TAMPC: A Controller for Escaping Traps in Novel Environments

Sheng Zhong[1], Zhenyuan Zhang[1], Nima Fazeli[1], and Dmitry Berenson[1]

***Abstract*—We propose an approach to online model adaptation and control in the challenging case of hybrid and discontinuous dynamics where actions may lead to difficult-to-escape "trap" states, under a given controller. We first learn dynamics for a system without traps from a randomly collected training set (since we do not know what traps will be encountered online). These "nominal" dynamics allow us to perform tasks in scenarios where the dynamics matches the training data, but when unexpected traps arise in execution, we must find a way to adapt our dynamics and control strategy and continue attempting the task. Our approach, Trap-Aware Model Predictive Control (TAMPC), is a two-level hierarchical control algorithm that reasons about traps and non-nominal dynamics to decide between goal-seeking and recovery policies. An important requirement of our method is the ability to recognize nominal dynamics even when we encounter data that is out-of-distribution w.r.t the training data. We achieve this by learning a representation for dynamics that exploits invariance in the nominal environment, thus allowing better generalization. We evaluate our method on simulated planar pushing and peg-in-hole as well as real robot peg-in-hole problems against adaptive control, reinforcement learning, trap-handling baselines, where traps arise due to unexpected obstacles that we only observe through contact. Our results show that our method outperforms the baselines on difficult tasks, and is comparable to prior trap-handling methods on easier tasks.**

***Index Terms*—Machine Learning for Robot Control, Reactive and Sensor-Based Control.**

I. INTRODUCTION

In this paper, we study the problem of controlling robots in environments with unforeseen **traps**. Informally, traps are states in which the robot's controller fails to make progress towards its goal and gets "stuck". Traps are common in robotics and can arise due to many factors including geometric constraints imposed by obstacles, frictional locking effects, and nonholonomic dynamics leading to dropped degrees of freedom [1], [2], [3]. In this paper, we consider instances of trap dynamics in planar pushing with walls and peg-in-hole with unmodeled obstructions to the goal.

Developing generalizable algorithms that rapidly adapt to handle the wide variety of traps encountered by robots is important to their deployment in the real-world. Two central challenges in online adaptation to environments with traps are the data-efficiency requirements and the lack of progress towards the goal for actions inside of traps. In this paper, our key insight is that we can address these challenges by explicitly reasoning over different dynamic modes, in particular traps, together with contingent recovery policies, organized as a hierarchical controller. We introduce an online modeling and controls method that balances naive optimism and pessimism when encountering novel dynamics. Our method learns a dynamics representation that infers underlying invariances and exploits it when possible (optimism) while treading carefully to escape and avoid potential traps in non-nominal dynamics (pessimism). Specifically, we:

1) Introduce a novel representation learning approach that effectively generalizes dynamics and show its efficacy for task execution on out-of-distribution data when used with our proposed controller;
2) Introduce Trap-Aware Model Predictive Control (TAMPC), a novel control algorithm that reasons about non-nominal dynamics and traps to reach goals in novel environments with traps;
3) Evaluate our method on real robot and simulated peg-in-hole, and simulated planar pushing tasks with traps where adaptive control and reinforcement learning baselines achieve 0% success rate. These include difficult tasks where trap-handling baselines achieve less than 50% success, while our method achieves at least 60% success on all tasks.

We show that state-of-the-art techniques [4], [5], while capable of adapting to novel dynamics, are insufficient for escaping traps that our approach handles by their explicit consideration. Additionally, our method performs well on tasks that prior trap-handling methods struggle on.

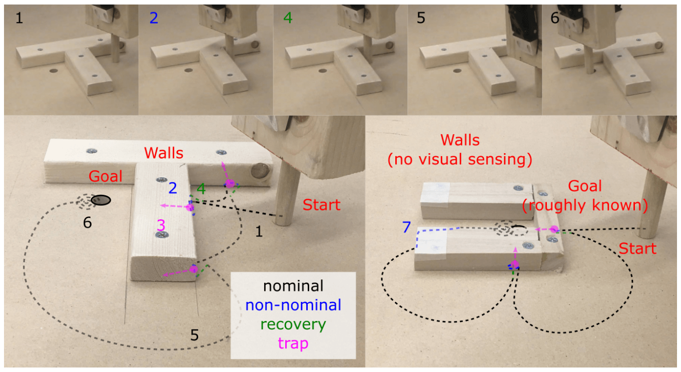

Fig. 1: TAMPC on peg-in-hole tasks with obstacles near the goal. The robot has no visual sensing (cannot anticipate walls) and has not encountered walls in the training data. Path segments show (1) the initial direct trajectory to goal, (2) detecting non-nominal dynamics from reaction forces and exploration of it by sliding along the wall, (3) detecting a trap due to the inability to make progress using non-nominal dynamics, (4) recovery to nominal dynamics, (5) going around seen traps to goal, (6) spiraling to find the precise location of the hole, and (7) sliding against the wall (non-nominal) towards the goal.

Manuscript received: October, 15, 2020; Revised January, 6, 2021; Accepted February, 1, 2021.

This paper was recommended for publication by Editor Dana Kulic upon evaluation of the Associate Editor and Reviewers' comments. This work was supported in part by NSF grant IIS-1750489.

[1]Robotics Institute, University of Michigan, MI 48109, USA {zhsh, cryscan, nfz, dmitryb}@umich.edu

Digital Object Identifier (DOI): see top of this page.

For code, see https://github.com/UM-ARM-Lab/tampc

II. PROBLEM STATEMENT

Let $\mathbf{x} \in \mathcal{X}$ and $\mathbf{u} \in \mathcal{U}$ denote the N_x dimensional state and N_u dimensional control. Under test conditions the system follows novel dynamics $\Delta\mathbf{x} = f_v(\mathbf{x}, \mathbf{u})$. The objective of our method is to reach a goal $\mathbf{x} \in \mathcal{G} \subset \mathcal{X}$ as quickly as possible:

$$\begin{aligned} \underset{\mathbf{u}_0,\dots,\mathbf{u}_{T-1}}{\arg\min} \quad & T \\ \text{s.t.} \quad & \mathbf{x}_{t+1} = \mathbf{x}_t + f_v(\mathbf{x}_t, \mathbf{u}_t),\ t = 0, \dots, T-1 \\ & \mathbf{x}_T \in \mathcal{G} \end{aligned} \tag{1}$$

This problem is difficult because the novel dynamics f_v are not known. Instead, we assume we have access to a dataset of sampled transitions with random actions, which can be used to learn an approximate dynamics model $\hat{f}$ of the system under *nominal* dynamics f where:

$$f_v(\mathbf{x}, \mathbf{u}) = f(\mathbf{x}, \mathbf{u}) + e(\mathbf{x}, \mathbf{u}) \tag{2}$$

