\theta) \leq 0$, via Lemma A.18. Then, via Lemma A.17, $\mathcal{G}_{\neg s} = I(\mathcal{F}) \subseteq I(\mathcal{F}_{\Delta x})$. As this holds for any arbitrary set of trajectories, $\mathcal{G}^*_{\neg s} \subseteq I(\mathcal{F}_{\Delta x})$ as well, and $\mathcal{G}_{\neg s} \subseteq \mathcal{G}^*_{\neg s}$. □

Corollary A.20 (Continuous-time learnability for parametric constraints)**.** *For trajectories generated by continuous time systems, $\mathcal{G}_{\neg s} \subseteq \mathcal{G}^*_{\neg s} \subseteq I(\mathcal{F}_{\partial\mathcal{A}})$, where*

$$\begin{aligned}\mathcal{F}_{\partial\mathcal{A}} = \{\theta \mid \forall x \in \xi_i^*, \forall i \in \{1,\dots,N_s\}, g(x,\theta) > 0,\\ \forall x \in \partial\mathcal{A}, g(x,\theta) \leq 0\}\end{aligned}$$

Proof. Since going from discrete time to continuous time implies $\Delta x \to 0$, $\mathcal{A}_{\Delta x} \to \partial\mathcal{A}$. Then, the logic from the proof of Theorem A.19 can be similarly applied to show the result. □

A.1.4 Conservativeness: Parametric

We write conditions for conservative recovery of the unsafe set and safe set when solving Problems III.3 and III.7 for discrete time and continuous time systems.

Theorem A.21. *For a discrete-time system, if M in Problem III.7 is chosen to be greater than $\max(M_1, M_2)$, where $M_1 = \max_{x_i\in\xi_s}\max_\theta \max_j (H(\theta)x_i - h(\theta))_j$ and $M_2 = \max_{x_i\in\xi_{\neg s}}\max_\theta \max_j (H(\theta)x_i - h(\theta))_j$, $\mathcal{G}_{\neg s} \subseteq \mathcal{A}$ and $\mathcal{G}_s \subseteq \mathcal{S}$.*

Proof. We first prove that $\mathcal{G}_{\neg s} \subseteq \mathcal{A}$. Consider first the case where $M = \infty$ and therefore Problem III.7 exactly enforces that at least one state in each unsafe trajectory is unsafe and all states on demonstrations are safe.

Suppose for contradiction that there exists some $x \in \mathcal{G}_{\neg s}, x \notin \mathcal{A}$. By definition of $\mathcal{G}_{\neg s}$, $g(x,\theta) \leq 0$, for all $\theta \in \mathcal{F}$, where $\mathcal{F}$ is the feasible set of parameters θ in Problem III.3. However, as $x \notin \mathcal{A}$, but for all $\theta \in \mathcal{F}, g(x,\theta) \leq 0$ we know that $\theta_{\mathcal{A}} \notin \mathcal{F}$, where $\theta_{\mathcal{A}}$ is the parameter associated with the true unsafe set $\mathcal{A}$. However, $\mathcal{F}$ will always contain $\theta_{\mathcal{A}}$, since:

- $\theta_{\mathcal{A}}$ satisfies $g(x,\theta_{\mathcal{A}}) > 0$ for all x in safe demonstrations, since all demonstrations are safe with respect to the true $\theta_{\mathcal{A}}$.
- For each trajectory $\xi_{\neg s}$ sampled using Algorithm III.1, there exists $x \in \xi_{\neg s}$ such that $g(x,\theta_{\mathcal{A}}) \leq 0$.

We come to a contradiction, and hence for $M = \infty$, $\mathcal{G}_{\neg s} \subseteq \mathcal{A}$.

Now, we consider the conditions on M such that choosing $M \geq$ const or $M = \infty$ causes no changes in the solution of Problem III.7. M must be chosen such that 1) $H(\theta)x_i - h(\theta) > -M\mathbf{1} \Leftrightarrow H(\theta)x_i - h(\theta) > -\infty\mathbf{1}$, for all safe states $x_i \in \xi_s$, and 2) $H(\theta)x_i - h(\theta) \leq M\mathbf{1} \Leftrightarrow H(\theta)x_i - h(\theta) \leq M\mathbf{1}$ for all states x_i on unsafe trajectories $\xi_{\neg s}$. Condition 1 is met if $-M < \min_{x_i \in \xi_s} \min_\theta \min_j (H(\theta)x_i - h(\theta))_j$, where v_j denotes the j-th element of vector v; denote as M_1 an M which satisfies this inequality. Condition 2 is met if $M \geq \max_{x_i \in \xi_{\neg s}} \max_\theta \max_j (H(\theta)x_i - h(\theta))_j$; denote as M_2 an M which satisfies this inequality. Then, M should be chosen to satisfy $M > \max(M_1, M_2)$.

The proof that $\mathcal{G}_s \subseteq \mathcal{S}$ is analogous. If there exists $x \in \mathcal{G}_s, x \notin \mathcal{S}$, $g(x,\theta) > 0$, for all $\theta \in \mathcal{F}$, then $\theta_{\mathcal{A}} \notin \mathcal{F}$. We follow the same reasoning from before to show that $\theta_{\mathcal{A}} \in \mathcal{F}$ for $M = \infty$. Now, provided the condition on M holds, we reach a contradiction. □

Corollary A.22. *For a continuous-time system, where demonstrations are time-discretized as discussed in Section A.1.2, if M is chosen as in Theorem A.21, $\mathcal{G}_s \subseteq \mathcal{S}$ and $\mathcal{G}_{\neg s} \subseteq \mathcal{A}(D^*)$, where D^* is as defined in Theorem A.11.*

Proof. The reasoning for $\mathcal{G}_s \subseteq \mathcal{S}$ follows from the proof of Theorem A.21.

For proving $\mathcal{G}_{\neg s} \subseteq \mathcal{A}(D^*)$, we follow the proof of Theorem A.11 until it is shown that any learned guaranteed unsafe state must be contained in the $\mathcal{A}(D^*)$. However, for the parametric case, there is no notion of a grid and hence the further padding by R^* is unnecessary. □

Finally, we restate and prove our results on constraint conservativeness when working with unknown constraint parameterizations.

Theorem A.23 (Conservativeness: Over-parameterization (Theorem III.23 in the main body))**.** *Suppose the true parameterization and over-parameterization are defined as in* (3.14) *and* (3.16)*. Then, $\mathcal{G}_{\neg s} \subseteq \mathcal{A}$ and $\mathcal{G}_s \subseteq \mathcal{S}$.*

Proof. Note that (3.14) is equivalent to $\left(\bigvee_{i=1}^{\bar{N}} \left(g_s(x,\theta_i) \leq 0 \right) \right)$, where $\theta_{N^*+1}, \ldots, \theta_{\bar{N}}$ are constrained to satisfy $\{x \mid g_s(x,\theta_i) \leq 0\} = \emptyset, i = N^*+1, \ldots, \bar{N}$. Thus, the true θ is equivalent to adding additional constraints on a loosened parameterization (the

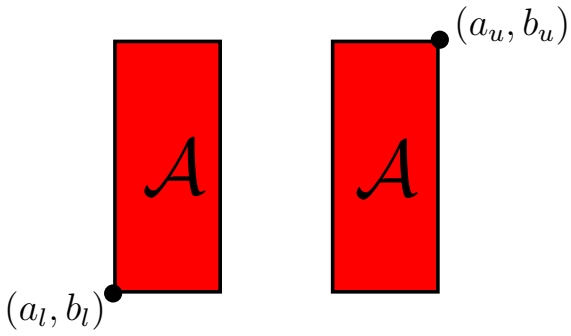


Figure A.5: Counterexample used in the proof of the first statement in Theorem A.24.

over-parameterization). Let $\hat{\mathcal{F}}$ be the feasible set of Problem III.3 with θ loosened as above, i.e. $\mathcal{F} = \hat{\mathcal{F}} \cap \{\theta \mid \{x \mid g_s(x, \theta_i) \leq 0\} = \emptyset, i = N^* + 1, \ldots, \bar{N}\}$. Via Lemma A.17, $\mathcal{F} \subseteq \hat{\mathcal{F}}$; thus, $I_{\neg s}(\hat{\mathcal{F}}) \subseteq I_{\neg s}(\mathcal{F}) \subseteq \mathcal{A}$, where the last set containment follows from Theorem III.21. Vice versa, $I_s(\hat{\mathcal{F}}) \subseteq I_s(\mathcal{F}) \subseteq \mathcal{S}$, where again the last set containment follows from Theorem III.21. □

Theorem A.24 (Conservativeness: Under-parameterization (Theorem III.24 in the main body))**.** *Suppose the true parameterization and under-parameterization are defined as in* (3.14) *and* (3.15)*. Furthermore, assume that we incrementally grow the parameterization as described in Section 3.3.5.3. Then, the following are true:*

1. *$\mathcal{G}_{\neg s}$ and $\mathcal{G}_s$ are not guaranteed to be contained in $\mathcal{A}$ (unsafe set) and $\mathcal{S}$ (safe set), respectively.*

2. *Each recovered simple unsafe set $\mathcal{A}(\theta_i)$, $i = 1, \ldots, \underline{N}$, for any $\theta_1, \ldots, \theta_{\underline{N}} \in \mathcal{F}$, touches the true unsafe set (there are no spurious simple unsafe sets): for $i = 1, \ldots, \underline{N}$, for $\theta_1, \ldots, \theta_{\underline{N}} \in \mathcal{F}$, $\mathcal{A}(\theta_i) \cap \mathcal{A} \neq \emptyset$ ($\underline{N}$ is as defined in Section 3.3.5.3).*

Proof. 1. We first formally prove the statement with a counterexample and then follow up with logic related to the proof of Theorem A.23.

Consider the example in Fig. A.5, where the parameterization is chosen as a single axis-aligned box $[I_{2\times 2}, -I_{2\times 2}]^\top x \leq \theta$ but $\mathcal{A}$ is only representable with at least two boxes. Suppose demonstrations are provided which imply that (a_l, b_l) and (a_u, b_u) are unsafe; then $\text{AABB}(\{(a_l, b_l), (a_u, b_u)\}) \not\subseteq \mathcal{A}$ is implied unsafe.

Note that (3.15) is equivalent to $\left(\bigvee_{i=1}^{N^*} \left(g_s(x, \theta_i) \leq 0\right)\right)$, where $\theta_{\underline{N}+1}, \ldots, \theta_{N^*}$ are constrained to satisfy $\{x \mid g_s(x, \theta_i) \leq 0\} = \emptyset, i = \underline{N} + 1, \ldots, N^*$. Thus, restricting the parameterization is equivalent to adding additional constraints on the true θ. Let $\hat{\mathcal{F}}$ be the feasible set of Problem III.3 with θ restricted as above, i.e. $\hat{\mathcal{F}} = \mathcal{F} \cap \{\theta \mid \{x \mid g_s(x, \theta_i) \leq 0\} = \emptyset, i = \underline{N}+1, \ldots, N^*\}$. Via Lemma A.17, $\hat{\mathcal{F}} \subseteq \mathcal{F}$; thus, $I_{\neg s}(\mathcal{F}) \subseteq I_{\neg s}(\hat{\mathcal{F}})$. Since $I_{\neg s}(\mathcal{F})$ can equal $\mathcal{A}$, potentially $\mathcal{G}_{\neg s} = I_{\neg s}(\hat{\mathcal{F}}) \cap \mathcal{S} \neq \emptyset$. Vice versa, $I_s(\mathcal{F}) \subseteq I_s(\hat{\mathcal{F}})$, and since $I_s(\mathcal{F})$ can equal $\mathcal{S}$, potentially $\mathcal{G}_s = I_s(\hat{\mathcal{F}}) \cap \mathcal{S} \neq \emptyset$.

2. Assume, by contradiction, that Problem III.3 outputs a simple unsafe set $\mathcal{A}(\theta_i), i \in \{1, \ldots, \underline{N}\}$, which does not touch the true unsafe set: $\exists i \in \{1, \ldots, \underline{N}\}, \mathcal{A}(\theta_i) \cap$

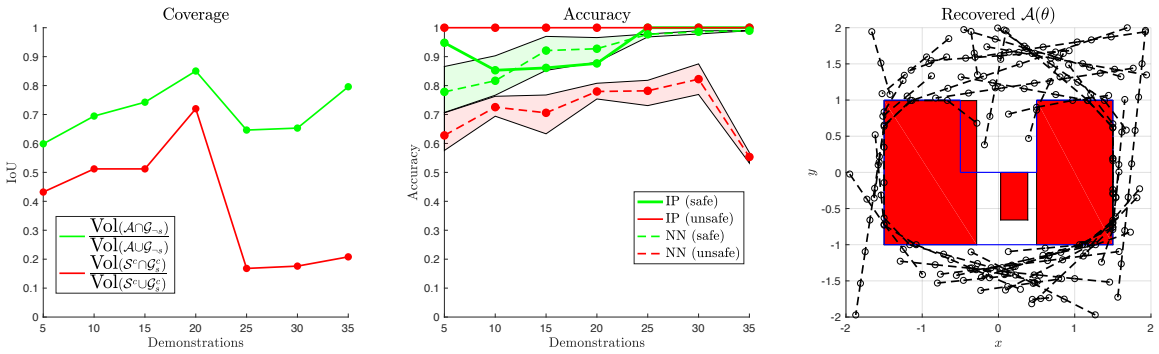


Figure A.6: U-shape performance with random demonstrations. **Left**: Coverage of $\mathcal{A}$ and $\mathcal{S}$. **Center**: Classification accuracy. **Right**: A recovered feasible $\mathcal{A}(\theta)$, overlaid with demonstrations, and the true unsafe set $\mathcal{A}$ is outlined in blue.

$\mathcal{A}(\theta^*) = \emptyset$. Then, $\theta_j, j \in \{1, \ldots, \underline{N}\} \setminus \{i\}$ would be a feasible point for Problem III.3 with a parametrization that contains only $\underline{N} - 1$ simple sets. However, we know Problem III.3 with $\underline{N} - 1$ simple sets is infeasible. Contradiction.

□

A.2 Appendix: Chapter III: Extra numerical examples

A.2.1 U-shape (random demonstrations)

In this example, we show what the performance of our method looks like with random demonstrations on the U-shape example. On the left of Fig. A.6, we show that our coverage grows more slowly than for the case where demonstrations are chosen for their informativeness; furthermore, coverage for the safe set is higher and coverage for the unsafe set is lower in the random demonstration case. This is because by using random demonstrations, we cover a good deal of $\mathcal{S}$, so $\mathcal{G}_s$ becomes larger; on the other hand, many of these safe demonstrations may not come in contact with the constraint, so there are relatively few unsafe trajectories that can be sampled, so $\mathcal{G}_{\neg s}$ is not as large. In the center of Fig. A.6, we show that the accuracy of our method doesn't change much, though the relative performance of the NN gets worse for classifying safe states; this is because the accuracy for the NN is now being evaluated on a larger region since $\mathcal{G}_s$ is larger due to more demonstrations. As in previous examples, the NN error bars are generated by training the NN ten times with initializations using different random seeds. On the right of Fig. A.6, we display a feasible $\mathcal{A}(\theta)$ recovered by solving a multi-box variant of Problem III.7. With more demonstrations, the gap between $\mathcal{A}(\theta)$ and the true unsafe set $\mathcal{A}$ will continue to shrink.

The main takeaways from this experiment are: 1) when demonstrations are not informative (in the sense that they do not interact with the constraint), it can take many demonstrations to learn the unsafe set (this holds for any constraint recovery method), and 2) our accuracy remains just as high as for the case with specifically chosen demonstrations and is not much affected by the coverage.

A.3 Appendix: Chapter III: Experimental details

For all neural network baseline results in every experiment, the network is trained with weights initialized using ten different random seeds, and the resulting performance range (displayed as a shaded region) and average performance over the ten random seeds are plotted in the figures.

A.3.1 Unknown parameterizations

We emphasize that for all examples with unknown parameterization, by following the incremental procedure detailed in Section 3.3.5.3, we are finding the minimum number of boxes required to represent the data; in other words, we are always operating with the minimal feasible parameterization.

U-shape and infinite boxes:

- For both experiments, the system dynamics are $x_{t+1} \doteq [\chi_{t+1}, y_{t+1}]^\top = [\chi_t, y_t]^\top + [u_t^\chi, u_t^y]^\top$. The U-shape experiment uses control constraints $\|[u_t^\chi, u_t^y]\|_2 \leq 0.5$, while the infinite-box experiment uses control constraints $\|[u_t^\chi, u_t^y]\|_2 \leq 1$.
- For both experiments, the cost function is $c(\xi_x, \xi_u) = \sum_{i=1}^{T-1} \|x_{t+1} - x_t\|_2^2$.
- Since the cost function has optimal substructure, 100000 unsafe trajectories for each sub-trajectory are sampled. The dataset is downsampled to 50 unsafe trajectories for each sub-trajectory, which are to be fed into the multi-box variant of Problem III.7.
- For both experiments, the initial parameter set is restricted to $[-5, -5, -3, -3]^\top \leq \theta_i \leq [8, 8, 3, 3]^\top$, for each θ_i (the parameter for box i). For the infinite-box experiment, each box is restricted to be at least 1.25×1.25 in width/height.
- Sampling time is around 15 seconds per demonstration (for the U-shape experiment) and 10 seconds per demonstration (for the infinite-box experiment). Computation time for solving Problem III.7 is around 40 seconds (for the U-shape experiment) and 15-20 seconds (for the infinite-box experiment).
- The same data is used for training the neural network (7800 trajectories total for the U-shape case, 2000 trajectories for the infinite-box case). The neural network architecture used for this example is a fully connected (FC) layer, 2×10 $\rightarrow$ LSTM, $10 \times 10 \rightarrow$ FC 10×1 (the recurrent layer is used since we have variable length trajectories as training input). The network is trained using Adam.

U-shape with random demonstrations:

- The system dynamics are $x_{t+1} \doteq [\chi_{t+1}, y_{t+1}]^\top = [\chi_t, y_t]^\top + [u_t^\chi, u_t^y]^\top$ with control constraints $\|[u_t^\chi, u_t^y]\|_2 \leq 0.5$.
- The cost function is $c(\xi_x, \xi_u) = \sum_{i=1}^{T-1} \|x_{t+1} - x_t\|_2^2$.

- Demonstrations are generated for 35 pairs of start/goal states sampled uniformly at random over $(\chi, y) \in [-2, 2] \times [-2, 2]$, rejecting any start/goal states that lie in $\mathcal{A}$.

- Since the cost function has optimal substructure, 10000 unsafe trajectories for each sub-trajectory are sampled. The dataset is downsampled to 25 unsafe trajectories for each sub-trajectory, which are to be fed into the multi-box variant of Problem III.7.

- The initial parameter set is restricted to $[-5, -5, -3, -3]^\top \leq \theta_i \leq [8, 8, 3, 3]^\top$, for each θ_i (the parameter for box i).

- Sampling time is around 2 minutes total. Computation time for solving the multi-box variant of Problem III.7 is around 90 seconds.

- The same data is used for training the neural network (10100 trajectories total). The neural network architecture used for this example is a fully connected (FC) layer, $2 \times 10 \rightarrow$ LSTM, $10 \times 10 \rightarrow$ FC 10×1. The network is trained using Adam.

A.3.2 High-dimensional examples

7-DOF arm, optimal/suboptimal demonstrations

- The system dynamics are $\doteq \theta^i_{t+1} = \theta^i_t + u^i_t$, $i = 1, \ldots, 7$, with control constraints $-2 \leq u^i_t \leq 2$, $i = 1, \ldots, 7$, where the state is $x = [\theta^1, \ldots, \theta^7]$.

- The cost function is $c(\xi_x, \xi_u) = \sum_{i=1}^{T-1} \|x_{t+1} - x_t\|_2^2$. Note that the generate demonstrations (displayed in Fig. 3.13) push up against the position constraint, since the trajectory minimizing joint-space path length without the position constraint is an arc that exceeds the bounds of the position constraint; the position constraint ends up increasing the cost by truncating that arc.

- The true safe set is $(x, y, z, \alpha, \beta, \gamma) \in [-0.51, 0.51] \times [-0.3, 1.1] \times [-0.51, 0.51] \times [-\pi, \pi] \times [-\pi/120, \pi/120] \times [-\pi/120, \pi/120]$ for the optimal case and the true safe set is $(x, y, z, \alpha, \beta, \gamma) \in [-0.57, 0.47] \times [-0.10, 1.17] \times [-0.56, 0.56] \times [-\pi, \pi] \times [-0.12, 0.12] \times [-0.125, 0.125]$ for the suboptimal case.

- Since the cost function has optimal substructure, 250000 unsafe trajectories for each sub-trajectory are sampled. For the suboptimal case, the continuous-time demonstrations are time-discretized down to $T = 10$ time-steps. The dataset is downsampled to 500 unsafe trajectories for each sub-trajectory, which are to be fed into Problem III.7.

- For the optimal case, the demonstrations are obtained by solving trajectory optimization problems solved with the IPOPT solver *Wächter and Biegler* (2006). For the suboptimal case, the demonstrations are recorded in a virtual reality (VR) environment displayed in Fig. A.7.

Figure A.7: VR setup. **Top**: VR environment as viewed from the Vive headset. The green box represents the position constraints on the end effector. The end effector is commanded to move by dragging it with the HTC Vive controllers (**bottom**).

- The initial parameter set is restricted to $[-1.5, -1.5, -1.5, -\pi, -\pi, -\pi]^\top \leq [x, y, z, \alpha, \beta, \gamma]^\top \leq [1.5, 1.5, 1.5, \pi, \pi, \pi]^\top$.

- Sampling time is 12.5 minutes total for the optimal case and 9 minutes total for the suboptimal case. Computation time for solving Problem III.3 is around 2 seconds for both the optimal/suboptimal case.

- The same data is used for training the neural network (70000 trajectories total for the optimal case, 49900 trajectories total for the suboptimal case). The neural network architecture used for this example is a fully connected (FC) layer, $3 \times 20 \rightarrow$ LSTM, $20 \times 20 \rightarrow$ FC 20×1. The network is trained using Adam.

12D quadrotor example

- The system dynamics *Sabatino* (2015) are

$$\begin{bmatrix} \dot{\chi} \\ \dot{y} \\ \dot{z} \\ \dot{\alpha} \\ \dot{\beta} \\ \dot{\gamma} \\ \ddot{\chi} \\ \ddot{y} \\ \ddot{z} \\ \ddot{\alpha} \\ \ddot{\beta} \\ \ddot{\gamma} \end{bmatrix} = \begin{bmatrix} \dot{\chi} \\ \dot{y} \\ \dot{z} \\ \dot{\beta}\frac{\sin(\gamma)}{\cos(\beta)} + \dot{\gamma}\frac{\cos(\gamma)}{\cos(\beta)} \\ \beta\cos(\gamma) - \dot{\gamma}\sin(\gamma) \\ \dot{\alpha} + \dot{\beta}\sin(\gamma)\tan(\beta) + \dot{\gamma}\cos(\gamma)\tan(\beta) \\ -\frac{1}{m}[\sin(\gamma)\sin(\alpha) + \cos(\gamma)\cos(\alpha)\sin(\beta)]u_1 \\ -\frac{1}{m}[\cos(\alpha)\sin(\gamma) - \cos(\gamma)\sin(\alpha)\sin(\beta)]u_1 \\ g - \frac{1}{m}[\cos(\gamma)\cos(\beta)]u_1 \\ \frac{I_y - I_z}{I_x}\dot{\beta}\dot{\gamma} + \frac{1}{I_x}u_2 \\ \frac{I_z - I_x}{I_y}\dot{\alpha}\dot{\gamma} + \frac{1}{I_y}u_3 \\ \frac{I_x - I_y}{I_z}\dot{\alpha}\dot{\beta} + \frac{1}{I_z}u_4 \end{bmatrix}, \tag{A.4}$$

  with control constraints $[0, -0.02, -0.02, -0.02]^\top \leq u_t \leq [mg, 0.02, 0.02, 0.02]^\top$. For our purposes, we convert the dynamics to discrete time by performing forward Euler integration with discretization time $\delta t = 0.4$ seconds. The state is $x = [\chi, y, z, \alpha, \beta, \gamma, \dot{x}, \dot{y}, \dot{z}, \dot{\alpha}, \dot{\beta}, \dot{\gamma}]^\top$, and the constants are $g = -9.81\text{m/s}^2$, $m = 1\text{kg}$, $I_x = 0.5\text{kg} \cdot \text{m}^2$, $I_y = 0.1\text{kg} \cdot \text{m}^2$, and $I_z = 0.3\text{kg} \cdot \text{m}^2$.

- The known unsafe set in (χ, y, z) is $(\chi, y, z) \notin [-0.5, 0.5] \times [-0.5, 0.5] \times [-0.5, 0.5]$.

- The true safe set in $(\dot{\alpha}, \dot{\beta}, \dot{\gamma})$ is $(\dot{\alpha}, \dot{\beta}, \dot{\gamma}) \in [-0.006, 0.006]^3$.

- The cost function is

$$c(\xi_x, \xi_u) = \sum_{i=1}^{T-1} \|[\chi_{i+1}, y_{i+1}, z_{i+1}, \dot{\alpha}_{i+1}, \dot{\beta}_{i+1}, \dot{\gamma}_{i+1}]^\top - [\chi_i, y_i, z_i, \dot{\alpha}_i, \dot{\beta}_i, \dot{\gamma}_i]^\top\|_2$$

  (penalizing acceleration and path length).

