Obeying Constraints During Motion Planning

Dmitry Berenson

Contents

D. Berenson (✉)
Department of Electrical Engineering and Computer Science, University of Michigan, Ann Arbor, MI, USA
e-mail: berenson@eecs.umich.edu; dberenson@cs.wpi.edu

A. Goswami, P. Vadakkepat (eds.), *Humanoid Robotics: A Reference*,
https://doi.org/10.1007/978-94-007-7194-9_58-1

1 Introduction

Every practical motion planning problem in robotics involves constraints. Whether the robot must avoid collision or joint limits, there are always states that are not permissible. Some constraints are straightforward to satisfy, while others can be so stringent that feasible states are very difficult to find. What makes planning with constraints challenging is that, for many constraints, it is impossible or impractical to provide the planning algorithm with the allowed states explicitly; it must discover these states as it plans. This chapter focuses on constraints relevant to motion planning for humanoids.

Motion planning for humanoid robots gives rise to a rich variety of tasks that include constraints on collision avoidance, torque, balance, closed-chain kinematics, and end-effector pose. Many researchers have developed representations and strategies to plan with these kinds of constraints, and the goal of this chapter is to provide an overview of both the representations of the constraints and the strategies used to enforce them in practical scenarios.

Some of the most important constraints for humanoid robots are functions of the pose of the robot's end-effectors, so a large part of this chapter is devoted to constraints on end-effector pose. However, we also discuss constraints on collision, balance, and closed-chain kinematics. This chapter focuses on the constraint representations and techniques to generate configurations that satisfy the constraints. To see how a sampling-based motion planner can use these kinds of constraint representations and strategies, the reader is referred to [1].

The techniques presented in this chapter can be applied to tasks where the constraints are evaluated as functions of a robot's configuration (not including velocity variables). While dynamic motion planning for humanoids is clearly desirable for many tasks, the planning problem becomes much more difficult, and often only local methods, such as trajectory optimizers, can be used. Planning without dynamics restricts the domain to motions that can be performed arbitrarily slowly and thus excludes domains like dynamic walking. However, quasi-static motion planning is still applicable to many practical tasks, for instance, many manipulation tasks in a domestic or industrial environment. Additionally, a sampling-based approach to planning with the kinds of pose constraints described here has been shown to be probabilistically complete [2].

In the following sections, we first define the types of constraints we consider and then present the constrained path planning problem. Next we discuss three types of methods to generate configurations that satisfy constraints: direct sampling, rejection sampling, and projection sampling. The sections that follow discuss how to represent constraints on collision, pose, closed-chain kinematics, and balance, as well as describing direct, rejection, and projection methods relevant for each constraint. We conclude with several example problems that show how to specify constraints for practical tasks, including results for running a motion planner on these problems.

2 Constraint Definition and Strategies

Depending on the robot and the task, many types of constraints can limit a robot's motion. One of the most common distinctions in the robotics literature is between *holonomic* and *nonholonomic* constraints. A *holonomic* constraint is one that can be expressed as a function of the configuration of the robot q (and possibly time t) and has the form $F(q,t) = 0$. A *nonholonomic* constraint is one that cannot be expressed in this way. It is important to note that the definition of *holonomic* above does not allow inequalities, i.e., it must be a bilateral constraint. This fact implies that constraints that are typically thought of as holonomic in robotics literature such as collision avoidance are in fact nonholonomic. To preserve consistency with the robotics literature, we will relax the definition of holonomic constraints to include inequality constraints for the purposes of this chapter.

Nonholonomic constraints are ones that are impossible to represent as a function of only the configuration of the robot and time. A classical example of such a constraint is the kinematics of the unicycle, which can move forward and back and rotate about the center of the wheel but cannot move sideways. This constraint can be expressed as

$$F(\dot{x}, \dot{y}, \theta) = \dot{x} \sin\theta - \dot{y} \cos\theta. \tag{1}$$

Some constraints that depend on the derivatives of configuration variables can be integrated into holonomic constraints, but the above one cannot; thus it is nonholonomic. This is the reason that nonholonomic constraints are sometimes referred to as *nonintegrable* constraints.

There is also a distinction between constraints with respect to their dependence on time. If a constraint depends on time (among other variables), it is referred to as *rheonomic*; otherwise it is referred to as *scleronomic*.

2.1 Defining Constraints on Configuration

In motion planning for humanoid robots, a common and effective approach is to plan paths in the configuration space of the robot and then track those trajectories with appropriate controllers [3]. Thus this chapter focuses on constraints that are functions of the robot's configuration only, i.e., scleronomic holonomic constraints. After a path in C-space is computed, it is then necessary to retime the path to assign a time index to each configuration, i.e., creating a trajectory, which can then be executed.

Let the configuration space of the robot be $\mathcal{Q}$. A path in that space is defined by $\tau : [0,1] \rightarrow \mathcal{Q}$. We consider constraints evaluated as a function of a configuration $q \in \mathcal{Q}$ in τ. The location of q in τ determines which constraints are active at that configuration. Thus a constraint is defined as the pair $\{C(q), s\}$, where $C(q) \in \mathbb{R} \geq 0$ is the *constraint evaluation function* and $s \subseteq [0,1]$ is the *domain* of the constraint.