We assume the error dynamics e are relatively small (w.r.t. nominal) except for non-nominal regions $\bar{\mathcal{X}} \subset \mathcal{X}$ for which:

$$|e(\mathbf{x}, \mathbf{u})| \sim |f(\mathbf{x}, \mathbf{u})|, \qquad \forall \mathbf{x} \in \bar{\mathcal{X}},\ \exists\, \mathbf{u} \in \mathcal{U} \tag{3}$$

In nominal regions where $\hat{f} \approx f \approx f_v$ a Model Predictive Controller (MPC) can provide high quality solutions without direct access to f_v. Using a specified cost function $C : \mathcal{X} \times \mathcal{U} \to [0, \infty)$, following the MPC policy $\mathbf{u} = \texttt{MPC}(\mathbf{x}, \hat{f}, C)$ creates the dynamical system:

$$\Delta\mathbf{x} = f_v(\mathbf{x}, \texttt{MPC}(\mathbf{x}, \hat{f}, C)) \tag{4}$$

This dynamical system may have attractors [6], which are subsets $A \subseteq \mathcal{X}$ where:

- $\mathbf{x}_{t_0} \in A \implies \mathbf{x}_t \in A \quad \forall t \geq t_0$
- A has a basin of attraction $B(A) \subseteq \mathcal{X} = \{\mathbf{x} \mid \mathbf{x}_0 = \mathbf{x},\ \exists\, t \geq 0,\ \mathbf{x}_t \in A\}$
- A has no non-empty subset with the first two properties

We define a *trap* as an attractor A such that $A \cap \mathcal{G} = \emptyset$, and the *trap set* $\mathcal{T} \subseteq \mathcal{X}$ as the union of all traps. Escaping a trap requires altering the dynamical system (Eq. 4) by altering $\hat{f}$, C, or the MPC function. Traps can be avoided in nominal regions with a sufficiently-powerful and long-horizon MPC, and a sufficiently-accurate dynamics approximation $\hat{f}$. This work addresses detecting, recovering from, and avoiding traps caused by non-nominal dynamics. To aid in trap detection, recovery, and avoidance we assume a state distance function $d : \mathcal{X} \times \mathcal{X} \to [0, \infty)$ and a control similarity function $s : \mathcal{U} \times \mathcal{U} \to [0, 1]$ are given.

III. RELATED WORK

In this section, we review related work to the two main components of this paper: handling traps and generalizing models to out-of-distribution (OOD) dynamics.

Handling Traps: Traps can arise due to many factors including nonholonomic dynamics, frictional locking, and geometric constraints [1], [2], [3]. In particular, they can occur when the environment is partially unknown, as in the case of online path planning.

Traps have been considered in methods based on Artificial Potential Fields (APF) [7], [8]. Traps are local minima in the potential field under the policy of following the gradient of the field. In the case where the environment is only partially known *a priori*, local minima escape (LME) methods such as [8], [9] can be used to first detect then escape local minima. Our method uses similar ideas to virtual obstacles (VO) [8] for detecting traps and avoiding revisits while addressing weaknesses common to APF-LME methods. Specifically, we are able to handle traps near goals by associating actions with trap states (using s) to penalize similar actions to the one previously entering the trap while near it, rather than penalize being near the trap altogether. We also avoid blocking off paths to the goal with virtual obstacles by shrinking their effect over time. Lastly, having an explicit recovery policy and using a controller that plans ahead lets us more efficiently escape traps with "deep" basins (many actions are required to leave the basin). We compare against two APF-LME methods in our experiments.

Another way to handle traps is with exploration, such as through intrinsic curiosity (based on model predictability) [10], [11], state counting [12], or stochastic policies [13]. However, trap dynamics can be difficult to escape from and can require returning to dynamics the model predicts well (so receives little exploration reward). We show in an ablation test how random actions are insufficient for escaping traps in some tasks we consider. Similar to [14], we remember interesting past states. While they design domain-specific state interest scores, we effectively allow for online adaptation of the state score based on how much movement the induced policy generates while inside a trap. We use this score to direct our recovery policy.

Adapting to trap dynamics is another possible approach. Actor-critic methods have been used to control nonlinear systems with unknown dynamics online [15], and we evaluate this approach on our tasks. Another approach is with locally-fitted models which [4] showed could be mixed with a global model and used in MPC. Similar to this approach, our method adapts a nominal model to local dynamics; however, we do not always exploit the dynamics to reach the goal.

Generalizing models to OOD Dynamics: One goal of our method is to generalize the nominal dynamics to OOD novel environments. A popular approach for doing this is explicitly learning to be robust to expected variations across training and test environments. This includes methods such as meta-learning [16], [17], domain randomization [18], [19], Sim-to-real [20], and other transfer learning [21] methods. These methods are unsuitable for this problem because our training data contains only nominal dynamics, whereas they need a diverse set of non-nominal dynamics. Instead, we learn a robust, or "disentangled" representation [22] of the system under which models can generalize. This idea is active in computer vision, where learning based on invariance has become popular [23]. Using similar ideas, we present a novel architecture for learning invariant representations for dynamics models.

IV. METHOD

Our approach is composed of two components: offline representation learning and online control. First, we present how we learn a representation that allows for generalization

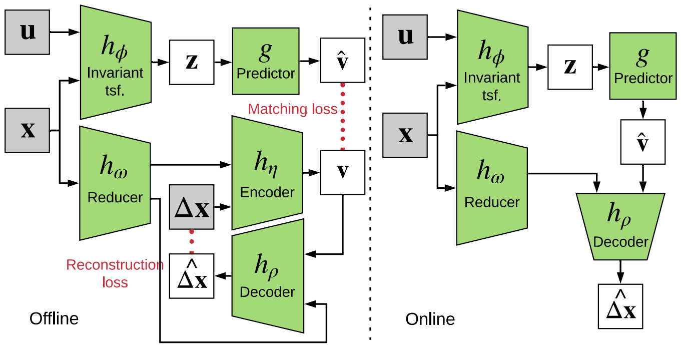

Fig. 2: Architecture for learning (left) and using (right) an invariant representation $\hat{f}$. Grey = given data, green = parameterized transforms, white = computed values, and red dotted lines = losses.

by exploiting inherent invariances inferred from the nominal data, shown in Fig. 2. Second, we present Trap-Aware Model Predictive Control (**TAMPC**), a two-level hierarchical MPC method shown in Fig. 3. The high-level controller explicitly reasons about non-nominal dynamics and traps, deciding when to exploit the dynamics and when to recover to familiar ones by outputting the model and cost function the low-level controller uses to compute control signals.