- The demonstrations are obtained by solving trajectory optimization problems solved with the IPOPT solver *Wächter and Biegler* (2006).

- Since the cost function has optimal substructure, 10000 unsafe trajectories for each sub-trajectory are sampled. The dataset is downsampled to 500 unsafe trajectories for each sub-trajectory, which are to be fed into Problem III.7.

- The initial parameter set is restricted to $[-\pi/2, -\pi/2, -\pi/2]^\top \leq [\dot{\alpha}, \dot{\beta}, \dot{\gamma}]^\top \leq [\pi/2, \pi/2, \pi/2]^\top$.

- Sampling time is 8.5 minutes total for the optimal case and 9 minutes total for the suboptimal case. Computation time for solving Problem III.3 is 12 seconds.

- The same data is used for training the neural network (30000 trajectories total). The neural network architecture used for this example is a fully connected (FC) layer, $6 \times 36 \rightarrow$ LSTM, $36 \times 42 \rightarrow$ FC 42×1. The network is trained using Adam.

A.3.3 Black-box system dynamics

Pushing example

- The cost function is $c(\xi_x, \xi_u) = \sum_{i=1}^{T-1} \|x_{t+1} - x_t\|_2^2$. The two demonstrations are manually generated and are not exactly optimal.
- 1000 unsafe trajectories for each demonstrations are sampled.
- The initial parameter set is restricted to $[-5, -5, -3, -3]^\top \leq \theta_i \leq [8, 8, 3, 3]^\top$.
- Sampling time is 2 hours for each demonstration (using the simulator is slower than using the closed form dynamics). Computation time for solving Problem III.3 is around 1 second.
- Demonstrations are time-discretized to 40 simulator timesteps when input to Problem III.7.
- The same data is used for training the neural network (2700 trajectories total). The neural network architecture used for this example is a fully connected (FC) layer, $8 \times 10 \rightarrow$ FC, $10 \times 10 \rightarrow$ FC 10×1. No recurrent layer is used this time since all trajectories are of the same length (no sub-trajectories were sampled this time due to speed). The network is trained using Adam.

APPENDIX B

Appendix for Chapter 7: Uncertainty-Aware Constraint Learning and Planning via Constraint Beliefs

In these appendices, we will first summarize and provide more details on the optimization problems used in our method (Appendix B.1, discuss various results on the representability of sets of cost function and constraint parameters which are consistent with demonstrations (Appendix B.2), provide expanded details on our methods for extracting the set of consistent cost function and constraint parameters (Appendix B.3), provide expanded details on our methods for planning policies for adaptively satisfying uncertain constraint parameters (Appendix B.4), provide proofs for the theoretical results in the main body (Appendix B.5), and provide additional details on our experimental results (Appendix B.6).

B.1 Appendix: Chapter VII: Optimization problem glossary

In this appendix, we provide a detailed summary of the optimization problems utilized in our approach.

Problem VII.1: This is the optimization problem that we assume the demonstrator is solving to local-optimality. This problem involves a potentially task-dependent cost function $c_\Pi(\xi_{xu})$, where a task in this chapter is simply steering the system state from a start state x_0 to a goal state x_g while satisfying a set of constraints. This problem also involves a known shared constraint $\bar{\phi}(\xi_{xu}) \in \bar{\mathcal{S}}$ (which embeds known constraints which that shared across all tasks, such as the system dynamics) as well as a known task-dependent constraint $\phi_\Pi(\xi_{xu}) \in \mathcal{S}_\Pi$ (which embeds known constraints that are task-dependent, such as the start and goal state constraints). Finally, there is the unknown shared constraint $\phi(\xi_{xu}) \in \mathcal{S}(\theta)$ (unknown to the learner, but known to the demonstrator) which is parameterized by unknown parameters θ.

$$
\begin{array}{lll}
\underset{\xi_{xu}}{\text{minimize}} & c_\Pi(\xi_{xu}) & \\
\text{subject to} & \phi(\xi_{xu}) \in \mathcal{S}(\theta) \subseteq \mathcal{C} & \Leftrightarrow \quad \mathbf{g}_{\neg k}(\xi_{xu}, \theta) \leq \mathbf{0} \\
& \bar{\phi}(\xi_{xu}) \in \bar{\mathcal{S}} \subseteq \bar{\mathcal{C}}, \quad \phi_\Pi(\xi_{xu}) \in \mathcal{S}_\Pi \subseteq \mathcal{C}_\Pi & \Leftrightarrow \quad \mathbf{h}_k(\xi_{xu}) = \mathbf{0}, \quad \mathbf{g}_k(\xi_{xu}) \leq \mathbf{0}
\end{array}
$$

Problem VII.2: This is the inverse optimization problem that the learner solves to learn *one possible assignment* of the unknown constraints that are satisfied by the demonstrations. Specifically, the problem searches over the unknown constraint parameters and the Lagrange multipliers which together make the KKT conditions of each demonstration satisfied.

$$
\begin{array}{rl}
\text{find} & \theta, \mathcal{L} \doteq \{\boldsymbol{\lambda}_k^j, \boldsymbol{\lambda}_{\neg k}^j, \boldsymbol{\nu}_k^j\}_{j=1}^{N_s} \\
\text{subject to} & \{\mathrm{KKT}(\xi_j^{\mathrm{loc}})\}_{j=1}^{N_s}
\end{array}
$$

Problem VII.3: This problem returns the set of all consistent constraint parameters $\mathcal{F}_\theta$. Intuitively, this problem finds the largest set $\hat{\mathcal{F}}_\theta$, in the sense of set containment, of constraint parameters θ, such that the KKT conditions can be made to hold for those parameters. The problem is intractable in its most general form, motivating simpler variants Problem VII.4 and Problem B.4.

$$
\begin{array}{ll}
\sup\limits_{\hat{\mathcal{F}}_\theta} & \mathrm{Vol}(\hat{\mathcal{F}}_\theta) \\
\text{s.t.} & \forall \theta \in \hat{\mathcal{F}}_\theta, \quad \exists \{\boldsymbol{\lambda}_k^j, \boldsymbol{\lambda}_{\neg k}^j, \boldsymbol{\nu}_k^j \mid \mathrm{KKT}(\xi_j^{\mathrm{loc}})\}_{j=1}^{N_s}
\end{array}
$$

Problem VII.4: This problem returns the largest axis-aligned hyper-rectangle contained within $\mathcal{F}_\theta$; this problem is a restricted, tractable version of Problem VII.3.

$$
\begin{array}{ll}
\underset{s,\theta,\mathcal{L}}{\text{maximize}} & \left(\prod_i s_i\right)^{1/d} \\
\text{subject to} & \{\mathrm{KKT}_{\mathrm{rob}}^{\mathrm{box}}(\xi_j^{\mathrm{loc}})\}_{j=1}^{N_s}
\end{array}
$$

Problem VII.7: This problem solves a chance-constrained trajectory optimization problem, minimizing a possibly task-dependent objective $c_\Pi(\xi_{xu})$ while ensuring that the resulting trajectory satisfies the uncertain constraint with prescribed probability $1 - \varepsilon$. We further consider two specific variants, **Problem VII.7-$\varepsilon_{\mathbf{min}}$**, which seeks to solve Problem VII.7 to be as safe as possible, i.e with the smallest ε for which there exists a feasible solution, and **Problem VII.7-R**, which directly trades off performance and safety with a modified objective $c_\Pi(\xi_{xu})/\Pr(\xi_{xu} \text{ safe})$. Problem VII.7 and its variants are in their most general form intractable, motivating the simpler variant Problem VII.8.

Problem VII.7:

$$
\begin{array}{lll}
\min\limits_{\xi_{xu}} & c_\Pi(\xi_{xu}) & \text{(B.1a)} \\
\text{s.t.} & \bar{\phi}(\xi_{xu}) \in \bar{\mathcal{S}} \subseteq \bar{\mathcal{C}} & \text{(B.1b)} \\
& \phi_\Pi(\xi_{xu}) \in \mathcal{S}_\Pi \subseteq \mathcal{C}_\Pi & \text{(B.1c)} \\
& \Pr(\xi_{xu} \text{ safe}) \geq 1 - \varepsilon & \text{(B.1d)}
\end{array}
$$

Problem VII.7-$\varepsilon_{\min}$:

$$\begin{aligned} \min_{\xi_{xu}} \min_{\varepsilon} \quad & c_\Pi(\xi_{xu}) \\ \text{s.t.} \quad & \bar{\phi}(\xi_{xu}) \in \bar{\mathcal{S}} \subseteq \bar{\mathcal{C}} \\ & \phi_\Pi(\xi_{xu}) \in \mathcal{S}_\Pi \subseteq \mathcal{C}_\Pi \\ & \Pr(\xi_{xu} \text{ safe}) \geq 1 - \varepsilon \end{aligned}$$

Problem VII.7-R:

$$\begin{aligned} \min_{\xi_{xu}} \quad & c_\Pi(\xi_{xu}) / \Pr(\xi_{xu} \text{ safe}) \\ \text{s.t.} \quad & \bar{\phi}(\xi_{xu}) \in \bar{\mathcal{S}} \subseteq \bar{\mathcal{C}} \\ & \phi_\Pi(\xi_{xu}) \in \mathcal{S}_\Pi \subseteq \mathcal{C}_\Pi \end{aligned}$$

Problem VII.8: This is a simplified variant of Problem VII.7, which makes the probability constraint tractable by restricting the integration of probability mass over a fixed number of axis-aligned boxes, where the location and extents of the boxes are also optimized over. We consider two specific variants, **Problem VII.8-$\varepsilon_{\min}$** and **Problem VII.8-R**, which are as described for Problem VII.7.

$$\begin{aligned} \min_{\xi_{xu}, \mathcal{B}_i, t_i} \quad & c_\Pi(\xi_{xu}) && \text{(B.4a)} \\ \text{s.t.} \quad & \bar{\phi}(\xi_{xu}) \in \bar{\mathcal{S}} \subseteq \bar{\mathcal{C}},\ \phi_\Pi(\xi_{xu}) \in \mathcal{S}_\Pi \subseteq \mathcal{C}_\Pi && \text{(B.4b)} \\ & \xi_{xu} \in \mathcal{S}(\theta),\ \forall \theta \in \mathcal{B}_1, \ldots, \mathcal{B}_{N_{\text{box}}} && \text{(B.4c)} \\ & \mathcal{B}_i \cap \mathcal{B}_j = \emptyset,\ i \neq j, \quad \mathcal{B}_i \subseteq \mathcal{F}_\theta, \forall i && \text{(B.4d)} \\ & 0 \leq t_i \leq (\textstyle\prod_i b_i^{\text{scale}})^{1/d},\ i = 1, ..., N_{\text{box}} && \text{(B.4e)} \\ & \textstyle\sum_i t_i^d \geq (1 - \varepsilon) \text{Vol}(\mathcal{F}_\theta) && \text{(B.4f)} \end{aligned}$$

We provide an overview of the constraints of Problem VII.8. First, note that (B.4a)-(B.4b) exactly correspond to (B.1a)-(B.1c). The remaining constraints in Problem VII.8 implement the box-limited integration. Specifically, (B.4c) enforces that the planned trajectory ξ_{xu} is safe with respect to all θ belonging to $\mathcal{B}_1, \ldots, \mathcal{B}_{N_{\text{box}}}$. Recall that each $\mathcal{B}_i$ is meant to represent a box contained in $\mathcal{F}_\theta$, and that ξ_{xu} is to be safe with respect to all θ belonging to this $\mathcal{B}_i$. Thus, (B.4d) further enforces that each $\mathcal{B}_i$ is contained in $\mathcal{F}_\theta$, and furthermore, that the $\mathcal{B}_i$ are disjoint; this is to avoid any double-counting of box volumes (and thus probability mass). Next, (B.4e) introduces the variables t_i, from which the volume of the corresponding box can be recovered: note that the volume of $\mathcal{B}_i$ is $\prod_i b_i^{\text{scale}}$. The last constraint, (B.4f), does exactly this: t_i^d is exactly the volume of $\mathcal{B}_i$, so $\sum_i t_i^d = \sum_i \text{Vol}(\mathcal{B}_i)$, which we ensure is at least $(1 - \varepsilon)\text{Vol}(\mathcal{F}_\theta)$ to satisfy the probability constraint.

Problem B.4: In Appendix B.3.1, we will discuss a modification of Algorithm VII.1 which makes it more efficient for constraint parameterizations other than the union-of-boxes parameterization assumed in Sec. 7.3. This modification hinges upon

Problem B.4, which returns a zonotope contained within $\mathcal{F}_\theta$ of approximately maximum volume; this problem is a restricted, tractable version of Problem VII.3 which is more general than Problem VII.4.

$$\begin{array}{ll} \underset{s,\theta,\boldsymbol{\lambda}_k^j,\boldsymbol{\lambda}_{\neg k}^j,\boldsymbol{\nu}_k^j,Q_i}{\text{maximize}} & \sum_{i=1}^{N_{\text{gen}}} \|\ell_i\|_1 \\ \text{subject to} & \{\text{KKT}_{\text{rob}}^{\text{zon}}(\xi_j^{\text{loc}})\}_{j=1}^{N_s}, \quad |\ell_m^\top \ell_n| \leq \delta, \ \forall m \neq n \end{array}$$

B.2 Appendix: Chapter VII: A geometric analysis of constrained inverse optimal control

We describe how the constraint learning problem can be extended to also learn unknown cost function parameters γ (Appendix B.2.1), the shape of the resulting feasible sets for unknown cost function parameters for various parameterizations (Appendix B.2.2), and the shape of the resulting feasible sets for consistent constraint parameters (Appendix B.2.3), for various parameterizations.

B.2.1 Modifying Problem VII.2 to handle unknown cost function parameters

As in Chapter IV, we note that the KKT conditions in Problem VII.2 can be modified to handle unknown cost function parameters, where the cost function can be written as $c_\Pi(\xi_{xu}, \gamma)$ for unknown cost function parameters $\gamma \in \Gamma$, with few changes: the only KKT condition that changes is stationarity (7.3d), where the term involving the gradient of the cost now also involves the unknown cost function parameters γ:

$$\nabla_{\xi_{xu}} c_\Pi(\xi_j^{\text{loc}}, \gamma) + \boldsymbol{\lambda}_k^{j\top} \nabla_{\xi_{xu}} \mathbf{g}_k(\xi_j^{\text{loc}}) + \boldsymbol{\lambda}_{\neg k}^{j\top} \nabla_{\xi_{xu}} \mathbf{g}_{\neg k}(\xi_j^{\text{loc}}, \theta) + \boldsymbol{\nu}_k^{j\top} \nabla_{\xi_{xu}} \mathbf{h}_k(\xi_j^{\text{loc}}) = \mathbf{0} \quad \text{(B.5)}$$

B.2.2 Unknown cost function, known constraint

Let us denote the feasible set of Problem VII.2, modified to handle unknown cost function parameters, again be $\mathcal{F}$, and let us denote the projection of $\mathcal{F}$ onto Γ as $\mathcal{F}_\gamma$:

$$\mathcal{F}_\gamma \doteq \{\gamma \mid \exists(\theta, \mathcal{L}) : (\theta, \gamma, \mathcal{L}) \in \mathcal{F}\} \quad \text{(B.6)}$$

In the following, we will analyze the shape of $\mathcal{F}_\gamma$ for various parameterizations the unknown cost function. In this subsection, we will assume that the constraint parameters θ are known, and focus only on analyzing the case of unknown γ.

B.2.2.1 Linear cost function parameterization

Consider the case where the cost function $c(\xi_{xu}, \gamma)$ is linear in the unknown parameters γ, i.e. $c(\xi_{xu}, \gamma) = \sum_{i=1}^{|\gamma|} \gamma_i c_i(\xi_{xu}) \doteq \gamma^\top c(\xi_{xu})$, and the constraints are fully known. The following result shows that in this setting, the set of cost function parameters consistent with the KKT conditions (7.3) is convex and closed under nonnegative scaling:

Theorem B.1 (Geometry of $\mathcal{F}_\gamma$ (linear in γ, known θ))**.** *If the cost function takes the form* $c(\xi_{xu}, \gamma) = \sum_{i=1}^{|\gamma|} \gamma_i c_i(\xi_{xu})$*, then* $\mathcal{F}_\gamma = \Gamma \cap \mathcal{C}$*, where* $\mathcal{C} \in \mathbb{R}^{|\gamma|}$ *is a convex cone in* γ*-space.*

Proof. From *Boyd and Vandenberghe* (2004), a set $\mathcal{C}$ is a convex cone if for all γ_1, γ_2 in $\mathcal{C}$, $\alpha_1\gamma_1 + \alpha_2\gamma_2 \in \mathcal{C}$, for all nonnegative scalars α_1, $\alpha_2 \geq 0$. For now, assume that other than the KKT constraints, γ is unconstrained, i.e. $\Gamma = \mathbb{R}^{|\gamma|}$. Suppose we have $\gamma_1 \in \mathcal{F}_\gamma$ and $\gamma_2 \in \mathcal{F}_\gamma$. In (7.3), as the constraint parameters θ are known, we drop (7.3a), merge (7.3b)-(7.3c), and absorb $\boldsymbol{\lambda}_{\neg k}^{j\top} \nabla_{\xi_{xu}} \mathbf{g}_{\neg k}(\xi_j^{\text{loc}}, \theta)$ into the previous term. Then, if $\gamma_i \in \mathcal{F}_\gamma$ for $i \in \{1, 2\}$, we know that $\boldsymbol{\lambda}_{k,i}^j \geq \mathbf{0}$, $\boldsymbol{\lambda}_{k,i}^j \odot \mathbf{g}_k(\xi_j^{\text{dem}}) = \mathbf{0}$, and $\gamma_i^\top \nabla_{\xi_{xu}} c(\xi_j^{\text{loc}}) + \boldsymbol{\lambda}_{k,i}^{j\top} \nabla_{\xi_{xu}} \mathbf{g}_k(\xi_j^{\text{loc}}) + \boldsymbol{\nu}_{k,i}^{j\top} \nabla_{\xi_{xu}} \mathbf{h}_k(\xi_j^{\text{loc}}) = \mathbf{0}$, for $i \in \{1, 2\}$. Then if $\gamma = \alpha_1\gamma_1 + \alpha_2\gamma_2$, for nonnegative scalars α_1 and α_2, we can select $\boldsymbol{\lambda}_k^j = \alpha_1 \boldsymbol{\lambda}_{k,1}^j + \alpha_2 \boldsymbol{\lambda}_{k,2}^j$ and $\boldsymbol{\nu}_k^j = \alpha_1 \boldsymbol{\nu}_{k,1}^j + \alpha_2 \boldsymbol{\nu}_{k,2}^j$ to satisfy (7.3d); it can also be verified algebraically that this choice of $\boldsymbol{\lambda}_k^j$ satisfies the nonnegativity and complementary slackness constraints. This implies the conic hull $\mathcal{C}$ of any feasible γ is feasible for Problem VII.2. Finally, if $\Gamma \subset \mathbb{R}^{|\gamma|}$, we can write $\mathcal{F}_\gamma$ as the intersection of Γ and the previously constructed cone $\mathcal{C}$. □

Furthermore, as Problem VII.2 simplifies to a linear program when θ is known, $\mathcal{F}$ is a polytope, and thus $\mathcal{F}_\gamma$ can be directly computed via a polytopic projection of $\mathcal{F}$ onto its γ coordinates *Herceg et al.* (2013).

B.2.2.2 Nonlinear cost function parameterization

For general nonlinear parameterizations of γ, the set of γ which satisfy (7.3) is much more challenging to represent explicitly, as checking if a γ satisfies (7.3) will involve satisfying a set of nonlinear, non-convex equality constraints (7.3d). However, if the parameterization is a polynomial function of γ, i.e. $c(\xi_{xu}, \gamma)$ is a polynomial in γ for fixed ξ_{xu}, $\mathcal{F}_\gamma$ can be represented as a semi-algebraic set; that is, a set described by a finite union of intersections of polynomial inequalities:

Theorem B.2. *For cost functions which are polynomial in* γ *for fixed* ξ_{xu}*,* $\mathcal{F}_\gamma = \Gamma \cap \mathcal{P}$*, where* $\mathcal{P} \in \mathbb{R}^{|\gamma|}$ *is a semi-algebraic set in* γ*-space.*

Proof. In this setting, $\boldsymbol{\lambda}_k^j \geq \mathbf{0}$, $\boldsymbol{\lambda}_k^j \odot \mathbf{g}_k(\xi_j^{\text{dem}}) = \mathbf{0}$, and $\nabla_{\xi_{xu}} c(\xi_j^{\text{loc}}, \gamma) + \boldsymbol{\lambda}_k^{j\top} \nabla_{\xi_{xu}} \mathbf{g}_k(\xi_j^{\text{loc}}) + \boldsymbol{\nu}_k^{j\top} \nabla_{\xi_{xu}} \mathbf{h}_k(\xi_j^{\text{loc}}) = \mathbf{0}$ define an intersection of polynomial inequalities in γ, that is, a basic semi-algebraic set. $\mathcal{F}_\gamma$ is then the projection of this set onto the γ coordinates; however, the projection of a basic semi-algebraic set may not be basic semi-algebraic in general; they are guaranteed to be semi-algebraic via the Tarski-Seidenberg theorem *Bochnak et al.* (1998). □

While explicitly representing a semi-algebraic set can be expensive, there exist well-established methods for doing so, including exact methods like cylindrical algebraic decomposition *Collins* (1975) and approximate methods involving semidefinite relaxations *Magron et al.* (2015).

B.2.3 Unknown constraints

In this section, we discuss details on the shape of $\mathcal{F}_\theta$ for unions of offset-parameterized constraints (Sec. B.2.3.1) and for unions of affine and higher-order parameterized constraints (Sec. B.2.3.2).

B.2.3.1 Unions of offset-parameterized constraints

Coordinate-independent parameterization:
We first analyze the case where the unknown constraint can be described as a union of offset-parameterized constraints:

Theorem B.3. *Consider the case when the unknown constraint can be described as a union of intersection of inequalities which are offset-parameterized, i.e.:*

$$\mathcal{S}(\theta) = \left\{ \kappa \in \mathcal{C} \mid \bigvee_{m=1}^{N_{ineq}} \bigwedge_{n=1}^{N_{ineq}^i} \Big(g_{mn}(\kappa) \leq \theta_{mn} \Big) \right\} \tag{B.7}$$

Then, the corresponding $\mathcal{F}_\theta$ can be described as a union of boxes as well:

$$\mathcal{F}_\theta = \left\{ \theta \in \Theta \mid \bigcup_{m=1}^{N_{box}} [I_{d\times d}, -I_{d\times d}]^\top \theta \leq s_m \right\} \tag{B.8}$$

Proof. In this setting, note that Problem VII.2 can be represented as a MILP, implying that $\mathcal{F}$ can be described as a finite union of polyhedra in the space of all decision variables. As polyhedra are closed under projection, $\mathcal{F}_\theta$ can also be represented as a finite union of polyhedra. Note that in the KKT conditions for parameterization (B.7), θ only appears in (7.3a) and (7.3c) (as the gradient term in (7.3d) drops out). Now, suppose we fix all of the boolean variables in Problem VII.2 (needed to implement (7.3a) and (7.3c)). Then, depending on those boolean variables for some t, j, (7.3c) either enforces $\theta_{mn} = g_{mn}(\kappa_t^j)$ or leaves θ_{mn} unconstrained, and (7.3a) imposes $\theta_{mn} \geq g_{mn}(\kappa)$ or leaves θ_{mn} unconstrained. Then, for fixed boolean variables, for any m and n, we obtain linear constraints on θ_{mn}, independent of other $\theta_{m'n'}$ for $m' \neq m$, $n' \neq n$; that is, θ_{mn} is constrained to lie in an interval. Then, unioning over all feasible boolean assignments, we obtain that $\theta_{m'n'}$ can be described as a finite union of intervals. As a result, $\mathcal{F}_\theta$ can be described as a union of boxes in θ space, as in (B.8). □

Coordinate-dependent parameterization: If instead the constraint is parameterized such that any single constraint can depend on multiple parameters:

$$\mathcal{S}(\theta) = \left\{ \kappa \in \mathcal{C} \mid \bigvee_{m=1}^{N_{\text{ineq}}} \bigwedge_{n=1}^{N_{\text{ineq}}^i} \Big(g_{mn}(\kappa) \leq \omega_{mn}^\top \theta \Big) \right\} \tag{B.9}$$

for some fixed mixing coefficients ω_{mn}, $\mathcal{F}_\theta$ can only be represented as the more general

union of polytopes, as for some m, n, the constraints on θ_{mn} will in general depend on $\theta_{m'n'}$, for $m' \neq m$, $n' \neq n$.

B.2.3.2 Unions of affine and higher-degree parameterized constraints

Consider the case where the constraint can be represented as:

$$\mathcal{S}(\theta) = \left\{ \kappa \in \mathcal{C} \mid \bigvee_{m=1}^{N_{\text{ineq}}} \bigwedge_{n=1}^{N_{\text{ineq}}^i} \Big(g_{mn}(\kappa, \theta_{mn}) \leq 0 \Big) \right\} \tag{B.10}$$

where $g_{mn}(\kappa, \theta_{mn})$ is affine or higher-degree in θ_{mn}. In this case, the gradient term does not drop out in (7.3d), meaning the set of consistent θ will depend on the projection of a set defined by polynomial equality constraints, which in general is a semialgebraic set described by polynomials of degree $2^{O(|\text{decision variables}|)}$, where "decision variables" denotes the Lagrange multipliers $\mathcal{L}$ and unknown constraint parameters θ in Problem VII.2.