$C(q)$ determines whether the constraint is met at that q and s specifies where in the path τ the constraint is active. To say that a given constraint is satisfied, we require that $C(q) = 0 \quad \forall q \in \tau(s)$. For instance, we may require that τ start at a given configuration q_{start}:

$$C(q) = \begin{cases} 0 \text{ if } q = q_{\text{start}} \\ 1 \quad \text{otherwise} \end{cases} \quad \text{for } q = \tau(0) \tag{2}$$

We may also require that τ be collision-free everywhere along the path. The collision-avoidance constraint is then defined as

$$C(q) = \begin{cases} 1 \text{ if InCollision}(q) \\ 0 \qquad \text{otherwise} \end{cases} \quad \forall q \in \tau([0, 1]). \tag{3}$$

Each constraint defined in this way implicitly defines a manifold in $\mathcal{Q}$ where $\tau(s)$ is allowed to exist. Given a constraint, the manifold of configurations that meet this constraint $\mathcal{M}_C \subseteq \mathcal{Q}$ is defined as

$$\mathcal{M}_C = \{q \in \mathcal{Q} : C(q) = 0\}. \tag{4}$$

In order for τ to satisfy a constraint, all the elements of $\tau(s)$ must lie within $\mathcal{M}_C$. If $\exists q \notin \mathcal{M}_C$ for $q \in \tau(s)$, then τ is said to violate the constraint.

In general, we can define any number of constraints for a given task, each with their own domain. Let a set of n constraint-evaluation functions be $\mathcal{C}$ and the set of domains corresponding to those functions be $\mathcal{S}$. Then we define the constrained path planning problem as

$$\begin{aligned} \text{find } \tau : q \in \mathcal{M}_{C_i} \quad & \forall q \in \tau(\mathcal{S}_i) \\ & \forall i \in \{1 \ldots n\}. \end{aligned} \tag{5}$$

Note that the domains of two or more constraints may overlap in which case an element of τ may need to lie within two or more constraint manifolds. For example, to specify a reaching task for a humanoid, we define three pose constraints: The first two pose constraints keep the feet fixed through the entire motion ($\mathcal{S}_1 = [0, 1]$, $\mathcal{S}_2 = [0, 1]$), and the third pose constraint specifies that the arm's end-effector should be at a certain pose at the end of the path ($\mathcal{S}_3 = 1$). Thus all configurations along the path must have the feet fixed, while at the last configuration in the path, the feet must be fixed and the arm's end-effector must be at the goal pose.

2.2 Challenges of Constrained Path Planning for Humanoids

Two main issues make solving the constrained path planning problem difficult for humanoids. First, constraint manifolds are difficult to represent, even for low-DOF

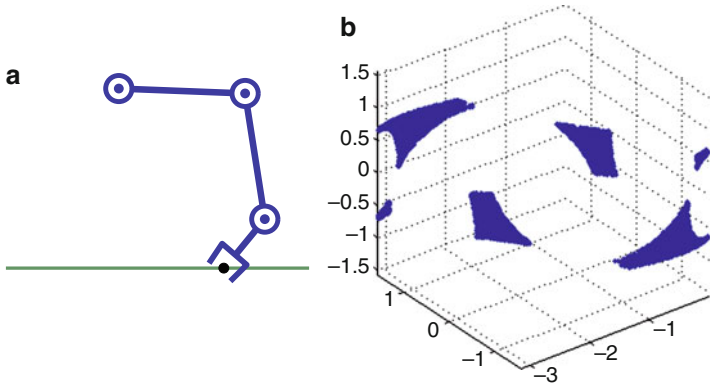

Fig. 1 (**a**) Pose constraint for a three-link manipulator: The end-effector must be on the line with an orientation within ± 0.7 rad of downward. (**b**) The manifold induced by this constraint in the C-space of this robot

robots. There is no known analytical representation for many types of constraint manifolds (including pose constraints), and the high-dimensional C-spaces of humanoids make representing the manifold through exhaustive sampling or discretization prohibitively expensive. It is possible to parameterize some constraint manifolds; however this can be insufficient for planning paths because the mapping from the parameter space to the manifold can be non-smooth (see Fig. 1). Thus, although we can construct a smooth path in the parameter space, its image on the constraint manifold may be disjoint. Restrictions imposed on the mapping to render it smooth, like imposing a one-to-one mapping from pose to configuration, compromise on completeness.

Second, and acutely important for the pose constraints needed for humanoid motion planning, is the fact that constraint manifolds can be of a lower dimension than the ambient C-space. Lower-dimensional manifolds cannot be sampled using rejection sampling (the sampling technique employed by most sampling-based planners), and thus more sophisticated sampling techniques are required. A key challenge is to demonstrate that the distribution of samples produced by these techniques densely covers the constraint manifold, which is necessary for probabilistic completeness.