A. *Offline: Invariant Representation for Dynamics*

In this section, our objective is to learn $\hat{f}$ while exploiting potential underlying invariances in $\mathcal{X} \times \mathcal{U}$ to achieve better generalization to unseen data. More formally, our representation consists of an invariant transform h_ϕ and a predictive module g, shown in Fig. 2. h_ϕ maps $\mathcal{X} \times \mathcal{U}$ to a latent space ($\mathbf{z} \in \mathbf{R}^{N_z}$) that g maps to latent output ($\mathbf{v} \in \mathbf{R}^{N_v}$) that is then mapped back to $\mathcal{X}$ using h_ρ. We parameterize the transforms with neural networks and build in two mechanisms to promote meaningful latent spaces:

First, we impose $N_z < N_x + N_u$ to create an information bottleneck which encourages $\mathbf{z}$ to ignore information not relevant for predicting dynamics. Typically, N_z can be iteratively decreased until predictive performance on $\boldsymbol{\tau}$ drops significantly compared to a model in the original space. Further, we limit the size of g to be smaller than that of h_ϕ such that the dynamics take on a simple form.

Second, to reduce compounding errors when $\mathbf{x}, \mathbf{u}$ is OOD, we partially decouple $\mathbf{z}$ and $\mathbf{v}$ by learning $\mathbf{v}$ in an autoencoder fashion from $\mathbf{x}, \Delta\mathbf{x}$ with encoder h_η and decoder h_ρ. Our innovation is to match the encoded $\mathbf{v}$ with the $\hat{\mathbf{v}}$ output from the dynamics predictor. To further improve generalization, we restrict information passed from $\mathbf{x}$ to $\mathbf{v}$ with a dimension reducing transform $h_\omega : \mathcal{X} \rightarrow \mathbf{R}^{n_\omega}$. These two mechanisms yield the following expressions:

$$\Delta\mathbf{x} \approx \hat{\Delta\mathbf{x}} = h_\rho(\mathbf{v}, h_\omega(\mathbf{x})) \quad \mathbf{v} = h_\eta(\Delta\mathbf{x}, h_\omega(\mathbf{x})) \approx \hat{\mathbf{v}} = g(\mathbf{z})$$

and their associated batch reconstruction and matching loss:

$$\mathcal{L}_r = \frac{\mathbb{E}\,||\Delta\mathbf{x} - h_\rho(\mathbf{v}, h_\omega(\mathbf{x}))||_2}{\mathbb{E}\,||\Delta\mathbf{x}||_2} \quad \mathcal{L}_m = \frac{\mathbb{E}\,||\mathbf{v} - \hat{\mathbf{v}}||_2}{\mathbb{E}\,||\mathbf{v}||_2}$$

$$\mathcal{L}_b(\boldsymbol{\tau}_i) = \lambda_r \mathcal{L}_r(\boldsymbol{\tau}_i) + \lambda_m \mathcal{L}_m(\boldsymbol{\tau}_i)$$

These losses are ratios relative to the norm of the quantity we are trying to match to avoid decreasing loss by scaling the representation. In addition to these two losses, we apply Variance Risk Extrapolation (V-REx [23]) to explicitly penalize the variance in loss across the M trajectories:

$$\mathcal{L}(\boldsymbol{\tau}) = \beta\,\mathbf{var}\,\{\mathcal{L}_b(\boldsymbol{\tau}_1), ..., \mathcal{L}_b(\boldsymbol{\tau}_M)\} + \sum_{i=1}^{M} \mathcal{L}_b(\boldsymbol{\tau}_i) \tag{5}$$

We train on Eq. (5) using gradient descent.

After learning the transforms, we replace g with a higher capacity model and fine-tune it on the nominal data with just $\mathcal{L}_m$. For further details, please see App. C. Since we have no access to $\Delta\mathbf{x}$ online, we pass $\hat{\mathbf{v}}$ to h_ρ instead of $\mathbf{v}$:

$$\hat{f}(\mathbf{x}, \mathbf{u}) = h_\rho(g(h_\phi(\mathbf{x}, \mathbf{u})), h_\omega(\mathbf{x})) \tag{6}$$

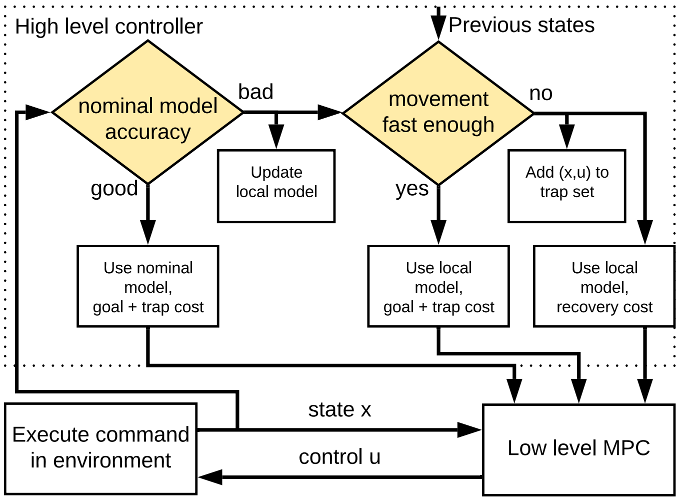

Fig. 3: High-level control architecture.

B. *Online: Trap-Aware MPC*

Online, we require a controller that has two important properties. First, it should incorporate strategies to escape from and avoid detected traps. Second, it should iteratively improve its dynamics representation, in particular when encountering previously unseen modes. To address these challenges, our approach uses a two-level hierarchical controller where the high-level controller is described in Alg. 1.

TAMPC operates in either an exploit or recovery mode, depending on dynamics encountered. When in nominal dynamics, the algorithm exploits its confidence in predictions. When encountering non-nominal dynamics, it attempts to exploit a local approximation built online until it detects entering a trap, at which time recovery is triggered. This approach attempts to balance between a potentially overly-conservative treatment of all non-nominal dynamics as traps and an overly-optimistic approach of exploiting all non-nominal dynamics assuming goal progress is always possible.

The first step in striking this balance is identifying non-nominal dynamics. Here, we evaluate the nominal model prediction error against observed states ("nominal model accuracy" block in Fig. 3 and lines 5 and 13 from Alg. 1):

$$||(\Delta\mathbf{x} - \hat{f}(\mathbf{x}, \mathbf{u}))/E||_2 > \epsilon_N \tag{7}$$

where ϵ_N is a designed tolerance threshold and E is the expected model error per dimension computed on the training data. To handle jitter, we consider transitions from N_n consecutive time steps.