B.3 Appendix: Chapter VII: Obtaining a belief over constraints (expanded)

In this section, we first discuss details on zonotope extraction (Appendix B.3.1), how extraction can be done for the case of jointly unknown cost function and constraints (Appendix B.3.2), how extraction can be sped up with parallelization (Appendix B.3.3), and conclude with a summary of complexity and representability for the extraction problems induced by various cost and constraint parameterizations (Appendix B.3.4).

B.3.1 Other constraint parameterizations: extracting with zonotopes

While filling $\mathcal{F}_\theta$ with boxes may be efficient for a union-of-boxes constraint parameterization, infinitely many boxes may be needed to cover $\mathcal{F}_\theta$ in more general cases where $\mathcal{F}_\theta$ may only be representable as a union of polytopes or semialgebraic sets (see Appendix B.3 for more details). To address this, we describe how to extract $\mathcal{F}_\theta$ with shapes more general than boxes while retaining efficiency. Covering $\mathcal{F}_\theta$ with polytopes instead is expensive, as polytope volume computation in high dimensions is hard. Instead, we cover $\mathcal{F}_\theta$ with zonotopes *Beck and Robins* (2015), which are between boxes and polytopes in representational power. A zonotope $\mathcal{Z}$ is a Minkowski sum of N_{gen} line segments: $\mathcal{Z} = \{\sum_{i=1}^{N_{\text{gen}}} \ell_i u_i \mid u_i \in [-1, 1]\}$, where $\ell_i \in \mathbb{R}^d$ is the ith segment.

Problem B.4 (Zonotope robustification)**.**

$$\begin{array}{ll} \underset{s,\theta,\boldsymbol{\lambda}_k^j,\boldsymbol{\lambda}_{\neg k}^j,\boldsymbol{\nu}_k^j,Q_i}{\text{maximize}} & \sum_{i=1}^{N_{\text{gen}}} \|\ell_i\|_1 \\ \text{subject to} & \{\text{KKT}_{\text{rob}}^{\text{zon}}(\xi_j^{\text{loc}})\}_{j=1}^{N_s} \\ & |\ell_m^\top \ell_n| \leq \delta, \ \forall m \neq n \end{array}$$

Robustifying KKT to a zonotope uncertainty is done similarly to boxes: the robust constraint can be simplified with these equivalences: $a^\top(x + \sum_{i=1}^{N_{\text{gen}}} \ell_i u_i) \leq b, \forall u_i \in [-1,1] \Leftrightarrow \max_{u_1 \in [-1,1],\ldots,u_{N_{\text{gen}}} \in [-1,1]} a^\top(x + \sum_i \ell_i u_i) \leq b \Leftrightarrow a^\top x + \sum_i |a^\top \ell_i| \leq b$. Denote (7.3b) and the robustified (7.3a), (7.3c), (7.3d) as $\text{KKT}^{\text{zon}}_{\text{rob}}(\xi_j^{\text{dem}})$. Optimizing zonotope volume is challenging, as it requires determinant computations *Beck and Robins* (2015) that render the overall problem a mixed integer semidefinite program, which lack reliable solvers. Instead, we optimize a surrogate, $\sum_i \|\ell_i\|_1$, and add bilinear optimization constraints to make the lines approximately orthogonal: $|\ell_m^\top \ell_n| \leq \delta$ for some small predetermined δ; these constraints are compatible with off-the-shelf solvers *Gurobi Optimization* (2020). Finally, we cannot ignore the existential quantifiers for this parameterization, so we introduce "feedback" Lagrange multipliers. Inspired from adjustable robust optimization *Ben-Tal et al.* (2004), we modify each Lagrange multiplier to take the form $\lambda_i + Q_i u$, where $Q_i u$ is a feedback term adjusting the value of the Lagrange multiplier as a linear function of the uncertainty u. The Q_i are jointly optimized to maximize the volume. The overall problem (Prob. B.4) is a mixed integer bilinear program (MIBLP).

Discussion on volume maximization: Recall from the statement of Problem B.4 that we are enforcing the approximate orthogonality $|\ell_m^\top \ell_n| \leq \delta$ of the line segment generators together with maximizing the line segment norms as a surrogate for volume maximization; this is to avoid the degenerate case where any two generators are parallel; this leads to the extracted volume being zero. Furthermore, we elect to use a prespecified δ instead of enforcing mutual orthogonality $\ell_m^\top \ell_n = 0$ to avoid restricting the search to rotated boxes (which is what occurs if all the generators are perfectly orthogonal).

B.3.2 Discussion on extracting with mixed cost function and constraint uncertainty

Extraction with mixed cost function and constraint uncertainty can be done in a similar way to Alg. VII.1. Specifically, we can robustify the stationarity condition to uncertainties in γ and θ jointly:

$$\nabla_{\xi_{xu}} c_\Pi(\xi_j^{\text{loc}}, \gamma + s_c \odot u_c) + \boldsymbol{\lambda}_k^{j\top} \nabla_{\xi_{xu}} \mathbf{g}_k(\xi_j^{\text{loc}}) + \boldsymbol{\lambda}_{\neg k}^{j\top} \nabla_{\xi_{xu}} \mathbf{g}_{\neg k}(\xi_j^{\text{loc}}, \theta + s \odot u) + \boldsymbol{\nu}_k^{j\top} \nabla_{\xi_{xu}} \mathbf{h}_k(\xi_j^{\text{loc}}) = \mathbf{0} \tag{B.11}$$

The remaining KKT conditions are unchanged, as γ does not factor into the other constraints. The modified stationarity condition can be robustified in a similar way. We denote (7.3b), the robustified (7.3a) and (7.3c), and the joint cost/constraint-robustified (7.3d) together as $\text{KKT}^{\text{box}}_{\text{rob,cost}}(\xi_j^{\text{dem}})$. We can then modify Problem VII.4 to account for the additional scaling variables as such:

$$\begin{aligned} \underset{s, s_c, \theta, \gamma, \mathcal{L}}{\text{maximize}} \quad & \left(\textstyle\prod_i s_i \prod_i s_{c_i}\right)^{1/d} \\ \text{subject to} \quad & \{\text{KKT}^{\text{box}}_{\text{rob,cost}}(\xi_j^{\text{loc}})\}_{j=1}^{N_s} \end{aligned}$$

This new optimization problem can be integrated into Algorithm VII.1, repeatedly carving out subsets of $\mathcal{F}_\theta \times \mathcal{F}_\gamma$.

B.3.3 Speeding up extraction with parallelization

We sketch one possible way that Alg. VII.1 can be sped up with parallelization on M cores:

- Partition the parameter space Θ into M disjoint boxes.
- Run Alg. VII.1 on each partition separately.
- Reconstruct $\mathcal{F}_\theta$ by unioning the extracted parameters from each partition.

B.3.4 Summary on problem complexity

In the following table, we organize the representability of particular constraint learning and feasible set extraction problems, for various constraint parameterizations. To summarize, the case of unknown constraints induces a set of integer variables due to the complementary slackness condition. The remaining complexity depends on the complexity of the constraint and cost function parameterizations. Furthermore, the extraction problems are only conic-representable due to our formulation of volume maximization.

Constraint param.	Cost param.	Problem VII.2, class	Class (extraction)
Known	Linear	LP	Direct projection
Union of offsets	Known	MILP	MISOCP
Union of offsets	Linear	MILP	MISOCP
Union of affine	Known	MIBLP	MIBLP
Union of affine	Linear	MIBLP	MIBLP
Nonlinear	Nonlinear	MINLP	MINLP

B.4 Appendix: Chapter VII: Policies for adaptive constraint satisfaction (expanded)

In this appendix, we first discuss fast reformulations for the chance-constrained planning problem (Appendix B.4.1), expanded details on the sampling-based planners that we use when the optimization constraints are not MICP-representable (Appendix B.4.2), discussion on extending Problem VII.8 to handle priors other than the uniform distribution (Appendix B.4.3), and discussion on how to perform belief updates for various constraint sensing modalities (Appendix B.4.4).

B.4.1 Fast reformulations of Problem VII.8

As written, Problem VII.8 can be expensive to solve due to the many possible assignments of the box decision variables b_i^{cen} and b_i^{scale}, for each i, and the combinatorial coupling between boxes $i \neq j$. Furthermore, the non-convex norm constraint (7.11f) causes Problem VII.8 to be an MIBLP, which are in general more challenging to solve than mixed integer convex programs. This can cause solving Problem VII.8 to be slow. To address this, we propose two reformulations:

- In the first, we still optimize over the N_{box} boxes simultaneously, and simply replace the non-convex constraint (7.11f) with a linear approximation: $\sum_i t_i \geq 1-\varepsilon$. However, by doing this, the original chance-constraint $\Pr(\xi_{xu} \text{ safe}) \geq 1-\varepsilon$ may not hold exactly. One can get around this by instead enforcing $\sum_i t_i \geq 1-\tilde{\varepsilon}$, incrementally shrinking $\tilde{\varepsilon}$ until the resulting trajectory satisfies the original chance constraint, but this can be cumbersome.

- The second reformulation, which is what we use in practice to solve Problem VII.8-$\varepsilon_{\min}$, instead optimizes over the boxes one at a time. That is, if we are given a budget of N_{box} boxes, we solve Problem VII.8-$\varepsilon_{\min}$ for $N_{\text{box}} = 1$, then at the next iteration, solve for $N_{\text{box}} = 2$, where the first box is fixed. This continues until we reach the box budget. If N_{box} is chosen to be large enough that the covered probability mass is the same when solving Problem VII.8 for N_{box} and $N_{\text{box}} + 1$, this sequential strategy is lossless, i.e. will return the same solution as Problem VII.8 without relaxations, for the case where choosing to satisfy one possible constraint can never cause the trajectory to not be able to satisfy a different constraint. In other cases, this strategy may lead to convergence to a local optimum in the amount of probability mass covered, but we observe that this strategy works well in practice.

B.4.2 Sampling-based planners

We utilize two sampling-based planners to compute open-loop plans in the case where the constraints of Problem VII.8 are not representable in a mixed integer convex program: Minimum Constraint Removal (MCR) and the Blindfolded Traveler's Problem (BTP).

Minimum Constraint Removal (MCR) *Hauser* (2014): MCR takes a *finite* set of constraints and a start/goal state. At each step of the algorithm, MCR keeps track of the path from the start to the goal that violates the least number of constraints so far. The algorithm initializes this with the straight-line edge between the start and goal states, and sets k, the minimum number of constraints that must be violated as the number of constraints that the start/goal state violate. Then, MCR incrementally grows a roadmap, where candidate expansions are limited based on the number of constraints the candidate edge will violate, and at each growth iteration, finds the path from the start to each vertex which violates the minimum number of constraints and updates the path to the goal which violates the least constraints. Every so often, we increase k to allow for more constraints to be violated when expanding the roadmap. This is summarized in Section 4.2 of *Hauser* (2014).

To use MCR to approximate Problem VII.8-$\varepsilon_{\max}$, we sample N_{sample} constraints from the belief $\{\theta_i \sim b(\theta)\}_{i=1}^{N_{\text{sam}}}$ as input to MCR, then run MCR as just described. The output of MCR will then be a path on the roadmap connecting the start and goal which violates the minimal number of *sampled constraints*.

The Blindfolded Traveler's Problem (BTP) *Saund et al.* (2019): The Blindfolded Traveler's Problem (BTP) can be modeled as a graph search problem. It is defined by a graph $G = (V, E, W, x_0, x_g)$ with vertices V, edges E, weights C, and

start/goal state x_0 and x_g. Furthermore, each edge e is invalid with probability $p(e) \in [0, 1]$. If an edge e_{uv} from u to v is traversed, either the traversal is successful, and the agent ends up at vertex v, or the agent discovers that the edge is invalid $\eta_{uv} \in [0, 1]$ fraction of the way through, at which point the agent needs to turn back and return to vertex u; such a traversal attempt costs $2\eta_{uv}C_{uv}$, where $c_\Pi(e_{uv})$ is the cost of traversing edge uv. The agent has a prior over the probability of edge validity and blockage, which can be updated based on the observations gained through traversing the graph. A solution to BTP is a policy which takes the start state, history of observations, and outputs an edge to traverse.

While computing an optimal policy for the BTP is NP-complete, it is possible to compute high-quality approximations, for example using the Collision Measure strategy (Section 5.1 of *Saund et al.* (2019)). This strategy approximates BTP by modifying the graph edge weights to penalize the log probability of the edges being unsafe, that is, edge weights are modified such that $\tilde{c}_{uv} = c_{uv} - \beta \log(p(e_{uv} \text{ safe})$, for some weights β. We then compute paths on the graph by running A* with the modified edge weights.

Specifically, to approximate Problem VII.8-R with BTP, we sample constraints $\{\theta_i \sim b(\theta)\}_{i=1}^{N_{\text{sam}}}$ from the belief $b(\theta)$ to approximate the probabilities $p(e_{uv} \text{ safe})$; specifically, for each edge, we estimate $p(e_{uv} \text{ safe})$ as the fraction of sampled constraints which are violated.

B.4.3 Priors $p(\theta)$ other than the uniform distribution

In Section 7.4.1, we discuss how to integrate over a *uniform prior* to optimize boxes over the probability density which can be embedded in an MISOCP for planning probabilistically safe trajectories. Here, we discuss a different prior which also satisfies the closed-form integrability and log-concave assumptions detailed in Sec. 7.4.1: $p(\theta) \propto \prod_{i=1}^{|\kappa|}(\bar{\kappa}_i(\theta) - \underline{\kappa}_i(\theta))$, where $\bar{\kappa}_i$, $\underline{\kappa}_i$ denote the upper and lower bounds of the box in dimension i of the constraint space. This prior places more probability mass on larger box constraints; hence, the behavior generated using this prior is more conservative. As a concrete example, see Fig. B.1 for trajectories solving Problem VII.8 for the different priors, solved over a range of different start/goal states. Observe that the trajectories that use the weighted prior are more conservative. Generally, an investigation of other useful priors which satisfy our assumption of closed-form integrability is the subject of future work.

B.4.4 Belief updates

Given an initial set of consistent constraint parameters $\mathcal{F}_\theta$, we are interested in updating $\mathcal{F}_\theta$ to be consistent with constraint information gathered in execution. We specifically write belief updates for the following constraint sensing modalities:

Direct, exact measurements: Consider a measurement that κ_{safe} is safe, given an initial set of consistent constraints $\mathcal{F}_\theta$. We want to find the maximum volume subset of $\mathcal{F}_\theta$ which satisfies $g(\kappa, \theta) \leq 0$. To accomplish this, we simply modify Problem VII.4 to:

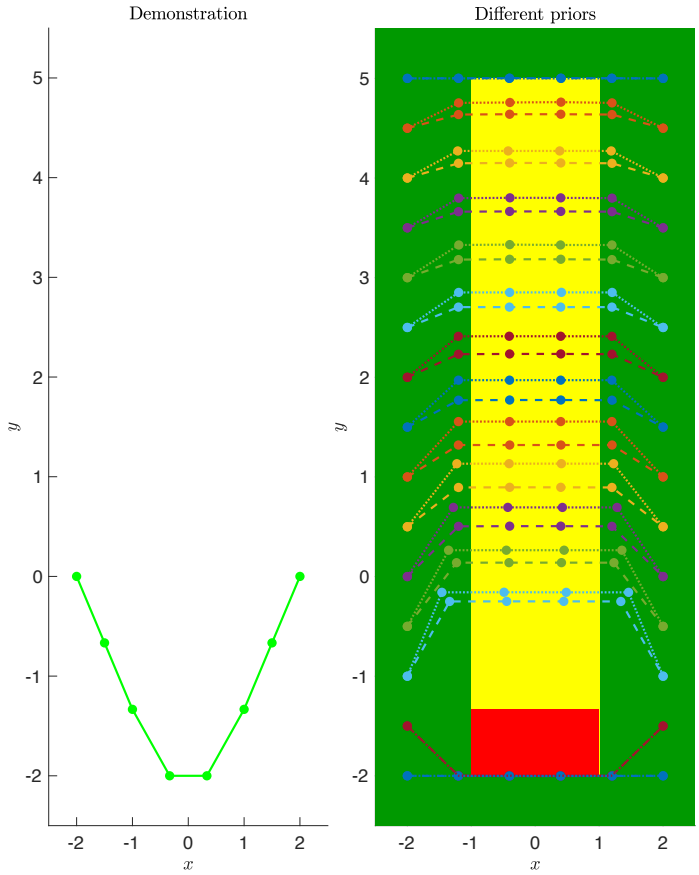


Figure B.1: Trajectories generated for different priors. We are provided one demonstration (left), which reveals the left, right, and bottom extents of a box obstacle constraint, but not the upper extent. Dashed lines correspond to the uniform prior $p(\theta) \propto 1$, while dotted lines correspond to the prior $p(\theta) \propto \prod_{i=1}^{|\kappa|}(\bar{\kappa}_i(\theta) - \underline{\kappa}_i(\theta))$.

$$\begin{array}{ll} \underset{s}{\text{maximize}} & \left(\prod_i s_i\right)^{1/d} \\ \text{subject to} & g(\kappa_{\text{safe}}, \theta + s \odot u) \leq 0 \end{array}$$

where we can eliminate the uncertain variable u with the identity used in Section 7.3.1, and use this modified problem in Algorithm VII.1. Note that as the local optimality of the demonstrations is already embedded in the initial $\mathcal{F}_\theta$, we do not need to add them as additional constraints in this modified problem, improving the computation time.

The same modification can be done an unsafe measurement κ_{unsafe}, except with a constraint $g(\kappa_{\text{unsafe}}, \theta + s \odot u) > 0$.

Ranged, exact measurements: This case can come up when given LiDAR scans of the environment obtained in execution. For this setting, we assume that we are given a finite set of states which are all sensed to be safe, or all sensed to be unsafe. In our examples, we obtain this finite set of points by discretizing the possibly continuous ranged LiDAR measurement using a grid of measurement locations. This is a simple extension of the previously discussed modification for a single observed

state; we simply have to add constraints corresponding to each state, and use this modified problem in Algorithm VII.1:

$$\begin{array}{ll} \underset{s}{\text{maximize}} & \left(\prod_i s_i\right)^{1/d} \\ \text{subject to} & g(\kappa^i_{\text{safe}}, \theta + s \odot u) \leq 0, \quad i = 1, \ldots, N_{\text{safe}} \\ & g(\kappa^i_{\text{unsafe}}, \theta + s \odot u) > 0, \quad i = 1, \ldots, N_{\text{unsafe}} \end{array}$$

Direct, uncertain measurements: In this case, suppose that we are given an initial set of consistent constraints $\mathcal{F}_\theta$ as well as a finite set of states which may be possibly unsafe (the same ideas extend to the case where a set of states may be possibly safe); that is, we are given $\mathfrak{C}_{\neg s}$, where we learn in execution that at least one element of $\mathfrak{C}_{\neg s}$ is unsafe: $\exists \mathfrak{C}^i_{\neg s} \in \mathcal{A}(\theta^*)$. In our examples for the 7-DOF arm, we obtain this finite set of points by discretizing the continuous set of points which could be in contact by sampling points on the surface of the arm on the links downstream from where a torque limit is violated. Again, a similar modification can be made to Problem VII.4:

$$\begin{array}{ll} \underset{s}{\text{maximize}} & \left(\prod_i s_i\right)^{1/d} \\ \text{subject to} & \bigvee_{i=1}^{|\mathfrak{C}_{\neg s}|} g(\mathfrak{C}^i_{\neg s}, \theta + s \odot u) \leq 0 \end{array}$$

Specifically, the logical constraints over which state is unsafe can be modeled with binary variables, so the overall problem is still an MISOCP. The modified problem can be used in Algorithm VII.1.

B.5 Appendix: Chapter VII: Theory

In this appendix, we provide proofs for the theorems in the main body of the chapter.

Theorem B.5. *If Alg. VII.1 terminates for any parameterization, its output is guaranteed to cover $\mathcal{F}_\theta$.*

Proof. Suppose for contradiction that Algorithm VII.1 terminates such that $\mathcal{F}_\theta \setminus (\bigcup_{i=1}^{N_{\text{infeas}}} \hat{\mathcal{F}}^i_\theta) = \mathcal{F}^{\text{remain}}_\theta \neq \emptyset$. However, by construction, Alg. VII.1 only terminates if there does not exist any $\theta \in \Theta$ for which $\{\text{KKT}(\xi^{\text{dem}}_j)\}_{j=1}^{N_{\text{dem}}}$ can be satisfied; otherwise, Prob. VII.4 remains feasible. For all $\theta \in \mathcal{F}^{\text{remain}}_\theta$, by definition of being an element of $\mathcal{F}_\theta$, there exist $\mathcal{L}$ to satisfy $\{\text{KKT}(\xi^{\text{dem}}_j)\}_{j=1}^{N_{\text{dem}}}$. Contradiction. □

Theorem B.6. *Alg. VII.1 is guaranteed to terminate in finite time for union-of-boxes parameterizations.*

Proof. From Theorem B.3, $\mathcal{F}_\theta$ can be described as a union of a finite number of axis-aligned rectangles: $\mathcal{F}_\theta = \bigcup_{i=1}^{N_{\text{box}}} \mathcal{B}_i\}$. Extend each hyperplane defining the boundary of a box $\mathcal{B}_i$ to infinity to obtain an irregular grid $\{\mathcal{G}_i\}_{i=1}^{N_{\text{grid}}}$ over Θ (see Figure B.2 for

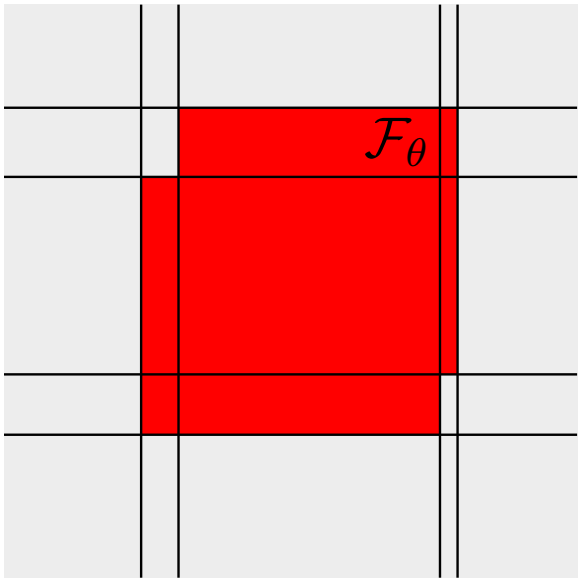


Figure B.2: Grid used in the proof of Theorem B.6.

the case in 2D). As $\mathcal{F}_\theta$ is composed of a finite number of boxes and hence there are a finite number of extended hyperplanes, there will be a finite number of grid cells, i.e. N_{grid} is finite.

We now prove that the solution of Problem VII.4 at any iteration i, $\hat{\mathcal{F}}_\theta^i$, can be exactly represented by some subset of grid cells: $\hat{\mathcal{F}}_\theta^i = \{\kappa \mid [I_{d\times d}, -I_{d\times d}]^\top \kappa \leq [\bar{\theta}, -\underline{\theta}]^\top\} = \bigcup_{j=1}^{N_{\text{rep}}} \mathcal{G}_j$. Suppose for contradiction that there exists some grid cell $\mathcal{G}_k$ that $\hat{\mathcal{F}}_\theta^i$ only partially contains: $(\mathcal{G}_k \cap \hat{\mathcal{F}}_\theta^i \neq \emptyset) \wedge (\mathcal{G}_k \cap \hat{\mathcal{F}}_\theta^i \neq \mathcal{G}_k)$. Formally, this means that in some coordinate of θ, say the mth coordinate, the upper bound of $\hat{\mathcal{F}}_\theta^i$, $\bar{\theta}_m^i$ satisfies $\bar{\theta}_m^i \in [\underline{\theta}_m^k, \bar{\theta}_m^k]$, where these denote the lower and upper bounds of grid k in dimension m; similar logic holds for analyzing the lower bound. For $\bar{\theta}_m^i$ to be the upper bound of $\hat{\mathcal{F}}_\theta^i$ in the mth coordinate, by the optimality of Problem VII.4, there must exist some constraint state κ contained in the expanded box $\{\kappa \mid [I_{d\times d}, -I_{d\times d}]^\top \kappa \leq [\bar{\theta}_1, \ldots, \bar{\theta}_{m-1}, \bar{\theta}_m^k, \bar{\theta}_{m+1}, \ldots, \bar{\theta}_d, -\underline{\theta}]^\top\}$ such that $\kappa \notin \mathcal{F}_\theta$. However, this is not possible, as by the grid partition, there exists no hyperplane defining $\mathcal{F}_\theta$ that can be crossed in the mth coordinate between $\underline{\theta}_m^k$ and $\bar{\theta}_m^k$. Contradiction.

Finally, as each iteration in Alg. VII.1 removes a finite number of grid cells, Alg. VII.1 will terminate in a finite number of iterations. □

Theorem B.7. *A solution to Prob. VII.8 is a guaranteed feasible, possibly suboptimal solution to Prob. VII.7.*

Proof. Feasibility follows by construction of Problem VII.8, as constraint (7.11f) directly models the probability constraint (7.10d): $\int_{\mathcal{B}_i} d\theta = t_i^d$, so $\sum_i t_i^d = \sum_i \int_{\mathcal{B}_i} d\theta = \int_{\Theta_s} d\theta$ for $\Theta_s = \bigcup_{i=1}^{N_{\text{box}}} \mathcal{B}_i$, which is exactly $\Pr(\xi_{xu} \text{ safe})$ when integrated over Θ_s, for the uniform prior $b_{\text{dem}}(\theta)$.

Suboptimality arises from the optimal partition of probability possibly not being representable as a union of boxes: in general, there exists Θ_s under which $c_\Pi(\xi_{xu})$ is minimized, such that there does not exist N_{box} boxes where $\Theta_s = \bigcup_{i=1}^{N_{\text{box}}} \mathcal{B}_i$. □

B.6 Appendix: Chapter VII: Further experimental details

In this section, we provide additional details on our experimental results. We first discuss in detail the baseline algorithms that we compare to in the results (Appendix B.6.1). We then demonstrate closed-loop planning with an uncertain nonlinear constraint using a sampled approximation of Problem VII.8 (Appendix B.6.2), and then discuss additional details and visualize example runs of our method and baseline approaches for the mixed quadrotor uncertainty example (Appendix B.6.3), the 7-DOF arm example (Appendix B.6.4), and the quadrotor maze example (Appendix B.6.5).

B.6.1 Planning baselines

B.6.1.1 Mixed quadrotor example

Scenario approach: The scenario approach *Grammatico et al.* (2016) satisfies uncertain constraints by sampling N_{sam} possible constraint parameters $\{\theta_i\}_{i=1}^{N_{\text{sam}}}$ and finds a solution that satisfies all of the sampled constraints. For this example, we may not be able to satisfy all of the sampled constraints, and hence the scenario approach may render the problem infeasible. To get around this to use the method as a baseline, we iteratively sample additional constraints, and solve the following open-loop planning problem:

$$\begin{aligned} \min_{\xi_{xu}} \quad & c_\Pi(\xi_{xu}) \\ \text{s.t.} \quad & \bar{\phi}(\xi_{xu}) \in \bar{\mathcal{S}} \subseteq \bar{\mathcal{C}},\ \phi_\Pi(\xi_{xu}) \in \mathcal{S}_\Pi \subseteq \mathcal{C}_\Pi \\ & \xi_{xu} \in \mathcal{S}(\theta),\ \forall \theta \in \{\theta_i\}_{i=1}^{N_{\text{sam}}} \end{aligned}$$

We stop sampling constraints when the problem becomes infeasible and return the feasible trajectory generated at the previous iteration.

Optimistic approach: In this approach, we are optimistic about the true constraint, only avoiding the set of guaranteed-unsafe states, buffering the extents by 0.5 in the uncertain constraint dimensions.

B.6.1.2 7-DOF arm example

BTP without constraint parameterization or demonstrations: In this version of BTP, we provide neither a union-of-boxes constraint parameterization nor a set of demonstrations. Instead, collision probabilities are measured with "Collision Hypothesis Sets" (CHS) *Saund and Berenson* (2018), which use a voxelization of the environment, with probabilities of particular voxels being occupied updated based on the occupancy of the robot volume during collision.

Optimistic approach: In this approach, we use the same graph provided to BTP and do not provide the information provided by the demonstrations, and we iteratively solve an optimistic problem. At the first iteration, we find run A* with all edges on the graph assumed valid, and attempt to execute the path. If the robot

collides when traversing some edge e_{uv} from vertex u to v, the robot backtracks to vertex u, removes edge e_{uv} from the graph, and replans on the modified graph. The procedure continues until the robot is at the goal.

B.6.1.3 Quadrotor maze example

Guaranteed-safe planning: In this approach *Chou et al.* (2020b), we compute paths which are guaranteed-safe with respect to the constraint uncertainty. Under the assumption that the constraint parameterization is correct, this guarantees that we will never need to replan upon discovering a constraint, but at the cost of possibly high-cost, conservative trajectories.

Optimistic approach: This approach *Janson et al.* (2018) is optimistic with respect to the uncertain space, and constructs high-level plans that plan to intermediate goals on the frontier of unknown space, between the current state and the goal, and executes the best high-level plan. In more detail, we replicate the approximations used in Section IV.B of *Janson et al.* (2018). In particular, we assume that the unknown space is free, but buffer known obstacles by 0.1 meters in the uncertain dimensions. We discretize the unknown frontier in 2 meter intervals to construct our subgoals.

B.6.2 Nonlinear constraint

We show that our method can plan with constraint beliefs for non-union-of-boxes constraint parameterizations. Specifically, we are given a demonstration on a 2D kinematic system $[\chi_{t+1}, y_{t+1}]^\top = [\chi_t, y_t]^\top + [u_t^\chi, u_t^y]^\top$ which minimizes path length $c(\xi) = \sum_{t=1}^{T-1} \|x_{t+1} - x_t\|^2$ (Figure B.3.A) while satisfying the constraint $g(x, \theta) = \theta_1(x_1^4 + x_2^4) + \theta_2(x_1^3 + x_2^3) + \theta_3(x_1 - 1)^3 + \theta_4(x_2 + 1)^3 > 2$ for $\theta = [2, -5, 5, 5]^\top$. As the demonstration is rather uninformative, many θ make it locally-optimal. As the constraint is affine in θ, we extract $\mathcal{F}_\theta$ with zonotopes (running Algorithm VII.1 with Problem B.4); see Figure B.3.B-C for $\text{proj}_\mathcal{X}(\mathcal{F}_\theta)$ and a corresponding probability heatmap. We now want to solve Prob. MCV, planning from $x_0 = [0, 0.75]^\top$ to $x_g = [0, -1.5]^\top$. As $g(x, \theta)$ is not MICP-representable, we solve an approximation of Prob. MCV with samples, sampling 100 constraints from $b_{\text{dem}}(\theta)$ as input to MCR (cf. Section 7.4.2). The resulting path, Plan 1, (Figure B.3.C) violates one possible constraint at $x_{\neg s} = [0.49, 0.8]^\top$. Though this path is safe for the true constraint, this is unknown to the learner, so we also precompute a contingency. Updating the belief $b_{\text{ex}}(\theta)$ with $\mathfrak{C}_{\neg s} = \{x_{\neg s}\}$ and previously visited states as $\mathfrak{C}_s$ reduces the constraint uncertainty (Figure B.3.D-E). In particular, we note that the measurement at $[0.49, 0.8]^\top$ also removes the constraint uncertainty on the left-hand side of the space; this is due to the nonlinearity of the constraint parameterization, and as we can see in the belief (Figure B.3.C), only some of the the convex obstacles that cover the right side of the space can explain the possible collision at $[0.49, 0.8]^\top$. Hence, this measurement will update the belief to eliminate the non-convex obstacles, removing the left-side uncertainty. For this updated belief, Contingency 1 can be planned with no constraint violations (Figure B.3.E). Extracting $\mathcal{F}_\theta$ with Algorithm VII.1 and

planning with MCR takes 30 min. (can be sped up with a parallel implementation, cf. Appendix B.3) and 30 sec., respectively.

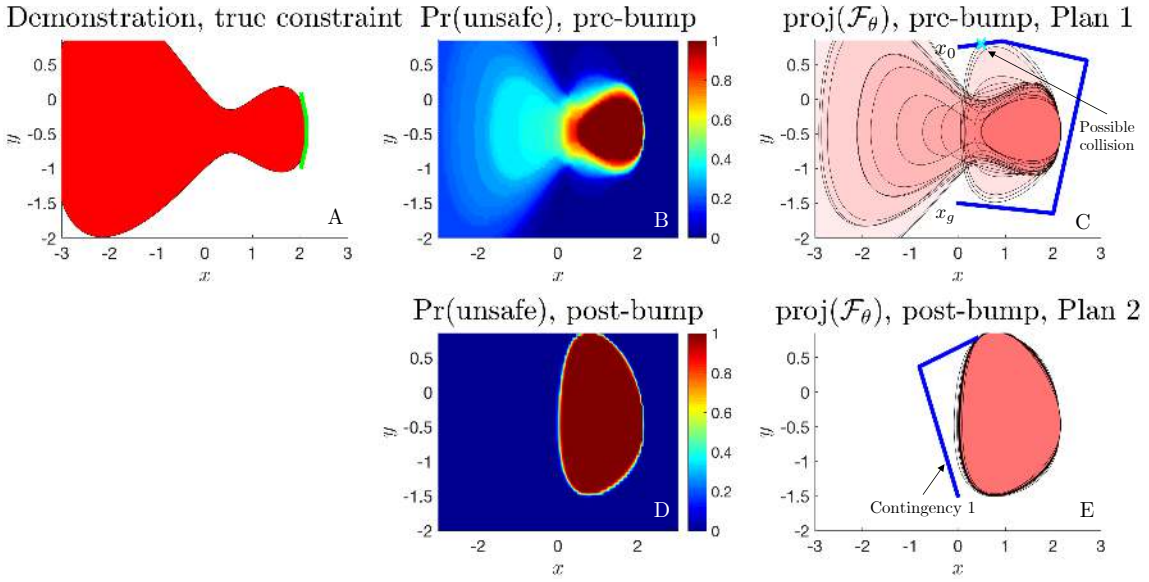


Figure B.3: Example demonstrating MCR on planning with a nonlinear constraint. **A**: Demonstration, overlaid with the true constraint (red). **B**: Initial constraint uncertainty, visualized as a normalized probability heatmap.**C**. The initial plan (blue) generated by MCR, overlaid by a subsampling of 20 of the sampled constraint parameters θ provided to MCR. A possible collision occurs at the cyan "x". **D**. The updated constraint uncertainty probability heatmap were the cyan state to be in collision. **E**. The new plan for the updated constraint uncertainty reaches the goal without violating any possible sampled constraints.

B.6.3 Mixed state-control constraint uncertainty on a quadrotor

The system dynamics for the quadrotor *Sabatino* (2015) are:

$$\begin{bmatrix} \dot{\chi} \\ \dot{y} \\ \dot{z} \\ \dot{\alpha} \\ \dot{\beta} \\ \dot{\gamma} \\ \ddot{\chi} \\ \ddot{y} \\ \ddot{z} \\ \ddot{\alpha} \\ \ddot{\beta} \\ \ddot{\gamma} \end{bmatrix} = \begin{bmatrix} \dot{\chi} \\ \dot{y} \\ \dot{z} \\ \dot{\beta}\frac{\sin(\gamma)}{\cos(\beta)} + \dot{\gamma}\frac{\cos(\gamma)}{\cos(\beta)} \\ \beta\cos(\gamma) - \dot{\gamma}\sin(\gamma) \\ \dot{\alpha} + \dot{\beta}\sin(\gamma)\tan(\beta) + \dot{\gamma}\cos(\gamma)\tan(\beta) \\ -\frac{1}{m}[\sin(\gamma)\sin(\alpha) + \cos(\gamma)\cos(\alpha)\sin(\beta)]u_1 \\ -\frac{1}{m}[\cos(\alpha)\sin(\gamma) - \cos(\gamma)\sin(\alpha)\sin(\beta)]u_1 \\ g - \frac{1}{m}[\cos(\gamma)\cos(\beta)]u_1 \\ \frac{I_y - I_z}{I_x}\dot{\beta}\dot{\gamma} + \frac{1}{I_x}u_2 \\ \frac{I_z - I_x}{I_y}\dot{\alpha}\dot{\gamma} + \frac{1}{I_y}u_3 \\ \frac{I_x - I_y}{I_z}\dot{\alpha}\dot{\beta} + \frac{1}{I_z}u_4 \end{bmatrix}, \tag{B.13}$$

We time-discretize the dynamics by performing forward Euler integration with discretization time $\delta t = 0.7$. The 12D state is $x = [\chi, y, z, \alpha, \beta, \gamma, \dot{x}, \dot{y}, \dot{z}, \dot{\alpha}, \dot{\beta}, \dot{\gamma}]^\top$, and the relevant constants are $g = -9.81\text{m/s}^2$, $m = 1\text{kg}$, $I_x = 0.5\text{kg} \cdot \text{m}^2$, $I_y = 0.1\text{kg} \cdot \text{m}^2$, and $I_z = 0.3\text{kg} \cdot \text{m}^2$.

The simplified double integrator model that we use to plan in Problem VII.8 is as follows:

$$\begin{bmatrix} \dot{\chi} \\ \dot{y} \\ \dot{z} \\ \ddot{x} \\ \ddot{y} \\ \ddot{z} \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} \chi \\ y \\ z \\ \dot{x} \\ \dot{y} \\ \dot{z} \end{bmatrix} + \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \\ u_3 \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ g \end{bmatrix} \tag{B.14}$$

which is then time discretized with $\delta t = 0.5$, with $g = -9.81\text{m/}s^2$.

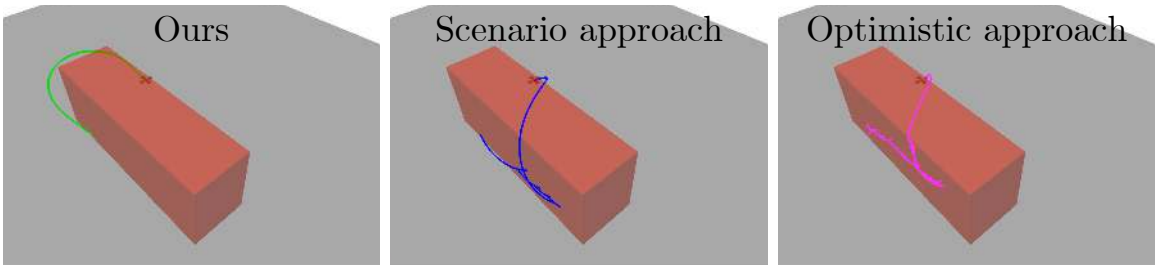


Figure B.4: Example run 1 (mixed quadrotor example).

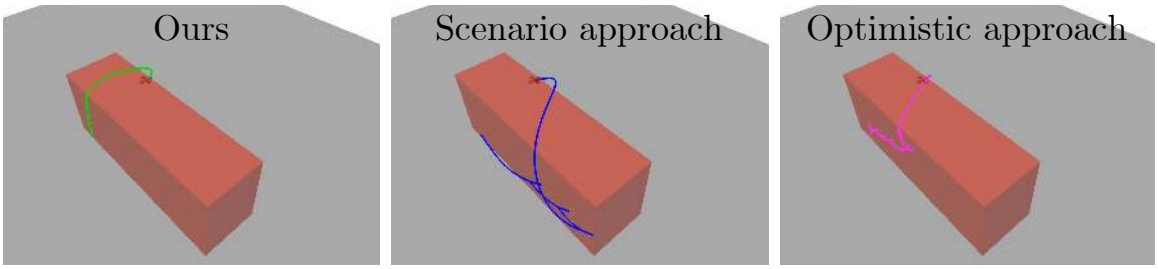


Figure B.5: Example run 2 (mixed quadrotor example).

Visualizing example runs: For two different sampled ground-truth constraints, we visualize the trajectories executed by our policy, the scenario approach policy *Grammatico et al.* (2016), and the optimistic policy to compare their properties.

In Figure B.4, we display the sampled state constraint in red, and the sampled control constraint is $\|u\|_2^2 \leq 98.27$. Our policy, which attempts to satisfy all of the state constraints by trying to move above all of the possible obstacles (see Sec. 7.5 for more discussion), violates the control constraint at the first time-step by attempting to do so. Our policy then switches to the first contingency plan and successfully reaches the goal with 1 constraint violation. On the other hand, the scenario approach suffers 6 constraint violations (due to sampling constraints), while the optimistic approach

suffers 25 constraint violations (because it does not try to avoid any states other than those which are known to be unsafe).

In Figure B.5, we display the sampled state constraint in red, and the sampled control constraint is $\|u\|_2^2 \leq 98.55$. Our policy reaches the goal in one try (0 constraint violations), as we do not end up violating the control constraint. On the other hand, the scenario approach suffers 4 constraint violations, while the optimistic approach suffers 15 constraint violations.

Histogram: In computing Plan 1 (the initial executed trajectory) and Contingencies 1-4 (the trajectories we switched to upon observing 1, 2, 3, and 4 constraint violations), we can calculate the volumes of the covered $\mathcal{B}_i$ which are optimized by Problem VII.8 at each iteration. When computing Plan 1, we cover $p_0 = 69.10\%$ of the possible constraints; when computing Contingency 1, we cover $p_1 = 54.90\%$ of the possible constraints under the updated belief, $p_2 = 56.33\%$ for Contingency 2, $p_3 = 73.91\%$ for Contingency 3, and $p_4 = 98.00\%$ for Contingency 4. Using these percentages, we can calculate the theoretical frequency of overrides as $p(0 \text{ overrides}) = p_0$, $p(1 \text{ override}) = (1 - p_0)p_1$, and so on, until $p(4 \text{ overrides}) = \prod_{i=1}^{3}(1 - p_i)p_4$. Using these formulae, we compute the theoretical probability of suffering i constraint violations before reaching the goal, which we compare with the empirical histograms (normalized over 500000 trials) (Fig. B.6), and we see the statistics match quite closely.

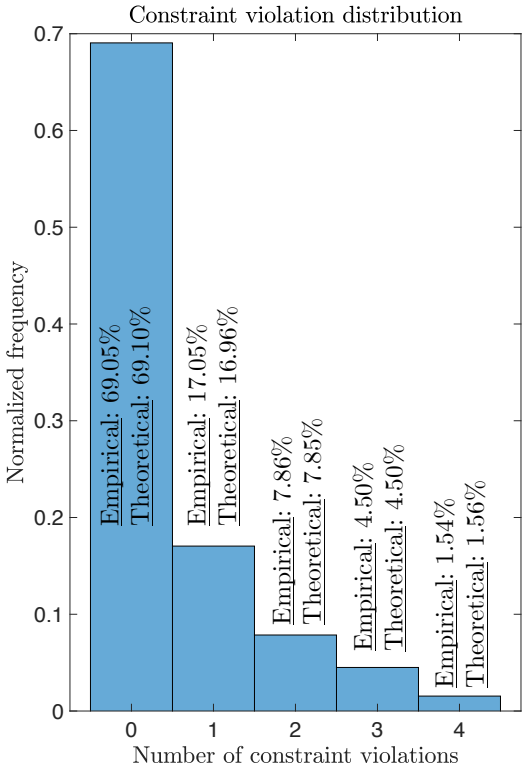


Figure B.6: Constraint violation histogram for the mixed quadrotor uncertainty example.

B.6.4 7-DOF arm with contact sensing uncertainty

We use a kinematic model of the arm: $x^i_{t+1} = x^i_t + u^i_t$, for $i = 1, \ldots, 7$. Here, $x \in \mathbb{R}^7$, where coordinate i of the state denotes the angle of the ith joint. For planning, we use a BTP graph with 5000 vertices (cf. Section 7.4.2). In the following, we discuss more details on the performance of different policies on the task discussed in Section 7.5.

BTP with CHS: We present a time-lapse of the trajectory executed by running BTP with CHS (as described in Appendix B.6.1.2) in Figure B.7. Since this approach is not given demonstrations, it requires several bumps in order to sufficiently localize the shelf. Furthermore, this approach does not leverage a union-of-boxes constraint parameterization as a prior on the world, as the CHS does not extrapolate about the constraint beyond the sensed collision, making it so that more bumps occur before reaching the goal.

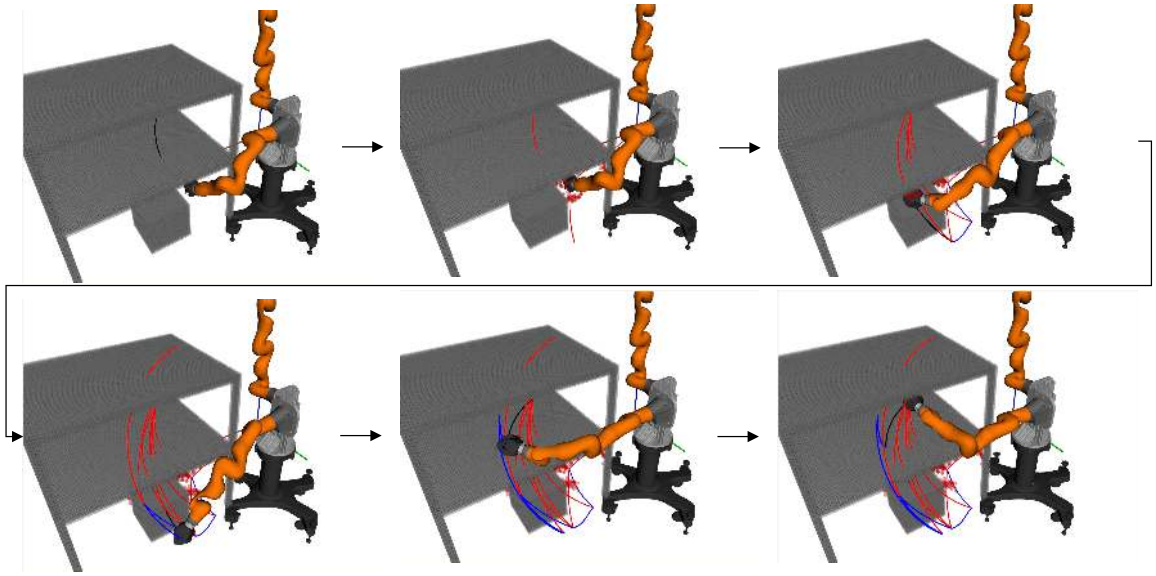


Figure B.7: BTP with CHS. Voxels are colored red if they are possibly unsafe according to the CHS. Red edges are attempted edges which are discovered to be blocked in execution. The attempted edges which were unblocked are colored blue.

Optimistic approach: We present the trajectory executed by running the optimistic strategy described in Appendix B.6.1.2 in Figure B.8. Since this strategy entirely ignores the correlation in validity between edges which are close to each other, it explores many edges, yielding a trajectory cost which is much higher than the other approaches.

Suboptimal human demonstrations: Finally, we present a time-lapse of our policy when initialized with suboptimal human demonstrations in Figure B.10. The demonstrations are captured using an HTC Vive in a virtual reality simulation environment (Figure B.9). Overall, the behavior of our policy when initialized with these human demonstrations is similar to the the case of synthetic demonstrations (which is discussed in Sec. 7.5): it plans to move around the shelf, and in doing so, collides

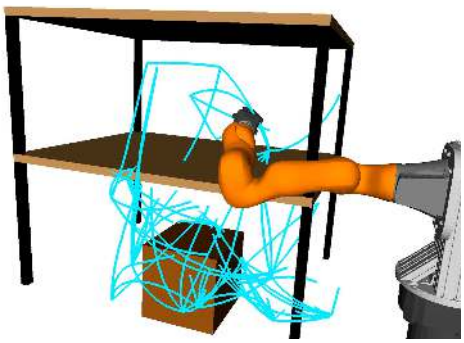


Figure B.8: Trajectory executed by the optimistic policy.

with the unmodeled obstacle. This triggers a constraint parameterization update, and the replanned path (which avoids the shelf and the uncertain region induced by the collision) steers the arm to the goal without further collision. For this case, the executed trajectory cost is 7.78 rad.

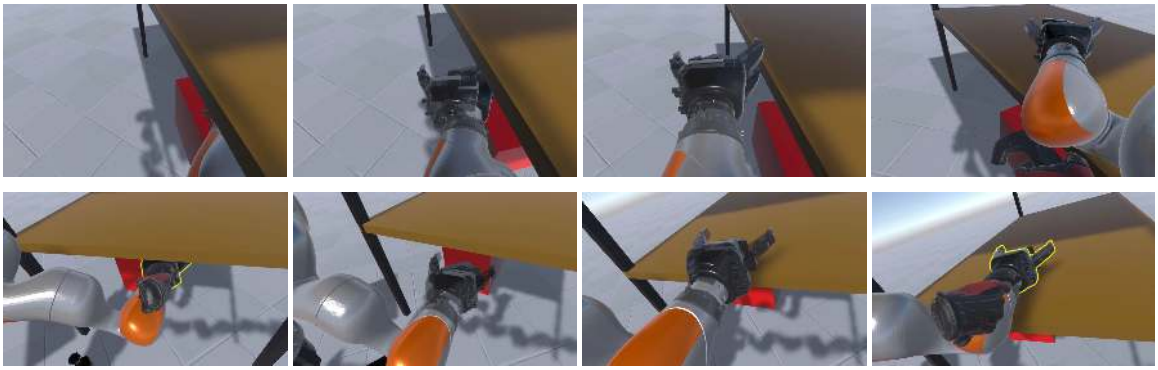


Figure B.9: Suboptimal human demonstrations. Top row: time-lapse of the first demonstration. Bottom row: time-lapse of the second demonstration.

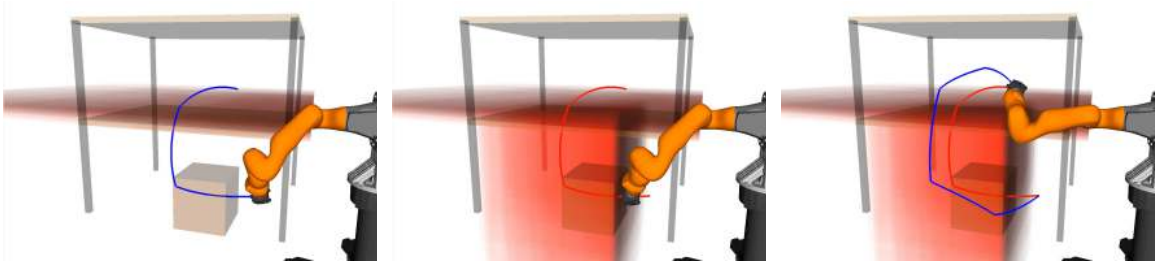


Figure B.10: Policy when initialized with suboptimal demonstrations. Left: plans an initial trajectory that bumps into the unmodeled obstacle. Center: constraint parameterization is updated to two boxes. Right: replanned trajectory successfully avoids all collisions, steering the arm to the goal.