Algorithm 1: TAMPC high-level control loop

```
Given : $C(\mathbf{x},\mathbf{u})$ cost, $\mathbf{x}_0$, MPC, parameters
        from Tab. III
1  $s \leftarrow$ NOM, $t \leftarrow 0$, $d_0 \leftarrow 0$, $\mathbf{x} \leftarrow \mathbf{x}_0$, $\mathcal{T}_c \leftarrow \{\}$,
     $\alpha \leftarrow 1$, $w \leftarrow \mathbf{0}$
2  MAB arms $\leftarrow N_a$ random convex combs.
3  while $C(\boldsymbol{x}, 0) >$ acceptable threshold do
4      if $s$ is NOM then
5          if not nominal from Eq. 7 then
6              $s \leftarrow$ NONNOM
7              initialize GP $\hat{e}$ with $(\mathbf{x}_{t-1}, \mathbf{u}_{t-1}, \Delta\mathbf{x}_{t-1})$
8          else
9              $\alpha \leftarrow \alpha \cdot \alpha_a$ // anneal
10             $d_0 \leftarrow \max(d_0, d(\mathbf{x}_{t-N_d}, \mathbf{x})/N_d)$
11     else
12         fit $\hat{e}$ to include $(\mathbf{x}_{t-1}, \mathbf{u}_{t-1}, \Delta\mathbf{x}_{t-1})$
13         $n \leftarrow$ was nominal last $N_n$ steps // Eq. 7
14         if EnteringTrap($d_0$) then
15             $s \leftarrow$ REC
16             expand $\mathcal{T}_c$ according to Eq. 10
17         if $s$ is REC then
18             if $n$ or Recovered($d_0$) then
19                 $s \leftarrow$ NONNOM
20                 $\alpha \leftarrow$ normalize $\alpha$ so $|C_T| \sim |C|$
21             else if $N_m$ steps since last arm pull then
22                 reward last arm pulled with
                     $d(\mathbf{x}_t, \mathbf{x}_{t-N_m})/N_m d_0$
23                 $w \leftarrow$ Thompson sample an arm
24         if $n$ then
25             $s \leftarrow$ NOM
26     MPC.model $\leftarrow \hat{f}$ if s is NOM else $\hat{f} + \hat{e}$
27     MPC.cost $\leftarrow \rho \cdot (w_1 C_R(\mathcal{X}_0) + w_2 C_R(\mathcal{X}_f))$ if $s$ is
         REC else $C + \alpha C_T$ // Eq. 12 and 11
28     $\mathbf{u} \leftarrow$ MPC($\mathbf{x}$), $t \leftarrow t + 1$
29     $\mathbf{x} \leftarrow$ apply $\mathbf{u}$ and observe from env
```

When in non-nominal dynamics, the controller needs to differentiate between dynamics it can navigate to reach the goal vs. traps and adapt its dynamics models accordingly. Similar to [8], we detect this as when we are unable to make progress towards the goal by considering the latest window of states. To be robust to cost function shape, we consider the state distance instead of cost. Specifically, we monitor the maximum one-step state distance $d(\mathbf{x}_t, \mathbf{x}_{t-1})$ in nominal dynamics, d_0, and compare it against the average state distance to recent states: $d(\mathbf{x}_t, \mathbf{x}_a)/(t-a) < \epsilon_T d_0$ (depicted in "movement fast enough" block of Fig. 3). Here, t is the time from task start. We consider $a = t_0, ..., t - N_d$, where t_0 is the start of non-nominal dynamics or the end of last recovery, whichever is more recent. We ensure state distances are measured over windows of at least size N_d to handle jitter. ϵ_T is how much slower the controller tolerates moving in non-nominal dynamics. For more details see Alg. 2.

Our model adaptation strategy, for both non-nominal dynamics and traps, is to mix the nominal model with an online fitted local model. Rather than the linear models considered in prior work [4], we add an estimate of the error dynamics $\hat{e}$ represented as a Gaussian Process (GP) to the output of the nominal model. Using a GP provides a sample-efficient model that captures non-linear dynamics. To mitigate over-generalizing local non-nominal dynamics to where nominal dynamics holds, we fit it to only the last N_e points since entering non-nominal dynamics. We also avoid over-generalizing the invariance that holds in nominal dynamics by constructing the GP model in the original state-control space. Our total dynamics is then

$$\hat{f}_v(\mathbf{x},\mathbf{u}) = \hat{f}(\mathbf{x},\mathbf{u}) + \hat{e}(\mathbf{x},\mathbf{u}) \tag{8}$$

$$f_v(\mathbf{x},\mathbf{u}) \approx \mathbb{E}[\hat{f}_v(\mathbf{x},\mathbf{u})] \tag{9}$$

When exploiting dynamics to navigate towards the goal, we regularize the goal-directed cost C with a trap set cost C_T to avoid previously seen traps (line 27 from Alg. 1). This trap set $\mathcal{T}_c$ is expanded whenever we detect entering a trap. We add to it the transition with the lowest ratio of actual to expected movement (from one-step prediction, $\hat{\mathbf{x}}$) since the end of last recovery:

$$b = \arg\min_a \frac{d(\mathbf{x}_a, \mathbf{x}_{a+1})}{d(\mathbf{x}_a, \hat{\mathbf{x}}_{a+1})}, \mathcal{T}_c \leftarrow \mathcal{T}_c \cup \{(\mathbf{x}_b, \mathbf{u}_b)\} \tag{10}$$

To handle traps close to the goal, we only penalize revisiting trap states if similar actions are to be taken. With the control similarity function $s : \mathcal{U} \times \mathcal{U} \to [0, 1]$ we formulate the cost, similar to the virtual obstacles of [8]:

$$C_T(\mathbf{x},\mathbf{u}) = \sum_{\mathbf{x}',\mathbf{u}' \in \mathcal{T}_c} \frac{s(\mathbf{u},\mathbf{u}')}{d(\mathbf{x},\mathbf{x}')^2} \tag{11}$$

The costs are combined as $C + \alpha C_T$ (line 27 from Alg. 1), where $\alpha \in (0, \infty)$ is annealed by $\alpha_a \in (0, 1)$ each step in nominal dynamics (line 9 from Alg. 1). Annealing the cost avoids assigning a fixed radius to the traps, which if small results in inefficiency as we encounter many adjacent traps, and if large results in removing many paths to the goal set.

We switch from exploit to recovery mode when detecting a trap, but it is not obvious what the recovery policy should be. Driven by the online setting and our objective of data-efficiency: First, we restrict the recovery policy to be one induced by running the low-level MPC on some cost function other than one used in exploit mode. Second, we propose hypothesis cost functions and consider only convex combinations of them. Without domain knowledge, one hypothesis is to return to one of the last visited nominal states. However, the dynamics may not always allow this. Another hypothesis is to return to a state that allowed for the most one-step movement. Both of these are implemented in terms of the following cost, where S is a state set and we pass in either $\mathcal{X}_0$, the set of last visited nominal states, or $\mathcal{X}_f$, the set of N_f states that allowed for the most single step movement since entering non-nominal dynamics:

$$C_R(\mathbf{x},\mathbf{u},S) = \min_{\mathbf{x}' \in S} d(\mathbf{x},\mathbf{x}')^2 \tag{12}$$

Third, we formulate learning the recovery policy online as a non-stationary multi-arm bandit (MAB) problem. We initialize N_a bandit arms, each a random convex combination of our hypothesis cost functions. Every N_m steps in recovery mode, we pull an arm to select and execute a recovery policy. After

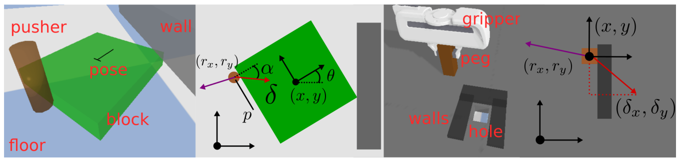

Fig. 4: Annotated simulation environments of (left) planar pushing, and (right) peg-in-hole.

executing N_m control steps, we update that arm's estimated value with a movement reward: $d(\mathbf{x}_t, \mathbf{x}_{t-N_m})/N_m d_0$. When in a trap, we assume any movement is good, even away from the goal. The normalization makes tuning easier across environments. To accelerate learning, we exploit the correlation between arms, calculated as the cosine similarity between the cost weights. Our formulation fits the problem from [24] and we implement their framework for non-stationary correlated multi-arm bandits.

Finally, we return to exploit mode after a fixed number of steps N_R, if we returned to nominal dynamics, or if we stopped moving after leaving the initial trap state. For details see Alg. 3.

V. EXPERIMENTS

In this section, we first evaluate our dynamics representation learning approach, in particular how well it generalizes out-of-distribution. Second, we compare TAMPC against baselines on tasks with traps in two environments.

A. *Experiment Environments*

Our two tasks are quasi-static planar pushing and peg-in-hole. Both tasks are evaluated in simulation using PyBullet [25] and the latter is additionally evaluated empirically using a 7DoF KUKA LBR iiwa arm depicted in Fig. 1. Our simulation time step is 1/240s, however each control step waits until reaction forces are stable. The action size for each control step is described in App B. In planar pushing, the goal is to push a block to a known desired position. In peg-in-hole, the goal is to place the peg into a hole with approximately known location. In both environments, the robot has access to its own pose and senses the reaction force at the end-effector. **Thus the robot cannot perceive the obstacle geometry visually, it only perceives contact through reaction force. During offline learning of nominal dynamics, there are no obstacles or traps**. During online task completion, obstacles are introduced in the environment, inducing unforeseen traps. See Fig. 4 for a depiction of the environments and Fig. 6 for typical traps from tasks in these environments, and App. B for environment details.

In planar pushing, the robot controls a cylindrical pusher restricted to push a square block from a fixed side. Fig. 6 shows traps introduced by walls. Frictional contact with a wall limits sliding along it and causes most actions to rotate the block into the wall. State is $\mathbf{x} = (x, y, \theta, r_x, r_y)$ where (x, y, θ) is the block pose, and (r_x, r_y) is the reaction force the pusher feels, both in world frame. Control is $\mathbf{u} = (p, \delta, \alpha)$, where p is where along the side to push, δ is the push distance, and α is the push direction relative to side normal. The state

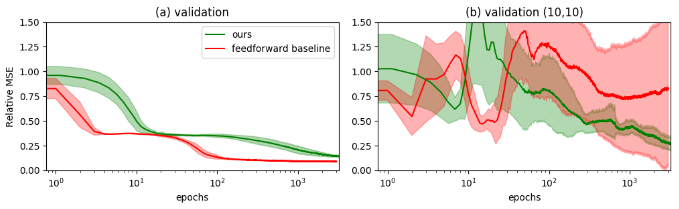

Fig. 5: Learning curves on validation and OOD data sets for planar pushing representation. Mean across 10 runs is solid while 1 std. is shaded.

distance is the 2-norm of the pose, with yaw normalized by the block's radius of gyration. The control similarity is $s(\mathbf{u}_1, \mathbf{u}_2) = \max(0, \text{cossim}((p_1, \alpha_1), (p_2, \alpha_2)))$ where cossim is cosine similarity.

In peg-in-hole, we control a gripper holding a peg (square in simulation and circular on the real robot) that is constrained to slide along a surface. Traps in this environment geometrically block the shortest path to the hole. The state is $\mathbf{x} = (x, y, z, r_x, r_y)$ and control is $\mathbf{u} = (\delta_x, \delta_y)$, the distance to move in x and y. We execute these on the real robot using a Cartesian impedance controller. The state distance is the 2-norm of the position and the control similarity is $s(\mathbf{u}_1, \mathbf{u}_2) = \max(0, \text{cossim}(\mathbf{u}_1, \mathbf{u}_2))$. The goal-directed cost for both environments is in the form $C(\mathbf{x}, \mathbf{u}) = \mathbf{x}^T Q \mathbf{x} + \mathbf{u}^T R \mathbf{u}$. The MPC assigns a terminal multiplier of 50 at the end of the horizon on the state cost. See Tab. IV for the cost parameters of each environment.

$\boldsymbol{\tau}$ for simulated environments consists of $M = 200$ trajectories with $T = 50$ transitions (all collision-free). For the real robot, we use $M = 20$, $T = 30$. We generate them by uniform randomly sampling starts from $[-1, 1] \times [-1, 1]$ (θ for planar pushing is also uniformly sampled; for the real robot we start each randomly inside the robot workspace) and applying actions uniformly sampled from $\mathcal{U}$.

B. *Offline: Learning Invariant Representations*

In this section we evaluate if our representation can learn useful invariances from offline training on $\boldsymbol{\tau}$. We expect nominal dynamics in freespace in our environments to be invariant to translation. Since $\boldsymbol{\tau}$ has positions around $[-1, 1] \times [-1, 1]$, we evaluate translational invariance translating the validation set by $(10, 10)$. We evaluate relative MSE $||\hat{\Delta \mathbf{x}} - \Delta \mathbf{x}||_2 / \mathbb{E}\, ||\Delta \mathbf{x}||_2$ (we do not directly optimize this) against a fully connected baseline of comparable size mapping $\mathbf{x}, \mathbf{u}$ to $\hat{\Delta \mathbf{x}}$ learned on relative MSE. As Fig. 5b shows, our performance on the translated set is better than the baseline, and trends toward the original set's performance. Note that we expect our method to have higher validation MSE since V-REx sacrifices in-distribution loss for lower variance across trajectories. We use $N_z = 5, N_v = 5, n_\omega = 2$, and implement the transforms with fully connected networks. For network sizes and learning details see App. C.

C. *Online: Tasks in Novel Environments*

We evaluate TAMPC against baselines and ablations on the tasks shown in Fig. 1 and Fig. 6. For TAMPC's low-level MPC, we use a modified model predictive path integral

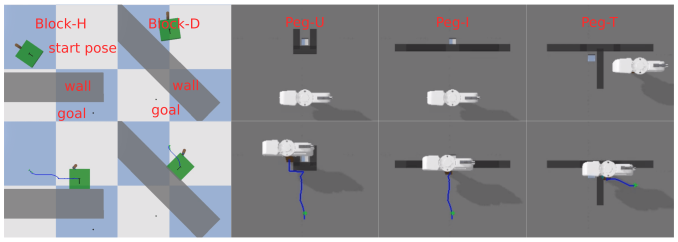

Fig. 6: (top) Initial condition and (bottom) typical traps for planar pushing and peg-in-hole tasks. Our method has no visual sensing and is pre-trained only on environments with no obstacles.