B.6.5 Quadrotor maze

Inevitable collision states: We integrate the inevitable collision state *Fraichard and Asama* (2004) avoidance constraints into our approach. We first enforce that on a planned trajectory, each state which may be possibly unsafe must be observable (within 2 meters) from a previous state on the trajectory. Furthermore, the line of sight between this previous state and the possible unsafe state cannot be occluded by any obstacle other than the obstacle which is making the state possibly unsafe. For these possibly unsafe states, we enforce that we can brake in time to avoid collision (and together with the line-of-sight constraint, enforces that we can also sense if we need to brake). This is done by explicitly optimizing "brake trajectories" in conjunction with the planned trajectory, which are rooted two time-steps on the plan before a possible collision and which bring the system to a stop without violating any possible constraints. We further enforce the line of sight constraint by discretizing the line segment between x_t and x_{t+2} (if x_{t+2} is possibly unsafe) into 10 points, and enforcing that each discretized state is guaranteed to satisfy all constraints other than the uncertain constraint which can make x_{t+2} possibly unsafe.

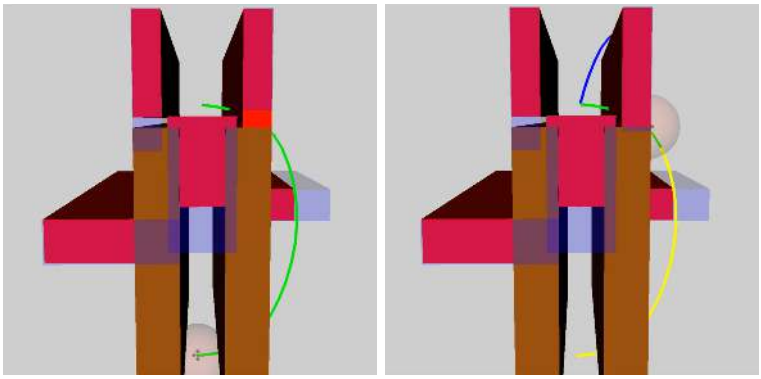


Figure B.11: Example run, our policy (quadrotor maze). Initial plan (green), contingency plan (blue), actually executed plan (yellow). The sphere around the quadrotor indicates the sensing radius.

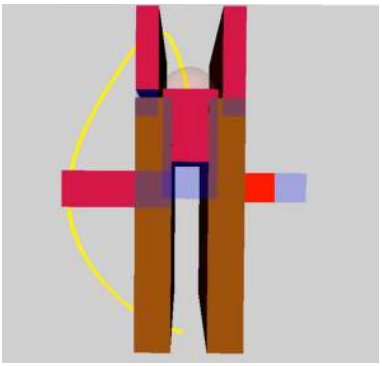


Figure B.12: Example run, guaranteed-safe policy (quadrotor maze). Initial/actually executed plan (yellow).

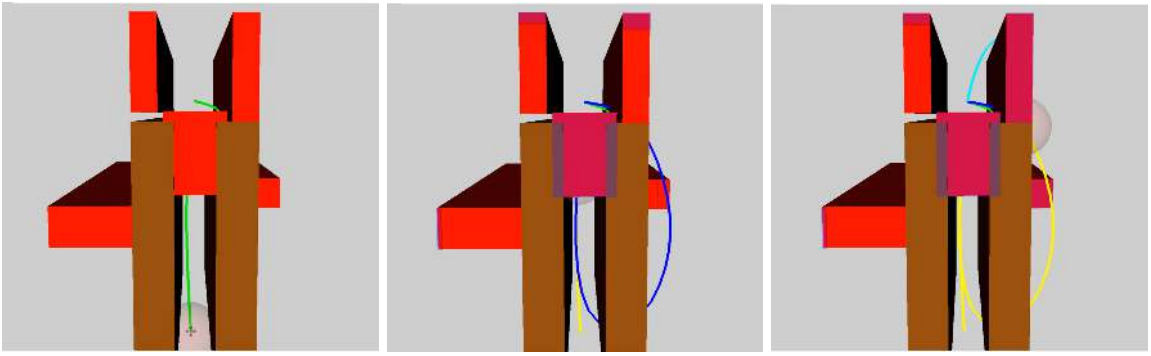


Figure B.13: Example run, optimistic policy (quadrotor maze). Initial plan (green), contingency plan 1 (blue), contingency plan 2 (cyan), actually executed plan (yellow).

Visualizing an example run: For one sampled possible environment (displayed in red in Figures B.11-B.13), we visualize the trajectories executed by our policy, a policy which seeks to plan guaranteed-safe trajectories *Chou et al.* (2020b), and an optimistic policy *Janson et al.* (2018). Our policy moves to the right and seeks to cut through the possibly unsafe region in the top right (Figure B.11); when observing that the region is blocked, our policy switches to the blue contingency trajectory. On the other hand, the guaranteed-safe policy (Figure B.12) seeks to avoid all possible constraints; as a result, while this policy never needs to switch to a contingency plan, it also ends up deterministically executing a higher-cost trajectory. Finally, the optimistic policy in *Janson et al.* (2018) explores the dead end between the brown obstacles and is forced to backtrack, yielding a higher cost compared to our policy.

B.6.6 Computation times

One challenge that our method faces when applied to real-time replanning is the computational intensity of online belief updates and online replanning of open-loop trajectories. First, we emphasize that if the assumptions in Section 7.4.4 hold, we can precompute the possible belief updates and contingency plans to avoid computing them online. If the assumptions are not satisfied, we will need to perform the computation online.

For belief updates, the computation time and number of measurements that can be updated depends heavily on the measurement type. For instance, updating the belief on the quadrotor maze example for a LiDAR scan with 30000 discretized points takes 1.4 seconds; LiDAR-type measurements are fast as each point is known safe or unsafe. However, contact measurements (as seen in the 7-DOF arm examples) are expensive as an unknown combination of the discretized points can be unsafe; modeling this requires the addition of many binary decision variables; it takes 30 minutes to incorporate a contact measurement with 300 discretized points. In this case, a further investigation of the tradeoff between accuracy and computation time based on the number of sampled possible contact measurements may lead to further computational gains.

The integer optimization variables are the key reason for slow $\mathcal{F}_\theta$ extraction in

Algorithm VII.1. Thus, we are optimistic that we can speed up computation with parallelization (see Appendix B.3.3) and recent advances in fast mixed integer programming *Bertsimas and Stellato* (2019), which enjoy orders of magnitude speedup by learning efficient branching heuristics.

For open-loop planning, we note that the aforementioned fast mixed integer programming methods, as well as other work in warm-starting mixed integer programs, can be useful in reducing planning times for solving Problem VII.8 and other variants, as all of these variants are mixed integer programs, and the previous open-loop plan can serve as a good initalization for replanning. We also emphasize that we can reduce BTP planning times for the 7-DOF arm examples to around 15 seconds (as in the original BTP paper *Saund et al.* (2019)) by precomputing arm swept volumes along roadmap edges and by employing lazy collision checking.

APPENDIX C

Appendix for Chapter 10: Safe Output Feedback Motion Planning from Images via Learned Perception Modules and Contraction Theory

C.1 Appendix: Chapter X: Trusted domain visualizations

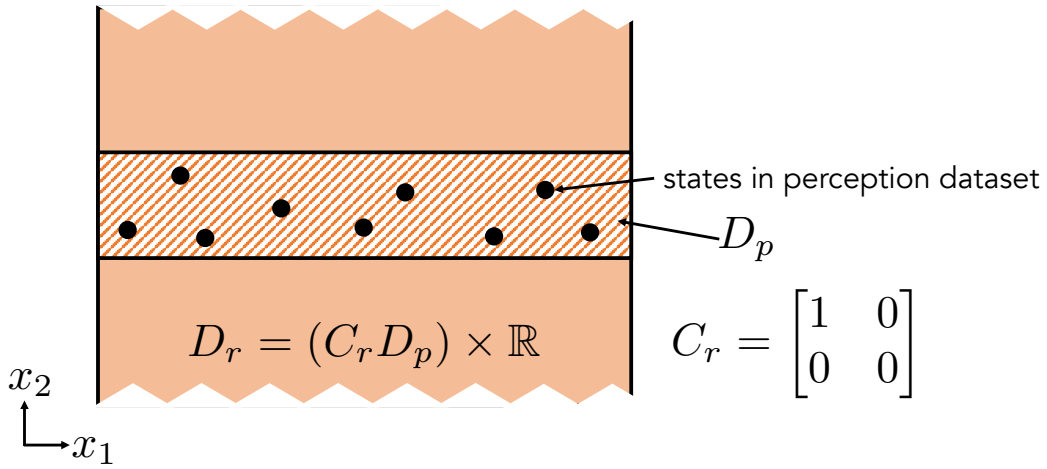


Figure C.1: An example of how $D_r \subseteq \mathcal{X}$ is constructed.

In Fig. C.1, we visualize for a toy example how we construct D_r, which is a critical set upon which we define the trusted domain D_x.

C.2 Appendix: Chapter X: Bounding estimation error (expanded)

How can we bound the learned perception module error $\epsilon(x, \theta) \doteq \|\hat{h}^{-1}\big(h(x, \theta), \theta\big) - C_r x\|$ over $D_r \times D_\theta$? We describe three options, each with their own strengths/drawbacks.

The first and simplest option is to estimate a *uniform bound* $\bar{\epsilon}_1$ on the error:

$$\epsilon(x,\theta) \leq \bar{\epsilon}_1, \quad \forall (x,\theta) \in D_r \times D_\theta. \tag{C.1}$$

This works well if the error is consistent across $D_r \times D_\theta$; however, it will be conservative if there are any error spikes. A second option is to derive a *spatially-varying bound* on $\epsilon(x,\theta)$. To do so, we first estimate the Lipschitz constant of ϵ, L_p, over $D_r \times D_\theta$:

$$\|\epsilon(\tilde{x},\tilde{\theta}) - \epsilon(\check{x},\check{\theta})\| \leq L_p \|(\tilde{x},\tilde{\theta}) - (\check{x},\check{\theta})\|, \quad \forall (\tilde{x},\tilde{\theta}),(\check{x},\check{\theta}) \in D_r \times D_\theta. \tag{C.2}$$

Denote $\mathcal{S}_{x\theta} = \{(x_i,\theta_i)\}_{i=1}^{N_{\text{data}}}$. We can then write this bound $\bar{\epsilon}_2(x^*,\theta)$ (cf. Fig. 10.4.B for visuals), which crucially is an explicit function of the plan x^*, and can thus directly guide our planner Alg. X.1:

Theorem C.1 ($\bar{\epsilon}_2(x^*,\theta)$)**.** *: Recall that $d_c(t) = d_c(x^*(t), x(t))$ is the Riemannian distance between the nominal and true state at some time t. An upper bound on the perception error $\epsilon(x,\theta)$ that scales linearly with $d_c(t)$ can be written as:*

$$\epsilon(x,\theta) \leq L_p d_c / \sqrt{\underline{\lambda}_{D_c}(M_c)} + \min_{1\leq i\leq N_{data}}\{L_p(\|(x^*,\theta) - (x_i,\theta_i)\|) + \epsilon_i\} \doteq \bar{\epsilon}_2(x^*,\theta) \tag{C.3}$$

Proof. For some training point $(x_i,\theta_i) \in (\mathcal{S}_{x\theta})$, we can calculate $\epsilon_i \doteq \|\hat{h}^{-1}(h(x_i,\theta_i),\theta_i) - C_r x_i\|$ as its training error. Using ϵ_i and L_p, we can bound the error at a query $(x,\theta) \in D_r \times D_\theta$, as $\epsilon(x,\theta) \leq L_p\|(x,\theta) - (x_i,\theta_i)\| + \epsilon_i$. This holds for all data, so we can tighten the bound by taking the pointwise minimum of the bounds over all datapoints:

$$\epsilon(x,\theta) \leq \min_{1\leq i\leq N_{\text{data}}}\{L_p\|(x,\theta) - (x_i,\theta_i)\| + \epsilon_i\} \tag{C.4}$$

As written, (C.4) is only implicitly a function of the plan, via the relationship between x and x^*, and cannot directly guide planning, since x is unknown at planning time. We can make this bound an explicit function of the plan x^* by rewriting it as follows:

$$\begin{aligned} \epsilon(x,\theta) &\leq \min_{1\leq i\leq N_{\text{data}}}\{L_p\|(x,\theta) - (x_i,\theta_i)\| + \epsilon_i\} \\ &\leq \min_{1\leq i\leq N_{\text{data}}}\{L_p(\|(x,\theta) - (x^*,\theta)\| + \|(x^*,\theta) - (x_i,\theta_i)\|) + \epsilon_i\} \\ &\leq \tfrac{L_p d_c}{\sqrt{\underline{\lambda}_{D_c}(M_c)}} + \min_{1\leq i\leq N_{\text{data}}}\{L_p(\|(x^*,\theta) - (x_i,\theta_i)\|) + \epsilon_i\} \doteq \bar{\epsilon}_2(x^*,\theta) \end{aligned} \tag{C.5}$$

The second inequality follows from the triangle inequality, and the third inequality by applying $\sqrt{\underline{\lambda}_{D_c}(M_c)}\|x_1 - x_2\| \leq d_c(x_1,x_2) \leq \sqrt{\bar{\lambda}_{D_c}(M_c)}\|x_1 - x_2\|$. □

Instead of bounding the error over the whole domain (as in $\bar{\epsilon}_1$), $\bar{\epsilon}_2(x^*,\theta)$ is tighter as it only bounds the error over the tube, $\Omega_c(t)$. However, due to the leading term in (C.3), $\bar{\epsilon}_2(x^*,\theta)$ scales linearly with d_c, even if the true error $\epsilon(x,\theta)$ is relatively constant, making it loose for large d_c. Thus, we derive a third error bound to mitigate this, $\bar{\epsilon}_3(x^*,\theta)$ (cf. Fig. 10.4.C). Overall, this third error bound $\bar{\epsilon}_3(x^*,\theta)$ is useful for large d_c, as there is no direct scaling with d_c; however, it can be conservative if the data has high dispersion. We describe this in more detail in the following, after some definitions and proving a property of the dispersion. We provide a more detailed visualization of our bound in Fig. C.2.

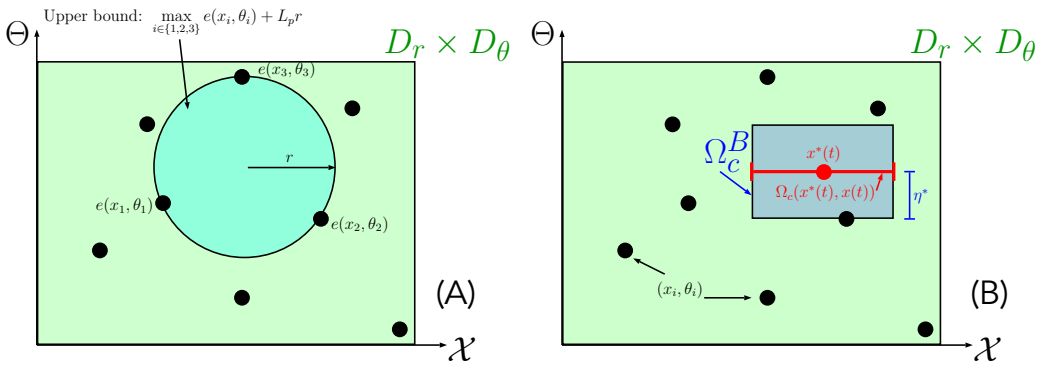


Figure C.2: (A) Visualization of the dispersion; together with the Lipschitz constant, it can bound the error within the blue set. (B) A visualization of our set construction in $\bar{\epsilon}_3(x^*, \theta)$.

Define $\mathcal{B}_r(x) = \{\tilde{x} \mid \|\tilde{x} - x\| < r\}$. and the dispersion of a finite set A contained in a set B, $\mathcal{R}(A, B)$:

$$\begin{aligned} \mathcal{R}(A,B) \doteq \quad & \sup_{r\geq 0, x\in B} \quad r \\ & \text{s.t.} \quad \mathcal{B}_r(x) \cap A = \emptyset, \quad \mathcal{B}_r(x) \subseteq B \end{aligned} \tag{C.6}$$

i.e., the radius of the largest open ball inside B that does not intersect with A. The following property of the dispersion will be useful to us in deriving $\bar{\epsilon}_3(x^*, \theta)$.

Lemma C.2 (Dispersion of a subset)**.** *Consider a finite set X, a compact set Y, where $X \subseteq Y$, and a compact subset $Z \subseteq Y$. Then, $\mathcal{R}(X \cap Z, Y \cap Z) \leq \mathcal{R}(X, Y)$.*

Proof. Consider a "sub-problem" of (C.6), which finds the largest open ball satisfying the constraints of (C.6) when the ball center is fixed to $x \in Z$:

$$\begin{aligned} \mathcal{R}_x(A,B) \doteq \quad & \sup_{r\geq 0} \quad r \\ & \text{s.t.} \quad \mathcal{B}_r(x) \cap B = \emptyset, \quad \mathcal{B}_r(x) \subseteq A \end{aligned} \tag{C.7}$$

Compare the value of $\mathcal{R}_x(X, Y)$ and $\mathcal{R}_x(X \cap Z, Y \cap Z)$ for any fixed $x \in Z$. Note that the feasible set of (C.7) when $A = X \cap Z$ and $B = Y \cap Z$ is contained within the feasible set of (C.7) when $A = X$ and $B = Y$: any ball which is contained in $Y \cap Z$ and does not intersect $X \cap Z$ is also contained in Y, and does not intersect X. Thus, $\mathcal{R}_x(X \cap Z, Y \cap Z) \leq \mathcal{R}_x(X, Z)$.

Now, consider the original problem (C.6), which can be rewritten as $\mathcal{R}(A, B) = \sup_{x\in B} \mathcal{R}_x(A, B)$. We have that $\mathcal{R}(X, Y) \geq \mathcal{R}(X \cap Z, Y \cap Z)$, since Y is a superset of $Y \cap Z$ and thus the feasible set when $B = Y$ is larger than when $B = Y \cap Z$.

□

As an abuse of notation, we refer to $\mathcal{R}$ without any arguments as shorthand for $\mathcal{R}(\mathcal{S}_{x\theta}, D_r \times D_\theta)$. Let $\eta^*(t) = \min_{\eta\geq 0, \mathcal{S}_{x\theta} \cap (\Omega_c(t) \times \mathcal{B}_\eta(\theta)) \neq \emptyset} \eta$ be the radius of the smallest ball around θ, $\mathcal{B}_{\eta^*(t)}(\theta)$, such that $(\Omega_c \times \mathcal{B}_{\eta^*(t)}(\theta)) \cap (D_r \times D_\theta) \doteq \Omega_c^B$ contains some datapoint. Then, we write our third error bound $\bar{\epsilon}_3(x^*, \theta)$:

Theorem C.3 ($\bar{\epsilon}_3(x^*, \theta)$)**.** *An upper bound on the perception error $\epsilon(x, \theta)$ can be written based on buffering the local training errors:*

$$\epsilon(x, \theta) \leq L_p \mathcal{R} + \max_{(x_i, \theta_i) \in \mathcal{S}_{x\theta} \cap \Omega_c^B} \epsilon(x_i, \theta_i) \doteq \bar{\epsilon}_3(x^*, \theta) \tag{C.8}$$

Proof. Finally, with an abuse of notation, denote $L_p(A)$ as the local Lipschitz constant of $\epsilon(x, \theta)$ in x and θ, valid for all $(x, \theta) \in A$.

$$\begin{aligned} \epsilon(x, \theta) &\leq \sup_{\tilde{x} \in \Omega_c} \epsilon(\tilde{x}, \theta) \leq \sup_{(\tilde{x}, \tilde{\theta}) \in \Omega_c^B} \epsilon(\tilde{x}, \tilde{\theta}) \\ &\leq L_p(\Omega_c^B) \mathcal{R}(\mathcal{S}_{x\theta} \cap \Omega_c^B, \Omega_c^B) + \max_{(x_i, \theta_i) \in \mathcal{S}_{x\theta} \cap \Omega_c^B} \epsilon(x_i, \theta_i) \\ &\leq L_p(D_r \times D_\theta) \mathcal{R}(\mathcal{S}_{x\theta}, D_r \times D_\theta) + \max_{(x_i, \theta_i) \in \mathcal{S}_{x\theta} \cap \Omega_c^B} \epsilon(x_i, \theta_i) \doteq \bar{\epsilon}_3(x^*, \theta), \end{aligned} \tag{C.9}$$

where the second inequality follows from the definition of dispersion and the fourth follows from a property of dispersion shown in Lem. C.2. □

Intuitively, $\bar{\epsilon}_3(x, \theta)$ uses the training errors e_i for points inside $\Omega_c(t)$ and buffers them with the Lipschitz constant of the error and the data density to upper bound the error inside $\Omega_c(t)$ (see Fig. 10.4). For this to make sense, there must be at least one datapoint in the considered set; however, in general, the external parameter $\theta \in D_\theta$ given as input to the OFMP may not be precisely in the dataset (as the dataset is just a finite sampling of D_θ). There will, however, be datapoints with θ_i close by θ. Thus, for the maxima and suprema terms to be well-defined, we must buffer $\Omega_c(t)$ in the θ coordinates until at least one datapoint lies in $\Omega_c(t) \times \mathcal{B}_\eta(\theta)$, for some buffer radius $\eta \geq 0$. In the proof, we make the relaxation in the last inequality so that we only have to estimate one Lipschitz constant and one dispersion, instead of needing to calculate them for each $\Omega_c(t)$; this simplifies the estimation of the constants discussed in Rem. X.4.

Each of these three bounds on $\epsilon(x, \theta)$ can be plugged into (10.17) to upper bound the integral in (10.9). As we use constant M_e and ρ in the results, this simplifies the integral to $\|R_e w_q(t)\|$; we prove a bound on $\|R_e w_q(t)\|$ in Lemma X.2.

C.3 Appendix: Chapter X: Proofs

In this appendix, we present the proofs of the theoretical results in the main body of the chapter; for convenience, we have copied the theorem statements here.

Lemma C.4 ($\dot{d}_c(t)$)**.** *The integral term in* (10.8) *can be bounded as*

$$\int_0^1 \|R_c(\gamma_c^t(s)) w_c(t)\| ds \leq \sqrt{\bar{\lambda}_{D_c}(M_c)} \bar{w}_x + L_{\Delta k} d_e. \tag{C.10}$$

Proof. Recall the form of w_c from (10.14). First consider the third term in (10.14) (the dynamics error). By using $\|w_x\| \leq \bar{w}_x$ and $\|R_c(\cdot)\| \leq \sqrt{\bar{\lambda}_{D_c}(M_c)}$, we can pull w_x and $R_c(\cdot)$ out of the integral to form the first term in the lemma statement. The second term follows by performing a max over geodesic parameters $s \in [0, 1]$ and substituting

the appropriate arguments into (10.15). Specifically, we have the following chain of relations:

$$\begin{aligned}
&\textstyle\int_0^1 \|R_c(\gamma_c^t(s))B(u(\hat{x},x^*,u^*)-u_{\text{closest}})\|ds \\
&\quad\leq \textstyle\int_0^1 \max_{s\in[0,1]} \|R_c(\gamma_c^t(s))B(u(\hat{x},x^*,u^*)-u_{\text{closest}})\|ds \\
&\quad= \textstyle\int_0^1 \Delta k(\hat{x},x,x^*,u^*)ds = \Delta k(\hat{x},x,x^*,u^*) \\
&\quad= |\Delta k(\hat{x},x,x^*,u^*)-\Delta k(x,x,x^*,u^*)| \leq L_{\Delta k} d_e(\hat{x},x) = L_{\Delta k} d_e
\end{aligned}$$

The first inequality holds via upper-bounding the norm over the geodesic parameters, the first equality holds by substituting the definition of Δk, the second equality holds as Δk is not a function of s, the third equality holds since $\Delta_k(x,x,x^*,u^*)=0$, and the final inequality holds from the definition of our Lipschitz constant.

□

Lemma C.5 ($\dot{d}_e(t)$)**.** *Let $\bar{\sigma}(B_y)$ denote the maximum singular value of B_y. For constant ρ and M_e, the integral in* (10.9) *simplifies to $\|R_e w_q(t)\|$ and can be bounded as:*

$$\|R_e w_q(t)\| \leq \sqrt{\bar{\lambda}(W_e)}\bar{w}_x + \tfrac{1}{2}\rho\bar{\lambda}(M_e)^{1/2}\big(L_{\hat{h}^{-1}}\sqrt{\bar{\sigma}(B_y)}\bar{w}_y + \bar{\epsilon}_{\{1,2,3\}}(x^*,\theta)\big) \tag{C.11}$$

Proof. The first term of the bound, $\sqrt{\bar{\lambda}(W_e)}\bar{w}_x$, follows the same logic from Lemma X.1, except this time $\|R_e(\cdot)\| \leq \sqrt{\bar{\lambda}(W_e)}$. The second term follows from first combining the final line of (10.17) with one of the three error bounds $\bar{\epsilon}_{\{1,2,3\}}$, and substituting that combination into the integrand of (10.9) (which here simplifies to $\|R_e w_q(t)\|$). Specifically, we have the following:

$$\begin{aligned}
\|R_e\tfrac{1}{2}\rho M_e C_r^\top(\hat{h}^{-1}(y,\theta)-C_r x)\| &= \tfrac{\rho}{2}\|R_e M_e C_r^\top(\hat{h}^{-1}(y,\theta)-C_r x)\| \\
&= \tfrac{\rho}{2}\|R_e^{-\top} C_r^\top(\hat{h}^{-1}(y,\theta)-C_r x)\| \\
&\leq \tfrac{\rho}{2}\|R_e^{-\top} C_r^\top\|\|\hat{h}^{-1}(y,\theta)-C_r x\| \\
&\leq \tfrac{\rho}{2}\bar{\lambda}(M_e)^{1/2}\Big(L_{\hat{h}^{-1}}\sqrt{\bar{\sigma}(B_y)}\bar{w}_y + \bar{\epsilon}_{\{1,2,3\}}(x,\theta)\Big)
\end{aligned}$$

The simplification in the second equality is done via properties of the Cholesky decomposition: i.e., $R_e M_e = R_e(R_e^\top R_e)^{-1} = R_e R_e^{-1} R_e^{-\top} = R_e^{-\top}$. The final inequality follows from $\|R_e^{-\top}\| \leq \sqrt{\bar{\lambda}(M_e)}$, $\|C_r\| \leq 1$, and applying the triangle inequality. □

Theorem C.6 (From derivative to value)**.** *Let RHS denote the right hand side of* (10.19)*. Given bounds on the Riemannian distances at $t=0$: $d_c(0) \leq \bar{d}_c(0)$ and $d_e(0) \leq \bar{d}_e(0)$, upper bounds $\bar{d}_c(t) \geq d_c(t)$ and $\bar{d}_e(t) \geq d_e(t)$ for all $t \in [0,T]$ can be written as*

$$\begin{bmatrix} d_c(t) \\ d_e(t) \end{bmatrix} \leq \int_{\tau=0}^{t} RHS\Big(\tau, \begin{bmatrix} d_c \\ d_e \end{bmatrix}\Big)d\tau \doteq \begin{bmatrix} \bar{d}_c(t) \\ \bar{d}_e(t) \end{bmatrix}, \quad d_c(0)=\bar{d}_c(0),\ d_e(0)=\bar{d}_e(0).$$

Proof. Using a vector-valued comparison theorem (*Lakshiliikantham and Leela*, 1969, Corollary 1.7.1), we have that $d_c(t)$ and $d_e(t)$ can be upper bounded on $[0,T]$ by the solution to the upper bound of (10.19), i.e., $[\dot{d}_c, \dot{d}_e]^\top = \text{RHS}$, provided that the solution

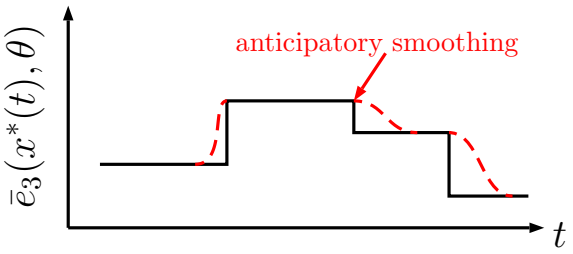


Figure C.3: An example of anticipatory smoothing: the red curve is a continuous upper bound to the potentially discontinuous $\bar{\epsilon}_3(x^*, \theta)$.

to RHS exists until at least $t = T$. Further prerequisites for applying (*Lakshiliikantham and Leela*, 1969, Corollary 1.7.1) are that RHS is quasi-monotone nondecreasing in $[d_c, d_e]^\top$ and that RHS is continuous in t and $[d_c, d_e]^\top$. In the following, we show that these requirements hold when using $\bar{\epsilon}_1$, $\bar{\epsilon}_2$, and a modified, continuous version of $\bar{\epsilon}_3$.

First, we describe this modification to $\bar{\epsilon}_3(x^*(t), \theta)$, which we note is not continuous in t, in general. This is because as t changes, the max term in (C.9) can discontinuously change, based on the datapoints that fall inside $\Omega_c(t)$. However, we can obtain a smooth upper bound to $\bar{\epsilon}_3(x^*, \theta)$ by anticipating these discontinuous changes during planning (we can do this since we know the nominal dynamics and the dataset), and smoothing out these discontinuities, e.g., as in red in Fig. C.3.

We can see that RHS is continuous for all $[d_c, d_e]^\top$ (since it is just a linear system). Furthermore, it is continuous in t, since all terms in RHS are constant, apart from the error bounds $\bar{\epsilon}_i$, which are continuous functions of t (using smoothing for $\bar{\epsilon}_3$). Moreover, using continuity of RHS over $t \in [0, T]$, the solution to RHS exists over $[0, T]$. A vector-valued function $g(t, r) : \mathbb{R} \times \mathbb{R}^N \to \mathbb{R}^N$ is quasi-monotone nondecreasing in r if $\frac{\partial g_i(t,r)}{\partial r_j} \geq 0$, for all r and for all $j \neq i$, $i = 1, \ldots, N$. We can see that RHS precisely satisfies these conditions, as the matrix in (10.19) is Metzler, i.e., all of its off-diagonal components are nonnegative: $L_{\Delta k}, L_p, \underline{\lambda}_{D_c}(M_c), \bar{\lambda}(M_e), \rho \geq 0$. More precisely, $\frac{\partial \text{RHS}_1(t,[d_c,d_e]^\top)}{\partial d_e} = L_{\Delta k} \geq 0$ and $\frac{\partial \text{RHS}_2(t,[d_c,d_e]^\top)}{\partial d_c} = (*) \geq 0$.

□

Theorem C.7 (CORRT correctness)**.** *Assume that $L_{\Delta k}$, $L_{\hat{h}^{-1}}$, and the estimated constants in $\bar{\epsilon}_{\{1,2,3\}}$ are valid over their computed domains. Then Alg. X.1 returns a trajectory $(x^*(t), u^*(t))$, which when tracked on the true system* (10.1a) *using $u(\hat{x}, x^*, u^*)$ with state estimates $\hat{x}$ generated by* (10.6)*, reaches $\mathcal{G}$ while satisfying $x(t) \in \mathcal{X}_{safe}$, for all $t \in [0, T]$.*

Proof. By construction, Alg. X.1 returns a plan $(x^*(t), u^*(t))$ which ensures that $\Omega_c(t) \cap \mathcal{X}_{\text{unsafe}} = \emptyset$, for all $t \in [0, T]$, where $\Omega_c(t) = \{x \mid d_c(x^*(t), x(t)) \leq \bar{d}_c(t)\}$, and $x(T) \in \mathcal{G}$. Thus, $x(t) \cap \mathcal{X}_{\text{unsafe}} = \emptyset$, provided that $x(t) \in \Omega_c(t)$, for all $t \in [0, T]$, i.e., the tubes are valid. Using Theorem X.3 and the assumption that all estimated constants are valid over their computed domains (from the theorem statement), the tubes are valid if $x(t)$ remains in the corresponding domains of validity for all constants in

(10.19). In the following, we exhaustively prove that the additional constraints introduced by Alg. X.1 enforce this domain invariance.

First, we list all constants in (10.20) with a non-trivial domain of validity.

1. CCM-related constants: the minimum and maximum eigenvalues of M_c: $\underline{\lambda}_{D_c}(M_c)$, $\bar{\lambda}_{D_c}(M_c)$, and the CCM contraction rate λ_c.

2. OCM-related constants: the minimum and maximum eigenvalues of M_e: $\underline{\lambda}(M_e)$, $\bar{\lambda}(M_e)$, the OCM contraction rate λ_e, and the multiplier ρ.

3. Constants estimated with probabilistic correctness guarantees using the Fisher-Tippett-Gnedenko (FTG) theorem *Chou et al.* (2021c); *Knuth et al.* (2021a); *Weng et al.* (2018): $L_{\Delta k}$, $L_{\hat{h}^{-1}}$, and all constants involved in the perception error bounds $\bar{\epsilon}_i(x^*, \theta)$, $i = 1, 2, 3$: $\bar{\epsilon}_1$, L_p, and $\mathcal{R}$.

Now, we prove why each group of constants is valid under the constraints of Alg. X.1:

1. Suppose M_c is constant; then, its maximum and minimum eigenvalues trivially extend for all of $\mathcal{X}$. If M_c is polynomial and is generated via the SoS program in (C.12), its eigenvalues are by construction bounded by $1/\underline{\beta}$ and $1/\bar{\beta}$; these provide globally-valid eigenvalue bounds over $\mathcal{X}$. Moreover, the CCM-based controller contracts the nominal dynamics at rate λ_c if 1) for all time $t \in [0, T]$, the geodesic connecting $x^*(t)$ and $x(t)$ (denoted as $\gamma_c^t(s), s \in [0, 1]$) is contained within the contraction domain D_c, and 2) there exists a feasible control input which achieves the λ_c contraction rate. Alg. X.1 enforces both of these conditions: 1) line 9 of Alg. X.1 ensures $\Omega_c(t) \subseteq D_c$, since Ω_c is defined as a sublevel set with respect to the Riemannian metric induced by M_c, i.e., $\gamma_c^t(s) \subseteq \Omega_c(t) \subseteq D_c$; 2) by (10.2), the feedback controller $u(\hat{x}, x^*, u^*)$ is always feasible (i.e., (10.13) is nonempty) as long as $\hat{x} \in D_c$; this is enforced by line 10 of Alg. X.1.

2. In this work, we use constant M_e, so its maximum and minimum eigenvalues trivially extend for all of $\mathcal{X}$. The OCM-based observer contracts at rate λ_e for its nominal dynamics if for all time $t \in [0, T]$, the geodesic connecting $x(t)$ and $\hat{x}(t)$ (denoted as $\gamma_e^t(s), s \in [0, 1]$) is contained within the contraction domain D_e. In Alg. X.1, line 10 ensures $\Omega_e(t) \subseteq D_e$, since Ω_e is defined as a sublevel set with respect to the Riemannian metric induced by W_e, $\gamma_e^t(s) \subseteq \Omega_e(t) \subseteq D_e$.

3. From its definition in the paragraph before (10.15), $L_{\Delta k}$ is valid if $d_c(t) \leq \bar{c}$ and $d_e(t) \leq \bar{e}$, and $x^* \in D_x$, $u^* \in \mathcal{U}$. We ensure this via the check in Alg. X.1, line 8. $L_{\hat{h}^{-1}}$, and the error bound constants $\bar{\epsilon}_1$, L_p, and $\mathcal{R}$ are valid if $x \in D_r$; this is checked in line 9 of Alg. X.1.

□

Remark C.8 (Overall correctness probability). If all FTG-estimated constants are obtained according to Rem. X.4 with probability ρ of correctness, using samples which

are mutually independent, then the overall probability of the correctness of CORRT can be bounded by the product of each constant being estimated correctly; for example, if $L_{\Delta k}$, $L_{\hat{h}^{-1}}$, and $\bar{\epsilon}_1$ are estimated, then CORRT returns a safely-trackable trajectory with probability at least ρ^3.

C.4 Appendix: Chapter X: Optimizing CCMs and OCMs for output feedback

In this section, we describe in more detail our method for synthesizing optimized CCMs and OCMs (described briefly in Sec 10.3.3. To implement the tracking feedback controller and observers needed at runtime, we need to obtain the CCMs and OCMs that define them. Two popular ways for synthesizing contraction metrics are convex optimization (SoS programming) *Manchester and Slotine* (2017) and learning *Chou et al.* (2021c). While the optimization-based methods are more efficient and easier to analyze than the learning-based approaches, the learning-based methods can be applied to higher-dimensional, non-polynomial systems. Since we focus on (approximately) polynomial systems in the results, we obtain CCMs and OCMs via SoS programming; however, our method is agnostic to how the CCMs/OCMs are generated, as long as all the associated constants (e.g., $\bar{\lambda}_{D_e}(M_e), \bar{\lambda}_{D_c}(M_c)$, etc.) can be accurately bounded.

To minimize conservativeness in planning (cf. Sec. 10.3.4), it is important that the contraction metrics induce small tubes. However, exactly optimizing the tubes in (10.20) is challenging, as they are coupled and depend on plan-dependent constants. For tractability, we optimize a surrogate objective and design the CCM and OCM separately.

Denote $-G_c(x)$ as the LHS of (10.2a), v as a vector of indeterminates of compatible dimension with $G_c(x)$, and $\bar{\beta} \geq \underline{\beta} > 0$. Let $\mathbf{I}_n$ be the $n \times n$ identity matrix. For the CCM, we minimize the steady-state tracking tube width for a uniform disturbance bound as in *Singh et al.* (2019) by solving the following:

$$\begin{array}{ll} \underset{W_c(x), \bar{W}_c, \lambda_c, \mu_i^e(x)}{\text{minimize}} & \frac{1}{\lambda_c}\sqrt{\frac{\bar{\lambda}_{D_c}(W_c)}{\underline{\lambda}_{D_c}(W_c)}} \\ \text{subject to} & \text{(10.2b)} \\ & v^\top G_c(x) v - \sum_i \mu_i^e(x) c_i(x) \in \text{SoS} \\ & \underline{\beta}\mathbf{I}_{n_x} \preceq W_c(x) \preceq \bar{W}_c \preceq \bar{\beta}\mathbf{I}_{n_x} \end{array} \tag{C.12}$$

We motivate (C.12) in the following. For polynomial dynamics and CCMs, the CCM condition (10.2) is equivalent to enforcing the nonnegativity of a polynomial function $p(x)$ over a domain. A sufficient condition for proving $p(x) \geq 0$ over a domain is to enforce that $p(x)$ can be written as a sum of squares; this can be enforced via a semidefinite constraint, which is convex and is referred to in (C.12) as $p(x) \in \text{SoS}$. As it may not be possible to enforce (10.2a) globally for all $x \in \mathcal{X}$, we further introduce nonnegative Lagrange multipliers $\mu_i^e(x)$ to only enforce (10.2a) over the subset $D_c \subseteq \mathcal{X}$. Specifically, we rewrite $x \in D_c \Leftrightarrow \bigwedge_{i=1}^{n_{\text{con}}} \{c_i(x) \leq 0\}$, where $c_i(\cdot)$ are constraint functions that together define the boundary of D_c; then, for some x

where $c_i(x) > 0$, $\mu_i^c(x)$ can take some positive value to satisfy the SoS condition even if $v^\top G(x)v < 0$. We handle condition (10.2b) by restricting $W_c(x)$ to be only a function of a subset of state variables (cf. *Singh et al.* (2019)), and we add additional constraints on the eigenvalues of W_c to ensure that its inverse is well-defined. As in *Singh et al.* (2019), while (C.12) is not convex as written, it can be solved to global optimality by solving a sequence of convex SoS feasibility programs, where a subset of the decision variables are fixed at each iteration. Specifically, we perform a line search over λ_c and a bisection search over the condition number $\bar{\lambda}_{D_c}(W_c)/\underline{\lambda}_{D_c}(W_c)$.

We take a similar approach for optimizing OCMs. First, we found it sufficient to use constant OCMs and multipliers ρ for the systems in this chapter due to the simplifications enabled by the inverse perception map. This simplifies the condition (10.4) to

$$W_e A(\hat{x}) + A(\hat{x})^\top W_e - \rho C_r^\top C_r + 2\lambda_e W_e \leq 0. \tag{C.13}$$

Let $-G_e(x)$ be the LHS of (C.13). We then obtain our OCM by solving the following:

$$\begin{array}{ll} \underset{W_e, \lambda_o, \mu_i^e(x)}{\text{minimize}} & \frac{1}{\lambda_c} \frac{\bar{\lambda}_{D_c}(W_c)}{\sqrt{\underline{\lambda}_{D_c}(W_c)}} \rho \\ \text{subject to} & v^\top G_e(x) v - \sum_i \mu_i^e(x) c_i^e(x) \in \text{SoS} \\ & \underline{\beta} \mathbf{I}_{n_x} \preceq W_e \preceq \bar{\beta} \mathbf{I}_{n_x} \end{array} \tag{C.14}$$

We change the objective function here to account for the known components of the disturbance bound; specifically, from (10.16) we know that any disturbance from the innovation term in the observer will be pre-multiplied by $\frac{1}{2}\rho M_e$; this accounts for the extra factors in the objective. As before, we introduce Lagrange multipliers $\mu_e(x)$ to only enforce (C.13) over the subset $D_e \subseteq \mathcal{X}$, and restrict the eigenvalues of W_e to ensure its inverse is well-defined. To solve (C.14) approximately, we drop the extra square root factor, and do the same line search on λ_e and bisection search on the condition number as for the CCM, but instead minimizing ρ instead of solving a feasibility problem in each SoS program. We note that for both CCM and OCM synthesis, if the system is linear, i.e., $\dot{x} = Ax + Bu$, then W_c and W_e can be chosen as constant, simplifying both (C.12) and (C.14) to a standard semidefinite program (SDP), since all constraints which were previously polynomial functions of x simplify to constant linear matrix inequalities (LMIs).

C.5 Appendix: Chapter X: System models

In this appendix, we provide an overview of the system models that we use in the results Sec. 10.4.

C.5.0.1 4D nonholonomic car:

$$\begin{bmatrix} \dot{p}_x \\ \dot{p}_y \\ \dot{\phi} \\ \dot{v} \end{bmatrix} = \begin{bmatrix} v\cos(\phi) \\ v\sin(\phi) \\ 0 \\ 0 \end{bmatrix} + \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} \omega \\ a \end{bmatrix} + \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 1 & 0 \\ 0 & 1 \end{bmatrix} w_x \tag{C.15}$$

where p_x and p_y are the x and y translations of the car, ϕ is its orientation, and v is its linear velocity. where $u = [\omega, a]^\top$. To make (C.15) compatible with SoS programming, when searching for the CCM and OCM, we polynomialize the dynamics by fitting a degree 5 polynomial to $\sin(\cdot)$ and $\cos(\cdot)$ over $[-\pi/2, \pi/2]$.

C.5.0.2 6D planar quadrotor:

We use the dynamics from (*Singh et al.*, 2019, p.20) with six states and two inputs:

$$\begin{bmatrix} \dot{p}_x \\ \dot{p}_y \\ \dot{\phi} \\ \dot{v}_x \\ \dot{v}_z \\ \ddot{\phi} \end{bmatrix} = \begin{bmatrix} v_x\cos(\phi) - v_z\sin(\phi) \\ v_x\sin(\phi) + v_z\cos(\phi) \\ \dot{\phi} \\ v_z\dot{\phi} - g\sin(\phi) \\ -v_x\dot{\phi} - g\cos(\phi) \\ 0 \end{bmatrix} + \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 1/m & 1/m \\ l/J & -l/J \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \end{bmatrix} + \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 1 & 0 \\ 0 & 1 \end{bmatrix} w_x, \tag{C.16}$$

where $x = [p_x, p_z, \phi, v_x, v_z, \dot{\phi}]$ models the linear/angular position and velocity, and $u = [u_1, u_2]$ models thrust. We use the parameters $m = 0.486$, $l = 0.25$, and $J = 0.07$. To make (C.16) compatible with SoS programming, when searching for the CCM and OCM, we polynomialize the dynamics by fitting a degree 5 polynomial to $\sin(\cdot)$ and $\cos(\cdot)$ over $[-\pi/2, \pi/2]$.

C.5.0.3 17D manipulation task:

Let the $m \times n$ zero matrix be denoted $\mathbf{0}_{m\times n}$. We define $\boldsymbol{\phi} = [\phi_1, \phi_2, \phi_3]^\top$, $\mathbf{j} = [j_1, j_2, \ldots, j_7]^\top$, and $\dot{\mathbf{j}} = [\dot{j}_1, \dot{j}_2, \ldots, \dot{j}_7]^\top$. We model the full 17D system as:

$$\begin{bmatrix} \dot{\boldsymbol{\phi}} \\ \dot{\mathbf{j}} \\ \ddot{\mathbf{j}} \end{bmatrix} = \begin{bmatrix} \mathbf{0}_{3\times3} & \mathbf{0}_{3\times7} & \mathbf{0}_{3\times7} \\ \mathbf{0}_{7\times3} & \mathbf{0}_{7\times7} & \mathbf{I}_{7\times7} \\ \mathbf{0}_{7\times3} & \mathbf{0}_{7\times7} & \mathbf{0}_{7\times7} \end{bmatrix} \begin{bmatrix} \boldsymbol{\phi} \\ \mathbf{j} \\ \dot{\mathbf{j}} \end{bmatrix} + \begin{bmatrix} \mathbf{0}_{3\times7} \\ \mathbf{0}_{7\times7} \\ \mathbf{I}_{7\times7} \end{bmatrix} u + \begin{bmatrix} \mathbf{0}_{3\times7} \\ \mathbf{0}_{7\times7} \\ \mathbf{I}_{7\times7} \end{bmatrix} w_x \tag{C.17}$$

Here, the ϕ_i are the Euler angles of the object relative to the end effector, the j_i are the Kuka joint angles, the $\dot{j}_i$ are the Kuka joint velocities, and u are the commanded joint accelerations.

The 14D subsystem referred to in the text is:

$$\begin{bmatrix} \dot{\mathbf{j}} \\ \ddot{\mathbf{j}} \end{bmatrix} = \begin{bmatrix} \mathbf{0}_{7\times7} & \mathbf{I}_{7\times7} \\ \mathbf{0}_{7\times7} & \mathbf{0}_{7\times7} \end{bmatrix} \begin{bmatrix} \mathbf{j} \\ \dot{\mathbf{j}} \end{bmatrix} + \begin{bmatrix} \mathbf{0}_{7\times7} \\ \mathbf{I}_{7\times7} \end{bmatrix} u \tag{C.18}$$

BIBLIOGRAPHY

BIBLIOGRAPHY

Abbasi-Yadkori, Y., P. L. Bartlett, V. Gabillon, and A. Malek (2017), Hit-and-run for sampling and planning in non-convex spaces, in *AISTATS 2017*.

Abbeel, P., and A. Y. Ng (2004), Apprenticeship learning via inverse reinforcement learning, in *International Conference on Machine Learning (ICML)*.

Abbeel, P., A. Coates, and A. Y. Ng (2010), Autonomous helicopter aerobatics through apprenticeship learning, *Int. J. Robotics Res.*, *29*(13), 1608–1639.

Agha-mohammadi, A., S. Chakravorty, and N. M. Amato (2014), FIRM: sampling-based feedback motion-planning under motion uncertainty and imperfect measurements, *Int. J. Robotics Res.*, *33*(2), 268–304.

Akametalu, A. K., J. F. Fisac, J. H. Gillula, S. Kaynama, M. N. Zeilinger, and C. J. Tomlin (2014), Reachability-based safe learning with gaussian processes, in *IEEE Conference on Decision and Control (CDC)*, pp. 1424–1431.

Alizadeh, F., and D. Goldfarb (2003), Second-order cone programming, *Math. Program.*, *95*(1).

Allaire, G., F. Jouve, and G. Michailidis (2016), Thickness control in structural optimization via a level set method, *Struct. and Multidisciplinary Optimization*.

Altman, E. (1999), Constrained markov decision processes.

Amin, K., N. Jiang, and S. P. Singh (2017), Repeated inverse reinforcement learning, in *Neural Information Processing Systems*, pp. 1813–1822.

Andersson, J. A. E., J. Gillis, G. Horn, J. B. Rawlings, and M. Diehl (2018), CasADi – A software framework for nonlinear optimization and optimal control, *Mathematical Programming Computation*.

Annpureddy, Y., C. Liu, G. E. Fainekos, and S. Sankaranarayanan (2011), S-taliro: A tool for temporal logic falsification for hybrid systems, in *Tools and Algorithms for the Construction and Analysis of Systems - 17th International Conference, TACAS*, pp. 254–257.

Aoude, G. S., B. D. Luders, J. M. Joseph, N. Roy, and J. P. How (2013), Probabilistically safe motion planning to avoid dynamic obstacles with uncertain motion patterns, *Autonomous Robots*, *35*(1), 51–76, doi:10.1007/s10514-013-9334-3.

Araki, B., K. Vodrahalli, T. Leech, C. I. Vasile, M. Donahue, and D. Rus (2019), Learning to plan with logical automata, in *Robotics: Science and Systems XV*.

Argall, B. D., S. Chernova, M. Veloso, and B. Browning (2009), A survey of robot learning from demonstration, *Robotics and Autonomous Systems*, *57*(5), 469–483.

Armesto, L., J. Bosga, V. Ivan, and S. Vijayakumar (2017), Efficient learning of constraints and generic null space policies, in *IEEE International Conference on Robotics and Automation, ICRA*.

Axelrod, B., L. P. Kaelbling, and T. Lozano-Pérez (2017), Provably safe robot navigation with obstacle uncertainty, in *Robotics: Science and Systems*.

Babes, M., V. Marivate, K. Subramanian, and M. L. Littman (2011), Apprenticeship learning about multiple intentions, in *Proceedings of the 28th International Conference on Machine Learning, ICML 2011, Bellevue, Washington, USA, June 28 - July 2, 2011*, pp. 897–904, Omnipress.