TABLE I: Success counts across all tasks, as defined by achieving distance below the thresholds in Fig. 7.

Method	Success over 10 trials							
	B-H	B-D	P-U	P-I	P-T	P-T(T)	RP-T	RP-U
TAMPC	8	9	7	7	9	6	10	6
TAMPC e=0	6	1	5	0	9	7	9	3
TAMPC rand. rec.	1	4	5	0	9	7	-	-
APF-VO	0	1	4	10	10	10	8	2
APF-SP	1	0	5	10	10	8	10	4
adaptive MPC++	0	0	0	0	0	0	0	0
non-adaptive	0	0	0	0	0	0	0	0
SAC	0	0	0	0	0	0	-	-

controller (MPPI) [26] where we take the expected cost across $R = 10$ rollouts for each control trajectory sample to account for stochastic dynamics. See Alg. 4 for our modifications to MPPI. TAMPC takes less than 1s to compute each control step for all tasks. We run for 500 steps (300 for Real Peg-T).

Baselines: We compare against five baselines. The first is the APF-VO method from [8], which uses the gradient on an APF to select actions. The potential field is populated with repulsive balls in $\mathcal{X}$ based on d as we encounter local minima. We estimate the gradient by sampling 5000 single-step actions and feeding through $\hat{f}$. Second is an APF-LME method from [9] (APF-SP) using switched potentials between the attractive global potential, and a helicoid obstacle potential, suitable for 2D obstacles. The APF methods use the $\hat{f}$ learned with our proposed method (Section IV-A) for next-state prediction. Third is online adaptive MPC from [4] ("adaptive MPC++"), which does MPC on a linearized global model mixed with a locally-fitted linear model. iLQR (code provided by [4]'s author) performs poorly in freespace of the planar pushing environment. We instead use MPPI with a locally-fitted GP model (effectively an ablation of TAMPC with control mode fixed to NONNOM). Next is model-free reinforcement learning with Soft Actor-Critic (SAC) [5]. Here, a nominal policy is learned offline for 1000000 steps on the nominal environment, which is used to initialize the policy at test time. Online, the policy is retrained after every control on the dense environment reward. Lastly, our "non-adaptive" baseline runs MPPI on the nominal model.

We also evaluated ablations to demonstrate the value of TAMPC components. "TAMPC rand. rec." takes uniform random actions until dynamics is nominal instead of using our recovery policy. "TAMPC original space" uses a dynamics model learned in the original $\mathcal{X} \times \mathcal{U}$ (only for Peg-T(T)). Lastly, "TAMPC $e = 0$" does not estimate error dynamics.

D. Task Performance Analysis

Fig. 7 and Tab. I summarizes the results. For Fig. 7, ideal controller performance would approach the lower left corners. We see that TAMPC outperforms all baselines on the block pushing tasks and Peg-U, with slightly worse performance on the other peg tasks compared to APF baselines. APF baselines struggled with block pushing since turning when the goal is behind the block is also a trap for APF methods, because any action increases immediate cost and they do not plan ahead. On the real robot, joint limits were handled naturally as traps by TAMPC and APF-VO.

Note that the APF baselines were tuned to each task, thus it is fair to compare against tuning TAMPC to each task. However, we highlight that our method is empirically robust to parameter value choices, as we achieve high success even when using the same parameter values across tasks and environments, listed in Tab. III. Peg-U and Peg-I were difficult tasks that benefited from independently tuning *only three* important parameters, which we give intuition for: We control the exploration of non-nominal dynamics with N_d. For cases like Peg-U where the goal is surrounded by non-nominal dynamics, we increase exploration by increasing N_d with the trade-off of staying longer in traps. Independently, we control the expected trap basin depth (steps required to escape traps) with the MPC horizon H. Intuitively, we increase H to match deeper basins, as in Peg-I, at the cost of more computation. Lastly, $\alpha_a \in (0, 1)$ controls the trap cost annealing rate. Too low a value prevents escape from difficult traps while values close to 1 leads to waiting longer in cost function local minima. We used $N_d = 15, H = 15$ for Peg-U, and $H = 20, \alpha_a = 0.95$ for Peg-I.

For APF-VO, Peg-U was difficult as the narrow top of the U could be blocked off if the square peg caught on a corner at the entrance. In these cases, TAMPC was able to revisit close to the trap state by applying dissimilar actions to before. This was less an issue in Real Peg-U as we used a round peg, but a different complicating factor is that the walls are thinner (compared to simulation) relative to single-step action size. This meant that virtual obstacles were placed even closer to the goal. APF-SP often oscillated in Peg-U due to traps on either side of the U while inside it.

The non-TAMPC and non-APF baselines tend to cluster around the top left corner in Fig. 7, indicating that they entered a trap quickly and never escaped. Indeed, we see that they all never succeed. For adaptive MPC, this may be due to a combination of insufficient knowledge of dynamics around traps, over-generalization of trap dynamics, and using too short of a MPC horizon. SAC likely stays inside of traps because no action immediately decreases cost, and action sequences that eventually decrease cost have high short-term cost and are thus unlikely to be sampled in 500 steps.

E. Ablation Studies

Pushing task performance degraded on both TAMPC rand. rec. and TAMPC $e = 0$, suggesting value from both the recovery policy and local error estimation. This is likely because trap escape requires long action sequences exploiting

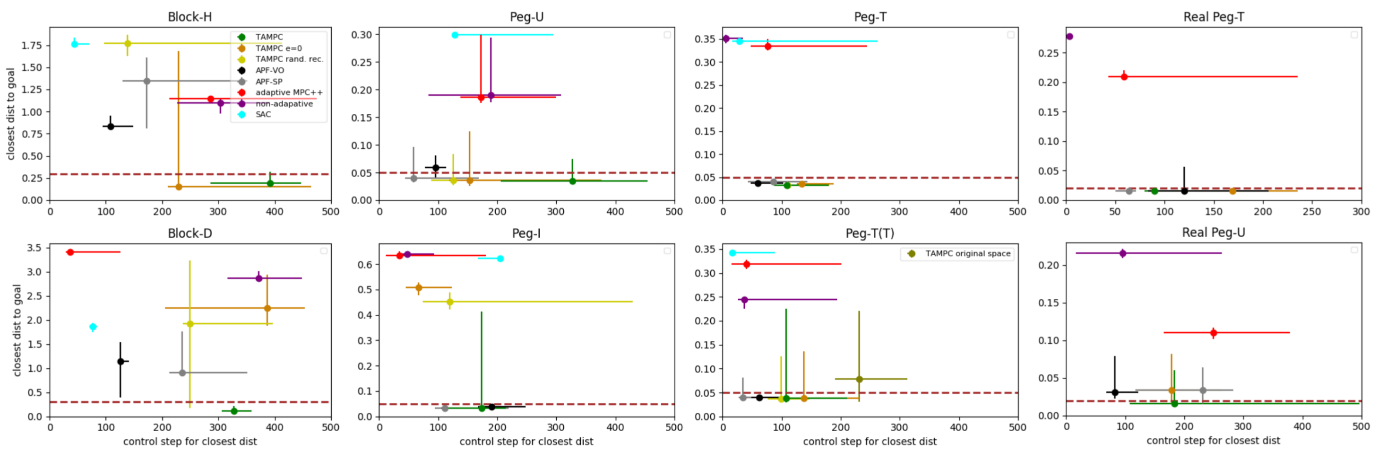

Fig. 7: Minimum distance to goal after 500 steps (300 for Real Peg-T) accounting for walls (computed with Djikstra's algorithm). Median across 10 runs is plotted with error bars representing 20–80th percentile. Task success is achieving lower distance than the dotted red line.