Bagnoli, M., and T. Bergstrom (2005), Log-concave probability and its applications, *Economic Theory*, *26*(2), 445–469.

Bahreinian, M., M. Mitjans, and R. Tron (2021), Robust sample-based output-feedback path planning, in *IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS*, pp. 5780–5787, IEEE.

Baier, C., and J. Katoen (2008), *Principles of model checking*, MIT Press.

Bakhirkin, A., T. Ferrère, and O. Maler (2018), Efficient parametric identification for STL, in *Proceedings of the 21st International Conference on Hybrid Systems: Computation and Control*, pp. 177–186.

Banijamali, E., R. Shu, M. Ghavamzadeh, H. H. Bui, and A. Ghodsi (2018), Robust locally-linear controllable embedding, in *International Conference on Artificial Intelligence and Statistics, AISTATS 2018, 9-11 April 2018, Playa Blanca, Lanzarote, Canary Islands, Spain*, *Proceedings of Machine Learning Research*, vol. 84, pp. 1751–1759, PMLR.

Beck, M., and S. Robins (2015), *Computing the Continuous Discretely: Integer-Point Enumeration in Polyhedra*, pp. 167–182, Springer New York, New York, NY.

Ben-Tal, A., A. P. Goryashko, E. Guslitzer, and A. Nemirovski (2004), Adjustable robust solutions of uncertain linear programs, *Math. Program.*, *99*(2), 351–376.

Ben-Tal, A., L. E. Ghaoui, and A. Nemirovski (2009), *Robust Optimization*, *Princeton Series in Applied Mathematics*, vol. 28, Princeton University Press.

Berenson, D. (2011), Constrained manipulation planning, Ph.D. thesis, Carnegie Mellon University, Pittsburgh, PA.

Berenson, D., S. S. Srinivasa, and J. J. Kuffner (2011), Task space regions: A framework for pose-constrained manipulation planning, *International Journal of Robotics Research (IJRR)*, *30*(12), 1435–1460.

Berkenkamp, F., R. Moriconi, A. P. Schoellig, and A. Krause (2016), Safe learning of regions of attraction for uncertain, nonlinear systems with gaussian processes, in *IEEE Conference on Decision and Control (CDC)*, pp. 4661–4666.

Berkenkamp, F., M. Turchetta, A. Schoellig, and A. Krause (2017), Safe model-based reinforcement learning with stability guarantees, in *Neural Information Processing Systems*, pp. 908–918.

Bertsekas, D. (1972), Infinite time reachability of state-space regions by using feedback control, *IEEE Transactions on Automatic Control*, *17*(5), 604–613, doi: 10.1109/TAC.1972.1100085.

Bertsimas, D., and B. Stellato (2019), Online mixed-integer optimization in milliseconds, *CoRR*, *abs/1907.02206*.

Bertsimas, D., and J. Tsitsiklis (1997), *Introduction to Linear Optimization*, 1st ed., Athena Scientific.

Biere, A., K. Heljanko, T. A. Junttila, T. Latvala, and V. Schuppan (2006), Linear encodings of bounded LTL model checking, *Logical Methods in Computer Science*, *2*(5).

Blackmore, L., H. Li, and B. Williams (2006), A probabilistic approach to optimal robust path planning with obstacles, in *American Control Conference (ACC)*.

Bochnak, J., M. Coste, and M.-F. Roy (1998), *Real Algebraic Geometry*, Springer.

Boffi, N. M., S. Tu, N. Matni, J. E. Slotine, and V. Sindhwani (2020), Learning stability certificates from data, *Conference on Robot Learning*.

Bombara, G., C. I. Vasile, F. Penedo, H. Yasuoka, and C. Belta (2016), A decision tree approach to data classification using signal temporal logic, in *Proceedings of the 19th International Conference on Hybrid Systems: Computation and Control, HSCC 2016*, pp. 1–10.

Bonet, B., and H. Geffner (2000), Planning with incomplete information as heuristic search in belief space, in *Proceedings of the Fifth International Conference on Artificial Intelligence Planning Systems, Breckenridge, CO, USA, April 14-17, 2000*, edited by S. A. Chien, S. Kambhampati, and C. A. Knoblock, pp. 52–61, AAAI.

Bonnabel, S., and J. E. Slotine (2015), A contraction theory-based analysis of the stability of the deterministic extended kalman filter, *TAC*, *60*(2), 565–569.

Boyd, S., and L. Vandenberghe (2004), *Convex Optimization*, Cambridge University Press, New York, NY, USA.

Brown, D. S., R. Coleman, R. Srinivasan, and S. Niekum (2020), Safe imitation learning via fast bayesian reward inference from preferences, *International Conference on Machine Learning*.

Bry, A., and N. Roy (2011), Rapidly-exploring random belief trees for motion planning under uncertainty, in *IEEE International Conference on Robotics and Automation, ICRA*.

Bufo, S., E. Bartocci, G. Sanguinetti, M. Borelli, U. Lucangelo, and L. Bortolussi (2014), Temporal logic based monitoring of assisted ventilation in intensive care patients, in *Leveraging Applications of Formal Methods, Verification and Validation. Specialized Techniques and Applications - 6th International Symposium, ISoLA 2014*, pp. 391–403.

Burns, B., and O. Brock (2007), Sampling-based motion planning with sensing uncertainty, in *IEEE International Conference on Robotics and Automation, ICRA*.

Calinon, S., and A. Billard (2007), Incremental learning of gestures by imitation in a humanoid robot, in *ACM/IEEE International Conference on Human-Robot Interaction (HRI)*, pp. 255–262.

Calinon, S., and A. Billard (2008), A probabilistic programming by demonstration framework handling constraints in joint space and task space, in *International Conference on Intelligent Robots and Systems (IROS)*.

Çalli, B., A. Singh, J. Bruce, A. Walsman, K. Konolige, S. S. Srinivasa, P. Abbeel, and A. M. Dollar (2017), Yale-cmu-berkeley dataset for robotic manipulation research, *Int. J. Robotics Res.*, *36*(3), 261–268.

Calliess, J. (2014), *Conservative decision-making&inference in uncertain dynamical systems*.

Camacho, A., and S. A. McIlraith (2019), Learning interpretable models expressed in linear temporal logic, in *Proceedings of the Twenty-Ninth International Conference on Automated Planning and Scheduling, ICAPS 2018*, pp. 621–630.

Choi, J., and K. Kim (2012), Nonparametric bayesian inverse reinforcement learning for multiple reward functions, in *Advances in Neural Information Processing Systems 25: 26th Annual Conference on Neural Information Processing Systems 2012. Proceedings of a meeting held December 3-6, 2012, Lake Tahoe, Nevada, United States*, pp. 314–322.

Choi, S., K. Lee, S. Lim, and S. Oh (2018), Uncertainty-aware learning from demonstration using mixture density networks with sampling-free variance modeling, in *IEEE International Conference on Robotics and Automation, ICRA*.

Chou, G., D. Berenson, and N. Ozay (2018a), Learning constraints from demonstrations, *Workshop on the Algorithmic Foundations of Robotics (WAFR)*.

Chou, G., N. Ozay, and D. Berenson (2018b), Incremental segmentation of arx models, *IFAC-PapersOnLine*, *51*(15), 587–592, 18th IFAC Symposium on System Identification SYSID 2018.

Chou, G., Y. E. Sahin, L. Yang, K. J. Rutledge, P. Nilsson, and N. Ozay (2018c), Using control synthesis to generate corner cases: A case study on autonomous driving, *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst.*, *37*(11), 2906–2917.

Chou, G., N. Ozay, and D. Berenson (2019), Learning parametric constraints in high dimensions from demonstrations, *3rd Conference on Robot Learning (CoRL)*.

Chou, G., N. Ozay, and D. Berenson (2020a), Uncertainty-aware constraint learning for adaptive safe motion planning from demonstrations, in *CoRL 2020, Cambridge, MA, USA*.

Chou, G., N. Ozay, and D. Berenson (2020b), Learning constraints from locally-optimal demonstrations under cost function uncertainty, *Robotics and Automation Letters (RA-L)*.

Chou, G., N. Ozay, and D. Berenson (2020c), Explaining Multi-stage Tasks by Learning Temporal Logic Formulas from Suboptimal Demonstrations, in *Robotics: Science and Systems*, Corvalis, Oregon, USA, doi:10.15607/RSS.2020.XVI.097.

Chou, G., D. Berenson, and N. Ozay (2021a), Learning constraints from demonstrations with grid and parametric representations, *Int. J. Robotics Res.*, *40*(10-11).

Chou, G., N. Ozay, and D. Berenson (2021b), Learning temporal logic formulas from suboptimal demonstrations: theory and experiments, *Autonomous Robots*.

Chou, G., N. Ozay, and D. Berenson (2021c), Model error propagation via learned contraction metrics for safe feedback motion planning of unknown systems, *Conference on Decision and Control*.

Chou, G., N. Ozay, and D. Berenson (2022a), Safe output feedback motion planning from images via learned perception modules and contraction theory, in *Workshop on the Algorithmic Foundations of Robotics (WAFR)*.

Chou, G., H. Wang, and D. Berenson (2022b), Gaussian process constraint learning for scalable chance-constrained motion planning from demonstrations, *IEEE Robotics and Automation Letters (RA-L)*.

Chow, Y., O. Nachum, E. A. Duéñez-Guzmán, and M. Ghavamzadeh (2018), A lyapunov-based approach to safe reinforcement learning, in *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, December 3-8, 2018, Montréal, Canada*, edited by S. Bengio, H. M. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, pp. 8103–8112.

Chow, Y., O. Nachum, A. Faust, M. Ghavamzadeh, and E. A. Duéñez-Guzmán (2019), Lyapunov-based safe policy optimization for continuous control, *CoRR*, *abs/1901.10031*.

Chua, K., R. Calandra, R. McAllister, and S. Levine (2018), Deep reinforcement learning in a handful of trials using probabilistic dynamics models, in *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, December 3-8, 2018, Montréal, Canada*, pp. 4759–4770.

Collins, G. E. (1975), Hauptvortrag: Quantifier elimination for real closed fields by cylindrical algebraic decomposition, in *Automata Theory and Formal Languages*, pp. 134–183.

Cosner, R., A. Singletary, A. Taylor, T. Molnár, K. Bouman, and A. Ames (2021), Measurement-robust control barrier functions: Certainty in safety with uncertainty in state, in *IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS*.

Cui, Y., D. Isele, S. Niekum, and K. Fujimura (2019), Uncertainty-aware data aggregation for deep imitation learning, in *IEEE International Conference on Robotics and Automation, ICRA*, pp. 761–767.

Dacorogna, B. (2015), *Introduction to the calculus of variations*, Imp. College Press.

Dani, A. P., S. Chung, and S. Hutchinson (2015), Observer design for stochastic nonlinear systems via contraction-based incremental stability, *TAC*, *60*(3), 700–714.

Dawson, C., B. Lowenkamp, D. Goff, and C. Fan (2022), Learning safe, generalizable perception-based hybrid control with certificates, *Robotics and Automation Letters*.

De Haan, L., and A. Ferreira (2007), *Extreme value theory: an introduction*, Springer Science & Business Media.

Dean, S., N. Matni, B. Recht, and V. Ye (2020a), Robust guarantees for perception-based control, in *L4DC*, vol. 120, pp. 350–360, PMLR.

Dean, S., A. J. Taylor, R. K. Cosner, B. Recht, and A. D. Ames (2020b), Guaranteeing safety of learned perception modules via measurement-robust control barrier functions, in *Conference on Robot Learning*.

Deglurkar, S., M. H. Lim, J. Tucker, Z. N. Sunberg, A. Faust, and C. J. Tomlin (2021), Visual learning-based planning for continuous high-dimensional pomdps, *CoRR*, *abs/2112.09456*.

DeGroot, M., and M. Schervish (2013), *Probability & Statistics*,Pearson.

Deisenroth, M. P., and C. E. Rasmussen (2011), PILCO: A model-based and data-efficient approach to policy search, in *Proceedings of the 28th International Conference on Machine Learning, ICML 2011, Bellevue, Washington, USA, June 28 - July 2, 2011*, edited by L. Getoor and T. Scheffer, pp. 465–472, Omnipress.

Deits, R., and R. Tedrake (2014), Computing large convex regions of obstacle-free space through semidefinite programming, in *Algorithmic Foundations of Robotics XI - Selected Contributions of the Eleventh International Workshop on the Algorithmic Foundations of Robotics, WAFR 2014, 3-5 August 2014, Boğaziçi University, İstanbul, Turkey*, *Springer Tracts in Advanced Robotics*, vol. 107, pp. 109–124, Springer.

Demri, S., and P. Schnoebelen (2002), The complexity of propositional linear temporal logics in simple cases, *Inf. Comput.*, *174*(1), 84–103.

Deng, X., A. Mousavian, Y. Xiang, F. Xia, T. Bretl, and D. Fox (2019), Poserbpf: A rao-blackwellized particle filter for 6d object pose estimation, in *Robotics: Science and Systems XV*.

Dor, A., E. Greenshtein, and E. Korach (1998), Optimal and myopic search in a binary random vector, *Journal of Applied Probability*, *35*(2).

Englert, P., N. A. Vien, and M. Toussaint (2017), Inverse kkt: Learning cost functions of manipulation tasks from demonstrations, *International Journal of Robotics Research (IJRR)*, *36*(13-14), 1474–1488.

Fan, D. D., A. Agha-mohammadi, and E. A. Theodorou (2020), Deep learning tubes for tube MPC, *Robotics: Science and Systems*.

Fazlyab, M., A. Robey, H. Hassani, M. Morari, and G. Pappas (2019), Efficient and accurate estimation of lipschitz constants for deep neural networks, in *Neural Information Processing Systems*, pp. 11,427–11,438.

Findeisen, R., L. Imsland, and F. Allgöwer (2003), State and output feedback nonlinear model predictive control: An overview, *Eur. J. Control*, *9*(2-3), 190–206.

Finn, C., S. Levine, and P. Abbeel (2016), Guided cost learning: Deep inverse optimal control via policy optimization, in *International Conference on Machine Learning*, vol. 48, pp. 49–58.

Fraichard, T., and H. Asama (2004), Inevitable collision states - a step towards safer robots?, *Adv. Robotics*, *18*(10), 1001–1024.

Fu, J., S. Levine, and P. Abbeel (2016), One-shot learning of manipulation skills with online dynamics adaptation and neural network priors, in *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS 2016, Daejeon, South Korea, October 9-14, 2016*, pp. 4019–4026, IEEE.

Fu, J., I. Papusha, and U. Topcu (2017), Sampling-based approximate optimal control under temporal logic constraints, in *Proceedings of the 20th International Conference on Hybrid Systems: Computation and Control, HSCC 2017*, pp. 227–235.

Genz, A., and G. Trinh (2014), Numerical computation of multivariate normal probabilities using bivariate conditioning, in *MCQMC*.

Golub, G. H., and C. F. Van Loan (1996), *Matrix Computations (3rd Ed.)*.

Grammatico, S., X. Zhang, K. Margellos, P. J. Goulart, and J. Lygeros (2016), A scenario approach for non-convex control design, *IEEE Trans. Autom. Control.*, *61*(2), 334–345.

Gurobi Optimization, L. (2020), Gurobi optimizer reference manual.

Guzzi, J., R. Chavez-Garcia, M. Nava, L. Gambardella, and A. Giusti (2020), Path planning with local motion estimations, *Robotics and Automation Letters*, *5*(2).

gym-kuka mujoco (2019), Harvardagileroboticslab/gym-kuka-mujoco.

Haarnoja, T., A. Ajay, S. Levine, and P. Abbeel (2016), Backprop KF: learning discriminative deterministic state estimators, in *Neural Information Processing Systems*, edited by D. D. Lee, M. Sugiyama, U. von Luxburg, I. Guyon, and R. Garnett, pp. 4376–4384.

Hafner, D., T. P. Lillicrap, I. Fischer, R. Villegas, D. Ha, H. Lee, and J. Davidson (2019), Learning latent dynamics for planning from pixels, in *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA*, *Proceedings of Machine Learning Research*, vol. 97, edited by K. Chaudhuri and R. Salakhutdinov, pp. 2555–2565, PMLR.

Hart, P. E., N. J. Nilsson, and B. Raphael (1968), A formal basis for the heuristic determination of minimum cost paths, *IEEE Trans. Syst. Sci. Cybern.*, *4*(2), 100–107.

Hauser, K. K. (2014), The minimum constraint removal problem with three robotics applications, *International Journal of Robotics Research (IJRR)*, *33*(1), 5–17.

Herbert, S. L., M. Chen, S. Han, S. Bansal, J. F. Fisac, and C. J. Tomlin (2017), Fastrack: A modular framework for fast and guaranteed safe motion planning, in *56th IEEE Annual Conference on Decision and Control, CDC 2017, Melbourne, Australia, December 12-15, 2017*, pp. 1517–1522, IEEE.

Herceg, M., M. Kvasnica, C. Jones, and M. Morari (2013), Multi-Parametric Toolbox 3.0, in *European Control Conference*.

Huang, Y., H. Zhang, Y. Shi, J. Z. Kolter, and A. Anandkumar (2021), Training certifiably robust neural networks with efficient local lipschitz bounds, in *Advances*

in Neural Information Processing Systems 34: Annual Conference on Neural Information Processing Systems 2021, NeurIPS 2021, December 6-14, 2021, virtual, pp. 22,745–22,757.

Ichter, B., and M. Pavone (2019), Robot motion planning in learned latent spaces, *IEEE Robotics and Automation Letters*, *4*(3), 2407–2414.

Igl, M., L. Zintgraf, T. A. Le, F. Wood, and S. Whiteson (2018), Deep variational reinforcement learning for POMDPs, in *Proceedings of the 35th International Conference on Machine Learning*, *Proceedings of Machine Learning Research*, vol. 80, edited by J. Dy and A. Krause, pp. 2117–2126, PMLR.

Janson, L., T. Hu, and M. Pavone (2018), Safe motion planning in unknown environments: Optimality benchmarks and tractable policies, in *Robotics: Science and Systems*.

Jha, S. (2017), susmitjha/telex.

Jha, S., A. Tiwari, S. A. Seshia, T. Sahai, and N. Shankar (2019), Telex: learning signal temporal logic from positive examples using tightness metric, *Formal Methods in System Design*, *54*(3), 364–387.

Jha, S. K., E. M. Clarke, C. J. Langmead, A. Legay, A. Platzer, and P. Zuliani (2009), A bayesian approach to model checking biological systems, in *Computational Methods in Systems Biology, 7th International Conference, CMSB 2009*, pp. 218–234.

Johnson, M., N. Aghasadeghi, and T. Bretl (2013), Inverse optimal control for deterministic continuous-time nonlinear systems, in *IEEE Conference on Decision and Control (CDC)*.

Jonschkowski, R., D. Rastogi, and O. Brock (2018), Differentiable particle filters: End-to-end learning with algorithmic priors, in *Robotics: Science and Systems XIV, Carnegie Mellon University, Pittsburgh, Pennsylvania, USA, June 26-30, 2018*, edited by H. Kress-Gazit, S. S. Srinivasa, T. Howard, and N. Atanasov.

Jordan, M., and A. G. Dimakis (2020), Exactly computing the local lipschitz constant of relu networks, in *Neural Information Processing Systems*.

Joseph, D. P., and T. J. Tou (1961), On linear control theory, *Transactions of the American Institute of Electrical Engineers, Part II: Applications and Industry*, *80*(4), 193–196.

Jr., R. P., R. Tedrake, L. P. Kaelbling, and T. Lozano-Pérez (2010), Belief space planning assuming maximum likelihood observations, in *Robotics: Science and Systems VI, Universidad de Zaragoza, Zaragoza, Spain, June 27-30, 2010*, edited by Y. Matsuoka, H. F. Durrant-Whyte, and J. Neira, The MIT Press.

Kaelbling, L. P., M. L. Littman, and A. R. Cassandra (1998), Planning and acting in partially observable stochastic domains, *Artif. Intell.*, *101*(1-2), 99–134.

Kalakrishnan, M., S. Chitta, E. A. Theodorou, P. Pastor, and S. Schaal (2011), STOMP: stochastic trajectory optimization for motion planning, in *IEEE International Conference on Robotics and Automation, ICRA*.

Kalman, R. (1960), On the general theory of control systems, *IFAC Proceedings Volumes*, *1*(1), 491–502, 1st International IFAC Congress on Automatic and Remote Control, Moscow, USSR, 1960.

Kalman, R. E. (1964), When is a linear control system optimal?, *Journal of Basic Engineering*, *86*(1), 51–60.

Karaman, S., and E. Frazzoli (2010), Incremental sampling-based algorithms for optimal motion planning, in *Robotics: Science and Systems*.

Karkus, P., D. Hsu, and W. S. Lee (2017), Qmdp-net: Deep learning for planning under partial observability, in *Neural Information Processing Systems*, pp. 4694–4704.

Karkus, P., D. Hsu, and W. S. Lee (2018), Particle filter networks with application to visual localization, in *Proceedings of The 2nd Conference on Robot Learning*, *Proceedings of Machine Learning Research*, vol. 87, edited by A. Billard, A. Dragan, J. Peters, and J. Morimoto, pp. 169–178, PMLR.

Kavraki, L. E., P. Svestka, J. Latombe, and M. H. Overmars (1996), Probabilistic roadmaps for path planning in high-dimensional configuration spaces, *IEEE Trans. Robotics Autom.*, *12*(4), 566–580.

Kawano, Y., and Y. Hosoe (2021), Contraction analysis of discrete-time stochastic systems.

Keshavarz, A., Y. Wang, and S. P. Boyd (2011), Imputing a convex objective function, in *IEEE International Symposium on Intelligent Control (ISIC)*, pp. 613–619, IEEE.

Khalil, H. K. (2002), *Nonlinear systems*, Prentice-Hall, Upper Saddle River, NJ.

Kiatsupaibul, S., R. L. Smith, and Z. B. Zabinsky (2011), An analysis of a variation of hit-and-run for uniform sampling from general regions, *TOMACS*.

Kloss, A., G. Martius, and J. Bohg (2021), How to train your differentiable filter, *Autonomous Robots*.

Knuth, C., G. Chou, N. Ozay, and D. Berenson (2021a), Planning with learned dynamics: Probabilistic guarantees on safety and reachability via lipschitz constants, *IEEE Robotics and Automation Letters (RA-L)*.

Knuth, C., G. Chou, N. Ozay, and D. Berenson (2021b), Inferring obstacles and path validity from visibility-constrained demonstrations, in *Algorithmic Foundations of Robotics XIV, Proceedings of the Fourteenth Workshop on the Algorithmic Foundations of Robotics, WAFR 2021, Oulu, Finland, June 21-23, 2021*, *Springer Proceedings in Advanced Robotics*, vol. 17, pp. 18–36, Springer.

Köhler, J., R. Soloperto, M. A. Müller, and F. Allgöwer (2021), A computationally efficient robust model predictive control framework for uncertain nonlinear systems, *IEEE Trans. Autom. Control.*, *66*(2), 794–801.

Koller, T., F. Berkenkamp, M. Turchetta, and A. Krause (2018), Learning-based model predictive control for safe exploration, in *IEEE Conference on Decision and Control (CDC)*.

Kong, Z., A. Jones, A. M. Ayala, E. A. Gol, and C. Belta (2014), Temporal logic inference for classification and prediction from data, in *17th International Conference on Hybrid Systems: Computation and Control (part of CPS Week), HSCC'14*, pp. 273–282.

Kong, Z., A. Jones, and C. Belta (2017), Temporal logics for learning and detection of anomalous behavior, *IEEE Transactions on Automatic Control*, *62*(3), 1210–1222.

Kousik, S., S. Vaskov, M. Johnson-Roberson, and R. Vasudevan (2017), Safe trajectory synthesis for autonomous driving in unforeseen environments, *CoRR*, *abs/1705.00091*.

Kress-Gazit, H., G. E. Fainekos, and G. J. Pappas (2009), Temporal-logic-based reactive mission and motion planning, *IEEE Trans. Robotics*, *25*(6), 1370–1381.

Krishnan, S., A. Garg, R. Liaw, B. Thananjeyan, L. Miller, F. T. Pokorny, and K. Goldberg (2019), SWIRL: A sequential windowed inverse reinforcement learning algorithm for robot tasks with delayed rewards, *International Journal of Robotics Research (IJRR)*, *38*(2-3).

Kurniawati, H., D. Hsu, and W. S. Lee (2008), SARSOP: efficient point-based POMDP planning by approximating optimally reachable belief spaces, in *Robotics: Science and Systems IV, Eidgenössische Technische Hochschule Zürich, Zurich, Switzerland, June 25-28, 2008*, The MIT Press.

Lakshiliikantham, V., and S. Leela (1969), in *Differential and Integral Inequalities - Theory and Applications: Ordinary Differential Equations*, vol. 55, pp. 3–44.

Lakshmanan, A., A. Gahlawat, and N. Hovakimyan (2020), Safe feedback motion planning: A contraction theory and l_1-adaptive control based approach, *IEEE Conference on Decision and Control (CDC)*.

LaValle, S. (2006), *Planning algorithms*, Cambridge university press.

LaValle, S. M., and J. James J. Kuffner (2001), Randomized kinodynamic planning, *International Journal of Robotics Research (IJRR)*, *20*(5), 378–400.