the local non-nominal dynamics to rotate the block against the wall. This is unlike the peg environment where the gripper can directly move away from the wall and where TAMPC rand. rec. performs well. Note that ablations used the parameter values in Tab. III for Peg-I and Peg-U, instead of the tuned parameters from Section V-D. This may explain their decreased performance on them. The Peg-T(T) task (copy of Peg-T translated 10 units in x, y) highlights our learned dynamics representation. Using our representation, we maintain closer performance to Peg-T than TAMPC original space (3 successes). This is because annealing the trap set cost requires being in recognized nominal dynamics, without which it is easy to get stuck in local minima.

VI. Conclusion

We presented TAMPC, a controller that escapes traps in novel environments, and showed that it performs well on a variety of tasks with traps in simulation and on a real robot. Specifically, it is capable of handling cases where traps are close to the goal, and when the system dynamics require many control steps to escape traps. In contrast, we showed that trap-handling baselines struggle in these scenarios. Additionally, we presented and validated a novel approach to learning an invariant representation. Through the ablation studies, we demonstrated the individual value of TAMPC components: learning an invariant representation of dynamics to generalize to out-of-distribution data, estimating error dynamics online with a local model, and executing trap recovery with a multi-arm-bandit based policy. Finally, the failure of adaptive control and reinforcement learning baselines on our tasks suggests that it is beneficial to explicitly consider traps. Future work will explore higher-dimensional problems where the state space could be computationally challenging for the local GP model.

Appendix A: Parameters and Algorithm Details

Making U-turns in planar pushing requires $H \geq 25$. We shorten the horizon to 5 and remove the terminal state cost of 50 when in recovery mode to encourage immediate progress.

$\rho \in (0, \infty)$ depends on how accurately we need to model trap dynamics to escape them. Increasing ρ leads to selecting only the best sampled action while a lower value leads to more exploration by taking sub-optimal actions.

Nominal error tolerance ϵ_N depends on the variance of the model prediction error in nominal dynamics. A higher variance requires a higher ϵ_N. We use a higher value for peg-in-hole because of simulation quirks in measuring reaction force from the two fingers gripping the peg.

TABLE II: MPPI parameters for different environments.

Parameter	block	peg	real peg
K samples	500	500	500
H horizon	40	10	15
R rollouts	10	10	10
λ	0.01	0.01	0.01
$\mathbf{u}'$	$[0, 0.5, 0]$	$[0, 0]$	$[0, 0]$
μ	$[0, 0.1, 0]$	$[0, 0]$	$[0, 0]$
Σ	**diag** $[0.2, 0.4, 0.7]$	**diag** $[0.2, 0.2]$	**diag** $[0.2, 0.2]$

TABLE III: TAMPC parameters across environments.

Parameter	block	peg	real peg
α_a trap cost annealing rate	0.97	0.9	0.8
ρ recovery cost weight	2000	1	1
ϵ_N nominal error tolerance	8.77	12.3	15
ϵ_T trap tolerance	0.6	0.6	0.6
N_d min dynamics window	5	5	2
N_n nominal window	3	3	3
N_m steps for bandit arm pulls	3	3	3
N_a number of bandit arms	100	100	100
N_R max steps for recovery	20	20	20
N_e local model window	50	50	50
ϵ_c converged threshold	$0.05\epsilon_T$	$0.05\epsilon_T$	$0.05\epsilon_T$
ϵ_m move threshold	1	1	1

TABLE IV: Goal cost parameters for each environment.

Term	block	peg	real peg
Q	**diag** $[10, 10, 0, 0, 0]$	**diag** $[1, 1, 0, 0, 0]$	**diag** $[1, 1, 0, 0, 0]$
R	**diag** $[0.1, 0.1, 0.1]$	**diag** $[1, 1]$	**diag** $[1, 1]$

Algorithm 2: EnteringTrap

```
Given: $t_0$ time since end of last recovery or start of
       local dynamics, whichever is more recent,
       $\mathbf{x}_{t_0}, ..., \mathbf{x}_t$, $d_0$, $N_d$, $\epsilon_T$
1 for $a \leftarrow t_0$ to $t - N_d$ do
2     if $d(\mathbf{x}_t, \mathbf{x}_a)/(t - a) < \epsilon_T d_0$ then
3         return True
4 return False
```

Algorithm 3: `Recovered`

```
Given: $\mathbf{x}_0, ..., \mathbf{x}_t$ since start recovery, $d_0$, parameters
       from Tab.III
1  if $t < N_d$ then
2      return False
3  else if $t > N_R$ then
4      return True
5  converged ← $d(\mathbf{x}_t, \mathbf{x}_{t-N_d})/N_d < \epsilon_c d_0$
6  away ← $d(\mathbf{x}_t, \mathbf{x}_0) > \epsilon_m d_0$
7  return converged and away
```

Algorithm 4: MPC Implementation: multi-rollout MPPI. Differences from MPPI [27] are highlighted.

```
Given: cost, model, $\mathbf{x}$, $\tilde{\mathbf{U}}$, Tab. II parameters
1  $\epsilon \leftarrow \mathcal{N}(\mu, \Sigma)$ // u perturbation for $H$ steps
2  $\mathbf{U}, \epsilon \leftarrow$ clip $\tilde{\mathbf{U}}$ to control bounds
3  $\mathbf{X}_0 \leftarrow \mathbf{x}$
4  $c \leftarrow 0$ // $R \times K$
5  for $r \leftarrow 0$ to $R-1$ do
6      for $t \leftarrow 0$ to $H-1$ do
7          $\mathbf{X}_{t+1} \leftarrow$ model$(\mathbf{X}_t, \mathbf{U}_t)$ // sample rollout
8          $c_{r,t} \leftarrow$ cost$(\mathbf{X}_{t+1}, \mathbf{U}_t)$
9          $c_r \leftarrow c_r + c_{r,t}$
10 $c \leftarrow$ mean $c$ across $R$
11 $\mathbf{U} \leftarrow$ softmax mix perturbations
12 return U
```

Appendix B: Environment details

The planar pusher is a cylinder with radius 0.02m to push a square with side length $a = 0.3$m. We have $\theta \in [-\pi, \pi]$, $p \in [-a/2, a/2]$, $\delta \in [0, a/8]$, and $\alpha \in [-\pi/4, \pi/4]$. All distances are in meters and all angles are in radians. The state distance function is the 2-norm of the pose, where yaw is normalized by the block's radius of gyration. $d(\mathbf{x}_1, \mathbf{x}_2) = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2 + \sqrt{a^2/6}(\theta_1 - \theta_2)^2}$. The sim peg-in-hole peg is square with side length 0.03m, and control is limited to $\delta_x, \delta_y \in [0, 0.03]$. These values are internally normalized so MPPI outputs control in the range $[-1, 1]$.

Appendix C: Representation learning & GP

Each of the transforms is represented by 2 hidden layer multilayer perceptrons (MLP) activated by LeakyReLU and implemented in PyTorch. They each have (16, 32) hidden units except for the simple dynamics g which has (16, 16) hidden units, which is replaced with (32, 32) for fine-tuning. The feedforward baseline has (16, 32, 32, 32, 16, 32) hidden units to have comparable capacity. We optimize for 3000 epochs using Adam with default settings (learning rate 0.001), $\lambda_r = \lambda_m = 1$, and $\beta = 1$. For training with V-REx, we use a batch size of 2048, and a batch size of 500 otherwise. We use the GP implementation of gpytorch with an RBF kernel, zero mean, and independent output dimensions. For the GP, on every transition, we retrain for 15 iterations on the last 50 transitions since entering non-nominal dynamics to only fit non-nominal data.

References

[1] J. Borenstein, G. Granosik, and M. Hansen, "The omnitread serpentine robot: design and field performance," in *Unmanned Ground Vehicle Technology VII*, vol. 5804. SPIE, 2005, pp. 324–332.

[2] I. Fantoni, R. Lozano, and R. Lozano, *Non-linear control for underactuated mechanical systems*, 2002.

[3] D. E. Koditschek, R. J. Full, and M. Buehler, "Mechanical aspects of legged locomotion control," *Arthropod structure & development*, vol. 33, no. 3, pp. 251–272, 2004.

[4] J. Fu, S. Levine, and P. Abbeel, "One-shot learning of manipulation skills with online dynamics adaptation and neural network priors," in *IROS*, 2016.

[5] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, "Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor," in *ICML*, 2018, pp. 1861–1870.

[6] J. Milnor, "On the concept of attractor," in *The theory of chaotic attractors*, 1985, pp. 243–264.

[7] O. Khatib, "Real-time obstacle avoidance for manipulators and mobile robots," in *Autonomous robot vehicles*, 1986, pp. 396–404.

[8] M. C. Lee and M. G. Park, "Artificial potential field based path planning for mobile robots using a virtual obstacle concept," in *AIM*, 2003.

[9] G. Fedele, L. D'Alfonso, F. Chiaravalloti, and G. D'Aquila, "Obstacles avoidance based on switching potential functions," *Journal of Intelligent & Robotic Systems*, vol. 90, no. 3-4, pp. 387–405, 2018.

[10] D. Pathak, P. Agrawal, A. A. Efros, and T. Darrell, "Curiosity-driven exploration by self-supervised prediction," in *CVPR Workshops*, 2017.

[11] Y. Burda, H. Edwards, A. Storkey, and O. Klimov, "Exploration by random network distillation," *arXiv preprint arXiv:1810.12894*, 2018.

[12] M. Bellemare, S. Srinivasan, G. Ostrovski, T. Schaul, D. Saxton, and R. Munos, "Unifying count-based exploration and intrinsic motivation," in *NeurIPS*, 2016, pp. 1471–1479.

[13] I. Osband, C. Blundell, A. Pritzel, and B. Van Roy, "Deep exploration via bootstrapped dqn," in *NeurIPS*, 2016, pp. 4026–4034.

[14] A. Ecoffet, J. Huizinga, J. Lehman, K. O. Stanley, and J. Clune, "Go-explore: a new approach for hard-exploration problems," *arXiv preprint arXiv:1901.10995*, 2019.

[15] T. Dierks and S. Jagannathan, "Online optimal control of affine nonlinear discrete-time systems with unknown internal dynamics by using time-based policy update," *IEEE Trans Neural Netw Learn Syst*, vol. 23, no. 7, pp. 1118–1129, 2012.

[16] C. Finn, P. Abbeel, and S. Levine, "Model-agnostic meta-learning for fast adaptation of deep networks," in *ICML*, 2017.

[17] D. Li, Y. Yang, Y.-Z. Song, and T. M. Hospedales, "Learning to generalize: Meta-learning for domain generalization," in *AAAI*, 2018.

[18] J. Tobin, R. Fong, A. Ray, J. Schneider, W. Zaremba, and P. Abbeel, "Domain randomization for transferring deep neural networks from simulation to the real world," in *IROS*. IEEE, 2017, pp. 23–30.

[19] S. J. Pan, I. W. Tsang, J. T. Kwok, and Q. Yang, "Domain adaptation via transfer component analysis," *IEEE Transactions on Neural Networks*, vol. 22, no. 2, pp. 199–210, 2010.

[20] X. B. Peng, M. Andrychowicz, W. Zaremba, and P. Abbeel, "Sim-to-real transfer of robotic control with dynamics randomization," in *ICRA*. IEEE, 2018, pp. 1–8.

[21] A. Zhang, H. Satija, and J. Pineau, "Decoupling dynamics and reward for transfer learning," *arXiv preprint arXiv:1804.10689*, 2018.

[22] X. Chen, Y. Duan, R. Houthooft, J. Schulman, I. Sutskever, and P. Abbeel, "Infogan: Interpretable representation learning by information maximizing generative adversarial nets," in *NeurIPS*, 2016.

[23] D. Krueger, E. Caballero, J.-H. Jacobsen, A. Zhang, J. Binas, R. L. Priol, and A. Courville, "Out-of-distribution generalization via risk extrapolation (rex)," *arXiv preprint arXiv:2003.00688*, 2020.

[24] D. McConachie and D. Berenson, "Bandit-based model selection for deformable object manipulation," in *Algorithmic Foundations of Robotics XII*, 2020, pp. 704–719.

[25] E. Coumans and Y. Bai, "Pybullet, a python module for physics simulation for games, robotics and machine learning," 2016.

[26] G. Williams, N. Wagener, B. Goldfain, P. Drews, J. M. Rehg, B. Boots, and E. A. Theodorou, "Information theoretic mpc for model-based reinforcement learning," in *ICRA*, 2017.

[27] S. Zhong and T. Power, "Pytorch model predictive path integral controller," https://github.com/LemonPi/pytorch_mppi, 2019.