Leung, K., and I. R. Manchester (2017), Nonlinear stabilization via control contraction metrics: A pseudospectral approach for computing geodesics, in *American Control Conference (ACC)*, pp. 1284–1289, IEEE.

Leung, K., N. Aréchiga, and M. Pavone (2019), Backpropagation for parametric STL, in *2019 IEEE Intelligent Vehicles Symposium, IV*, pp. 185–192.

Levine, S., and P. Abbeel (2014), Learning neural network policies with guided policy search under unknown dynamics, in *Advances in Neural Information Processing Systems 27: Annual Conference on Neural Information Processing Systems 2014, December 8-13 2014, Montreal, Quebec, Canada*, pp. 1071–1079.

Levine, S., Z. Popovic, and V. Koltun (2011), Nonlinear inverse reinforcement learning with gaussian processes, in *Neural Information Processing Systems*, pp. 19–27.

Levine, S., C. Finn, T. Darrell, and P. Abbeel (2016), End-to-end training of deep visuomotor policies, *J. Mach. Learn. Res.*, *17*, 39:1–39:40.

Lew, T., R. Bonalli, and M. Pavone (2020), Chance-constrained sequential convex programming for robust trajectory optimization, in *18th European Control Conference, ECC 2020, Virtual Event, Russia, May 12-15, 2020*, pp. 1871–1878, IEEE.

Li, C., and D. Berenson (2016), Learning object orientation constraints and guiding constraints for narrow passages from one demonstration, in *International Symposium of Experimental Robotics (ISER)*.

Li, C.-K., and F. Zhang (2019), Eigenvalue continuity and gersgorin's theorem, *Electronic Journal of Linear Algebra*, *35*(1), 619–625, doi:10.13001/1081-3810.4123.

Li, L., and J. Fu (2017), Sampling-based approximate optimal temporal logic planning, in *2017 IEEE International Conference on Robotics and Automation, ICRA*, pp. 1328–1335.

Li, L., X. Qi, T. Xie, and B. Li (2020), Sok: Certified robustness for deep neural networks, *arXiv preprint arXiv:2009.04131*.

Liberti, L., and C. C. Pantelides (2006), An exact reformulation algorithm for large nonconvex nlps involving bilinear terms, *J. Global Optimization*, *36*(2), 161–189.

Lin, H., M. Howard, and S. Vijayakumar (2015), Learning null space projections, in *IEEE International Conference on Robotics and Automation, ICRA 2015, Seattle, WA, USA, 26-30 May, 2015*, pp. 2613–2619.

Lin, H., P. Ray, and M. Howard (2017), Learning task constraints in operational space formulation, in *2017 IEEE International Conference on Robotics and Automation, ICRA 2017, Singapore, Singapore, May 29 - June 3, 2017*, pp. 309–315.

Liu, C., T. Arnon, C. Lazarus, C. A. Strong, C. W. Barrett, and M. J. Kochenderfer (2021), Algorithms for verifying deep neural networks, *Found. Trends Optim.*, *4*(3-4), 244–404, doi:10.1561/2400000035.

Lohmiller, W., and J. E. Slotine (1998), On contraction analysis for non-linear systems, *Autom.*, *34*(6), 683–696.

Lopez, B. T., J. E. Slotine, and J. P. How (2021), Robust adaptive control barrier functions: An adaptive & data-driven approach to safety, *IEEE Control. Syst. Lett.*, *5*(3), 1031–1036.

Lötstedt, P. (1983), Perturbation bounds for the linear least squares problem subject to linear inequality constraints, *BIT Numerical Mathematics*, *23*(4), 500–519.

Luders, B., M. Kothari, and J. How (2010), Chance constrained rrt for probabilistic robustness to environmental uncertainty, in *AIAA Guidance, Navigation, Control Conference*.

Luders, B., S. Karaman, and J. How (2013), Robust sampling-based motion planning with asymptotic optimality guarantees, in *AIAA Guidance, Navigation, Control Conference*.

Luenberger, D. (1971), An introduction to observers, *IEEE Transactions on Automatic Control*, *16*(6), 596–602.

Magron, V., D. Henrion, and J. Lasserre (2015), Semidefinite approximations of projections and polynomial images of semialgebraic sets, *SIAM Journal on Optimization*, *25*(4), 2143–2164.

Majumdar, A., and R. Tedrake (2017), Funnel libraries for real-time robust feedback motion planning, *International Journal of Robotics Research (IJRR)*, *36*(8), 947–982.

Manchester, I. R., and J. E. Slotine (2014), Output-feedback control of nonlinear systems using control contraction metrics and convex optimization, in *Australian Control Conference*.

Manchester, I. R., and J. E. Slotine (2017), Control contraction metrics: Convex and intrinsic criteria for nonlinear feedback design, *IEEE Trans. Autom. Control.*, *62*(6), 3046–3053.

Manchester, I. R., J. Z. Tang, and J. E. Slotine (2015), Unifying robot trajectory tracking with control contraction metrics, in *International Symposium of Robotics Research (ISRR)*, vol. 3, pp. 403–418, Springer.

Manek, G., and J. Z. Kolter (2019), Learning stable deep dynamics models, in *Neural Information Processing Systems*, pp. 11,126–11,134.

Maybeck, P. S. (1979), Stochastic models, estimation, and control.

Mayne, D. Q., J. B. Rawlings, C. V. Rao, and P. O. M. Scokaert (2000), Constrained model predictive control: Stability and optimality, *Autom.*, *36*(6), 789–814, doi: 10.1016/S0005-1098(99)00214-9.

Mayne, D. Q., M. M. Seron, and S. V. Rakovic (2005), Robust model predictive control of constrained linear systems with bounded disturbances, *Autom.*, *41*(2), 219–224.

McConachie, D., T. Power, P. Mitrano, and D. Berenson (2020), Learning when to trust a dynamics model for planning in reduced state spaces, *IEEE Robotics and Automation Letters*, *5*(2), 3540–3547.

Mehr, N., R. Horowitz, and A. D. Dragan (2016), Inferring and assisting with constraints in shared autonomy, in *IEEE Conference on Decision and Control (CDC)*, pp. 6689–6696.

Menda, K., K. R. Driggs-Campbell, and M. J. Kochenderfer (2019), Ensembledagger: A bayesian approach to safe imitation learning, in *IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS*, pp. 5041–5048.

Menner, M., P. Worsnop, and M. N. Zeilinger (2019), Constrained inverse optimal control with application to a human manipulation task, *IEEE Transactions on Control Systems Technology*.

Mitchell, I. M., A. M. Bayen, and C. J. Tomlin (2005), A time-dependent hamilton-jacobi formulation of reachable sets for continuous dynamic games, *TAC*, *50*(7), 947–957.

Mitrano, P., D. McConachie, and D. Berenson (2021), Learning where to trust unreliable models in an unstructured world for deformable object manipulation, *Science Robotics*, *6*(54).

Mode, G. R., and K. A. Hoque (2020), Adversarial examples in deep learning for multivariate time series regression.

Moerland, T. M., J. Broekens, and C. M. Jonker (2020), Model-based reinforcement learning: A survey, *CoRR*, *abs/2006.16712*.

Morin, T. L. (1982), Monotonicity and the principle of optimality, *Journal of Mathematical Analysis and Applications*, *88*(2), 665 – 674, doi:https://doi.org/10.1016/0022-247X(82)90223-2.

Neider, D., and I. Gavran (2018), Learning linear temporal properties, in *2018 Formal Methods in Computer Aided Design, FMCAD 2018*, pp. 1–10.

Ng, A. Y., and S. J. Russell (2000), Algorithms for inverse reinforcement learning, in *International Conference on Machine Learning (ICML)*, pp. 663–670, San Francisco, CA, USA.

Nguyen, Q. P., K. H. Low, and P. Jaillet (2015), Inverse reinforcement learning with locally consistent reward functions, in *Advances in Neural Information Processing Systems 28: Annual Conference on Neural Information Processing Systems 2015, December 7-12, 2015, Montreal, Quebec, Canada*, pp. 1747–1755.

Osa, T., J. Pajarinen, G. Neumann, J. A. Bagnell, P. Abbeel, and J. Peters (2018), An algorithmic perspective on imitation learning, *Found. Trends Robotics*, *7*(1-2), 1–179.

Pais, A. L., K. Umezawa, Y. Nakamura, and A. Billard (2013), Learning robot skills through motion segmentation and constraints extraction, ACM/IEEE International Conference on Human-Robot Interaction (HRI).

Papadimitriou, C. H., and K. Steiglitz (1982), *Combinatorial Optimization: Algorithms and Complexity*, Prentice Hall, Englewood Cliffs, NJ.

Papusha, I., M. Wen, and U. Topcu (2018), Inverse optimal control with regular language specifications, in *2018 Annual American Control Conference, ACC 2018*, pp. 770–777.

Park, J. J., P. Florence, J. Straub, R. Newcombe, and S. Lovegrove (2019), Deepsdf: Learning continuous signed distance functions for shape representation, in *CVPR*, pp. 165–174.

Pauli, P., A. Koch, J. Berberich, P. Kohler, and F. Allgöwer (2022), Training robust neural networks using lipschitz bounds, *IEEE Control. Syst. Lett.*, *6*, 121–126.

Pérez-D'Arpino, C., and J. A. Shah (2017), C-LEARN: learning geometric constraints from demonstrations for multi-step manipulation in shared autonomy, in *IEEE International Conference on Robotics and Automation, ICRA*.

Plappert, M., et al. (2018), Multi-goal reinforcement learning: Challenging robotics environments and request for research, *CoRR*, *abs/1802.09464*.

Potter, J. (1964), *A guidance-navigation separation theorem*.

Prentice, S., and N. Roy (2009), The belief roadmap: Efficient planning in belief space by factoring the covariance, *The International Journal of Robotics Research*, *28*(11-12), 1448–1465.

Quinlan, S. (1994), *Real-time modification of collision-free paths*, 1537, Stanford.

Rahimi, A., and B. Recht (2007), Random features for large-scale kernel machines, in *Neural Information Processing Systems*, pp. 1177–1184.

Rakovic, S. V., B. Kouvaritakis, M. Cannon, C. Panos, and R. Findeisen (2012), Parameterized tube model predictive control, *IEEE Trans. Autom. Control.*, *57*(11), 2746–2761.

Ramachandran, D., and E. Amir (2007), Bayesian inverse reinforcement learning, in *IJCAI 2007, Proceedings of the 20th International Joint Conference on Artificial Intelligence, Hyderabad, India, January 6-12, 2007*, edited by M. M. Veloso, pp. 2586–2591.

Ranchod, P., B. Rosman, and G. D. Konidaris (2015), Nonparametric bayesian reward segmentation for skill discovery using inverse reinforcement learning, in *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS 2015*, pp. 471–477.

Rasmussen, C. E., and C. K. I. Williams (2005), *Gaussian Processes for Machine Learning*, The MIT Press.

Ratliff, N. D., J. A. Bagnell, and M. Zinkevich (2006), Maximum margin planning, in *Proceedings of the Twenty-Third International Conference on Machine Learning (ICML 2006)*, pp. 729–736.

Ren, A. Z., S. Veer, and A. Majumdar (2020), Generalization guarantees for imitation learning, *arXiv:2008.01913*.

Renganathan, V., I. Shames, and T. H. Summers (2020), Towards integrated perception and motion planning with distributionally robust risk constraints, *IFAC World Congress*.

Richards, S. M., F. Berkenkamp, and A. Krause (2018), The lyapunov neural network: Adaptive stability certification for safe learning of dynamical systems, in *2nd Annual Conference on Robot Learning, CoRL 2018, Zürich, Switzerland, 29-31 October 2018, Proceedings*, *Proceedings of Machine Learning Research*, vol. 87, pp. 466–476, PMLR.

Richter, C., W. Vega-Brown, and N. Roy (2015), Bayesian learning for safe high-speed navigation in unknown environments, in *International Symposium of Robotics Research (ISRR)*, pp. 325–341.

Robey, A., H. Hu, L. Lindemann, H. Zhang, D. V. Dimarogonas, S. Tu, and N. Matni (2020), Learning control barrier functions from expert demonstrations.

Ross, S., G. J. Gordon, and D. Bagnell (2011), A reduction of imitation learning and structured prediction to no-regret online learning, in *AISTATS*, pp. 627–635.

Russell, S. J., and P. Norvig (2003), *Artificial Intelligence: A Modern Approach*.

Rusu, R. B., and S. Cousins (2011), 3d is here: Point cloud library (PCL), in *IEEE International Conference on Robotics and Automation, ICRA 2011*, IEEE.

Rutledge, K. J., G. Chou, and N. Ozay (2021), Compositional safety rules for inter-triggering hybrid automata, in *HSCC '21: 24th ACM International Conference on Hybrid Systems: Computation and Control, Nashville, Tennessee, May 19-21, 2021*, pp. 4:1–4:11, ACM.

Sabatino, F. (2015), Quadrotor control: modeling, nonlinearcontrol design, and simulation.

Sadigh, D., A. D. Dragan, S. Sastry, and S. A. Seshia (2017), Active preference-based learning of reward functions, in *Robotics: Science and Systems XIII*.

Sadraddini, S., and R. Tedrake (2020), Robust output feedback control with guaranteed constraint satisfaction, in *HSCC '20: 23rd ACM International Conference on Hybrid Systems: Computation and Control, Sydney, New South Wales, Australia, April 21-24, 2020*, pp. 24:1–24:10, ACM.

Saund, B., and D. Berenson (2018), Motion planning for manipulators in unknown environments with contact sensing uncertainty, in *International Symposium of Experimental Robotics (ISER)*, pp. 461–474.

Saund, B., S. Choudhury, S. Srinivasa, and D. Berenson (2019), The blindfolded robot: A bayesian approach to planning with contact feedback, in *International Symposium of Robotics Research (ISRR)*.

Schreiter, J., D. Nguyen-Tuong, M. Eberts, B. Bischoff, H. Markert, and M. Toussaint (2015), Safe exploration for active learning with gaussian processes, in *European Conference on Machine Learning*.

Schulman, J., et al. (2014), Motion planning with sequential convex optimization and convex collision checking, *Int. J. Robotics Res.*, *33*(9), 1251–1270.

Scobee, D. R. R., and S. S. Sastry (2020), Maximum likelihood constraint inference for inverse reinforcement learning, in *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*, OpenReview.net.

Shah, A., P. Kamath, J. A. Shah, and S. Li (2018), Bayesian inference of temporal task specifications from demonstrations, in *Advances in Neural Information Processing Systems (NeurIPS) 2018*, pp. 3808–3817.

Shah, A., S. Li, and J. Shah (2020), Planning with uncertain specifications (puns), *IEEE Robotics Autom. Lett.*, *5*(2), 3414–3421.

Sharma, A., N. Azizan, and M. Pavone (2021), Sketching curvature for efficient out-of-distribution detection for deep neural networks, in *Uncertainty in Artificial Intelligence (UAI)*, pp. 1958–1967, PMLR.

Shen, Z., J. Liu, Y. He, X. Zhang, R. Xu, H. Yu, and P. Cui (2021), Towards out-of-distribution generalization: A survey, *CoRR*, *abs/2108.13624*.

Shoukry, Y., P. Nuzzo, A. L. Sangiovanni-Vincentelli, S. A. Seshia, G. J. Pappas, and P. Tabuada (2018), SMC: satisfiability modulo convex programming, *Proceedings of the IEEE*, *106*(9), 1655–1679.

Singh, S., M. Chen, S. L. Herbert, C. J. Tomlin, and M. Pavone (2018), Robust tracking with model mismatch for fast and safe planning: An SOS optimization approach, in *Algorithmic Foundations of Robotics XIII, Proceedings of the 13th Workshop on the Algorithmic Foundations of Robotics, WAFR 2018, Mérida, Mexico, December 9-11, 2018*, *Springer Proceedings in Advanced Robotics*, vol. 14, edited by M. Morales, L. Tapia, G. Sánchez-Ante, and S. Hutchinson, pp. 545–564, Springer.

Singh, S., B. Landry, A. Majumdar, J. E. Slotine, and M. Pavone (2019), Robust feedback motion planning via contraction theory.

Singh, S., S. M. Richards, V. Sindhwani, J. E. Slotine, and M. Pavone (2020), Learning stabilizable nonlinear dynamics with contraction-based regularization, *International Journal of Robotics Research (IJRR)*.

Solak, E., R. Murray-Smith, W. E. Leithead, D. J. Leith, and C. E. Rasmussen (2002), Derivative observations in gaussian process models of dynamic systems, in *Neural Information Processing Systems*, pp. 1033–1040.

Somani, A., N. Ye, D. Hsu, and W. S. Lee (2013), DESPOT: online POMDP planning with regularization, in *Advances in Neural Information Processing Systems 26: 27th Annual Conference on Neural Information Processing Systems 2013. Proceedings of a meeting held December 5-8, 2013, Lake Tahoe, Nevada, United States*, edited by C. J. C. Burges, L. Bottou, Z. Ghahramani, and K. Q. Weinberger, pp. 1772–1780.

Stachowiak, T., and T. Okada (2006), A numerical analysis of chaos in the double pendulum, *Chaos, Solitons & Fractals*, *29*(2), 417–422.

Sun, D., S. Jha, and C. Fan (2020), Learning certified control using contraction metric, *Conference on Robot Learning*.

Sunberg, Z. N., and M. J. Kochenderfer (2018), Online algorithms for pomdps with continuous state, action, and observation spaces, in *ICAPS*, pp. 259–263, AAAI Press.

Sutanto, G., I. M. R. Fernández, P. Englert, R. K. Ramachandran, and G. S. Sukhatme (2020), Learning equality constraints for motion planning on manifolds, in *Conference on Robot Learning*.

Tao, T. (2016), *Analysis II*, 3rd ed., Springer.

Tedrake, R. (2009), Lqr-trees: Feedback motion planning on sparse randomized trees, *Robotics: Science and Systems V*.

Thakur, S., H. van Hoof, J. C. G. Higuera, D. Precup, and D. Meger (2019), Uncertainty aware learning from demonstrations in multiple contexts using bayesian neural networks, in *IEEE International Conference on Robotics and Automation, ICRA*.

Todorov, E., T. Erez, and Y. Tassa (2012), Mujoco: A physics engine for model-based control, in *IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS*, pp. 5026–5033.

Tsukamoto, H., and S. Chung (2021), Neural contraction metrics for robust estimation and control: A convex optimization approach, *IEEE Control. Syst. Lett.*, *5*(1), 211–216.

Tumova, J., L. I. R. Castro, S. Karaman, E. Frazzoli, and D. Rus (2013), Minimum-violation LTL planning with conflicting specifications, in *American Control Conference, ACC 2013, Washington, DC, USA, June 17-19, 2013*, pp. 200–205, IEEE.

Turchetta, M., F. Berkenkamp, and A. Krause (2016), Safe exploration in finite markov decision processes with gaussian processes, in *Neural Information Processing Systems*, pp. 4305–4313.

Vaidyanathan, P., R. Ivison, G. Bombara, N. A. DeLateur, R. Weiss, D. Densmore, and C. Belta (2017), Grid-based temporal logic inference, in *56th IEEE Annual Conference on Decision and Control, CDC 2017*, pp. 5354–5359.

van den Berg, J., P. Abbeel, and K. Goldberg (2011), Lqg-mp: Optimized path planning for robots with motion uncertainty and imperfect state information, *International Journal of Robotics Research (IJRR)*, *30*(7), 895–913.

van Willigenburg, L. G., and W. L. D. Koning (1999), Optimal reduced-order compensation of time-varying discrete-time systems with deterministic and white parameters, *Autom.*, *35*(1), 129–138.

Vazquez-Chanlatte, M., S. Jha, A. Tiwari, M. K. Ho, and S. A. Seshia (2018), Learning task specifications from demonstrations, in *Neural Information Processing Systems 2018, NeurIPS 2018*, pp. 5372–5382.

Veer, S., and A. Majumdar (2020), Probably approximately correct vision-based planning using motion primitives, in *Conference on Robot Learning*, vol. 155, pp. 1001–1014, PMLR.

Vitus, M. P., Z. Zhou, and C. J. Tomlin (2016), Stochastic control with uncertain parameters via chance constrained control, *IEEE Transactions on Automatic Control*, *61*(10), 2892–2905.

Wächter, A., and L. T. Biegler (2006), On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming, *Math. Program.*, *106*(1), 25–57.

Wang, A., A. Jasour, and B. C. Williams (2020), Moment state dynamical systems for nonlinear chance-constrained motion planning, *CoRR*, *abs/2003.10379*.

Watter, M., J. T. Springenberg, J. Boedecker, and M. A. Riedmiller (2015), Embed to control: A locally linear latent dynamics model for control from raw images, in *Advances in Neural Information Processing Systems 28: Annual Conference on Neural Information Processing Systems 2015, December 7-12, 2015, Montreal, Quebec, Canada*, pp. 2746–2754.

Weng, T.-W., H. Zhang, P.-Y. Chen, J. Yi, D. Su, Y. Gao, C.-J. Hsieh, and L. Daniel (2018), Evaluating the robustness of neural networks: An extreme value theory approach, *International Conference on Learning Representations (ICLR)*.

Williams, G., P. Drews, B. Goldfain, J. M. Rehg, and E. A. Theodorou (2016), Aggressive driving with model predictive path integral control, in *2016 IEEE International Conference on Robotics and Automation, ICRA 2016, Stockholm, Sweden, May 16-21, 2016*, edited by D. Kragic, A. Bicchi, and A. D. Luca, pp. 1433–1440, IEEE.

Wolff, E. M., U. Topcu, and R. M. Murray (2014), Optimization-based trajectory generation with linear temporal logic specifications, in *2014 IEEE International Conference on Robotics and Automation, ICRA*, pp. 5319–5325.

Wood, G. R., and B. P. Zhang (1996), Estimation of the lipschitz constant of a function, *J. Glob. Optim.*, *8*(1), 91–103.

Wu, A., M. C. Aoi, and J. W. Pillow (2018), Exploiting gradients and hessians in bayesian optimization and bayesian quadrature.

Wulfmeier, M., P. Ondruska, and I. Posner (2016), Maximum entropy deep inverse reinforcement learning, *CoRR*, *abs/1507.04888*.

Wulfmeier, M., D. Rao, D. Z. Wang, P. Ondruska, and I. Posner (2017), Large-scale cost function learning for path planning using deep inverse reinforcement learning, *International Journal of Robotics Research (IJRR)*, *36*, 1073–1087.

Xu, Z., A. J. Nettekoven, A. A. Julius, and U. Topcu (2019), Graph temporal logic inference for classification and identification, in *58th IEEE Conference on Decision and Control, CDC 2019*, pp. 4761–4768, IEEE.

Yan, W., A. Vangipuram, P. Abbeel, and L. Pinto (2020), Learning predictive representations for deformable objects using contrastive estimation, in *4th Conference on Robot Learning, CoRL 2020, 16-18 November 2020, Virtual Event / Cambridge, MA, USA*, *Proceedings of Machine Learning Research*, vol. 155, edited by J. Kober, F. Ramos, and C. J. Tomlin, pp. 564–574, PMLR.

Yang, H., and L. Carlone (2022), Certifiably optimal outlier-robust geometric perception: Semidefinite relaxations and scalable global optimization, *TPAMI*.

Yang, H., J. Shi, and L. Carlone (2021), TEASER: fast and certifiable point cloud registration, *T-RO*, *37*(2), 314–333.

Ye, G., and R. Alterovitz (2011), Demonstration-guided motion planning, in *International Symposium of Robotics Research (ISRR)*.

Zhang, A., R. T. McAllister, R. Calandra, Y. Gal, and S. Levine (2021), Learning invariant representations for reinforcement learning without reconstruction, in *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*, OpenReview.net.

Zhang, J., and K. Cho (2016), Query-efficient imitation learning for end-to-end autonomous driving, *CoRR*, *abs/1605.06450*.

Zhang, M., S. Vikram, L. M. Smith, P. Abbeel, M. J. Johnson, and S. Levine (2019), SOLAR: deep structured representations for model-based reinforcement learning, in *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA*, *Proceedings of Machine Learning Research*, vol. 97, edited by K. Chaudhuri and R. Salakhutdinov, pp. 7444–7453, PMLR.

Zhou, B., H. Zhao, X. Puig, T. Xiao, S. Fidler, A. Barriuso, and A. Torralba (2019), Semantic understanding of scenes through the ADE20K dataset, *Int. J. Comput. Vis.*, *127*(3), 302–321.

Zhou, K., and J. C. Doyle (1998), *Essentials of robust control*.

Zhou, W., and W. Li (2018), Safety-aware apprenticeship learning, in *Computer Aided Verification - 30th International Conference, CAV 2018*, pp. 662–680.

Ziebart, B. D., A. L. Maas, J. A. Bagnell, and A. K. Dey (2008), Maximum entropy inverse reinforcement learning, in *Proceedings of the Twenty-Third AAAI Conference on Artificial Intelligence, AAAI 2008, Chicago, Illinois, USA, July 13-17, 2008*, edited by D. Fox and C. P. Gomes, pp. 1433–1438, AAAI Press.

Zucker, M., N. D. Ratliff, A. D. Dragan, M. Pivtoraiko, M. Klingensmith, C. M. Dellin, J. A. Bagnell, and S. S. Srinivasa (2013), CHOMP: covariant hamiltonian optimization for motion planning, *Int. J. Robotics Res.*, *32*(9-10), 1164–1193.

Åström, K. J., and R. M. Murray (2004), Feedback systems: An introduction for scientists and engineers, *Tech. rep.*