To overcome these challenges, a planner must be able to generate configurations on (possibly lower-dimensional) constraint manifolds in high-dimensional spaces. Below we discuss three strategies that can be used to generate such configurations through sampling.

2.3 Sampling on Constraint Manifolds

We describe three sampling strategies for generating configurations on constraint manifolds: rejection, projection, and direct sampling (see Fig. 2).

In the *rejection* strategy, we simply generate a random sample $q \in \mathcal{Q}$ and check if $C(q) = 0$; if this is not the case, we deem q invalid. This strategy is effective when there is a high probability of randomly sampling configurations that satisfy this constraint; in other words, $\mathcal{M}_C$ occupies some significant volume in $\mathcal{Q}$. This strategy is used to satisfy torque, balance, and collision constraints, among others.

The *projection* strategy is robust to more stringent constraints, namely, ones whose manifolds do not occupy a significant volume of the C-space. However this robustness comes at the price of requiring a function to evaluate how close a given configuration is to the constraint manifold, i.e., $C(q)$ needs to encode some measure of distance to the manifold. The projection strategy first generates a $q_0 \in \mathcal{Q}$ and then moves that q_0 onto $\mathcal{M}_C$. The most common type of projection operator relevant for our application is an iterative gradient descent process. Starting at q_0, the projection operator iteratively moves the configuration closer to the constraint manifold so that $C(q_{i+1}) < C(q_i)$. This process terminates when the gradient descent reaches a configuration on $\mathcal{M}_C$, i.e., when $C(q_i) = 0$ (numerically, a small threshold is used to determine when the projection terminates). A key advantage of the projection strategy is that it is able to generate valid configurations near other configurations on $\mathcal{M}_C$, which allows us to use it in algorithms based on the RRT [4]. This strategy is used to sample on lower-dimensional constraint manifolds, such as those induced by end-effector pose or closed-chain kinematics constraints.

Finally, the *direct sampling* strategy uses a parameterization of the constraint to generate samples on $\mathcal{M}_C$. This strategy is specific to the constraint representation, and the mapping from the parameterization to $\mathcal{M}_C$ can be arbitrarily complex. Though this strategy can produce valid samples, it can be difficult to generate samples in a desired region of $\mathcal{M}_C$, for instance, generating a sample near other samples (a key requirement for building paths). Thus this strategy is mainly used when sampling goal configurations, not to build paths – e.g., using inverse

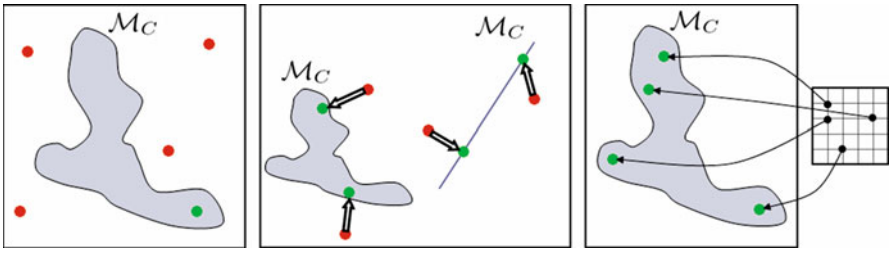

Fig. 2 The three sampling strategies used in our framework. *Red dots* represent invalid samples and *green dots* represent valid ones. (*Left*) rejection sampling, (*center*) projection sampling, (*right*) direct sampling from a parameterization of the constraint

kinematics to determine a goal configuration of the robot given a set of target end-effector poses.

A given constraint may be sampled using one or more of these strategies. The choice of strategy depends on the definition of the constraint as well as the path planning algorithm. Sometimes a mix of strategies may be appropriate. For instance, a PRM [5] planning with pose constraints may use the direct sampling strategy to generate a set of map nodes but may switch to the projection strategy when constructing edges between those nodes.

We now discuss constraints commonly used for humanoid robot motion planning and which strategies are effective for which constraints. This chapter focuses mainly on pose constraints, as they are especially important for humanoid motion.

3 Collision Constraints

Perhaps the most important and widely studied type of constraint in motion planning is the constraint that no link of the robot may collide with any other link (called "self-collision") or an obstacle in the environment. Collision detection has been studied extensively in the context of computer graphics (see [6] for a survey), as well as in robotics. Though 3D shapes can be represented in a variety of ways, in robotics applications, links and obstacles are often represented as either composites of geometric primitives (e.g., cylinders, boxes, and spheres) or triangle meshes. Composites of primitives are advantageous because computing whether two primitives intersect can be done very efficiently. However, composites of primitives usually overestimate the volume of the link, thus resulting in a conservative approximation. For particularly complex links, many primitives may be necessary, which impacts computation time. Triangle meshes are the most expressive representation, but checking for intersection between them can be computationally expensive. Several open-source libraries have been adopted by the robotics community for computing triangle mesh intersections [7, 8].

In many robotics applications, self-collisions and collision with the environment are checked using the same methods. However, because of the large number of humanoid links, specialized methods for humanoid self-collision checking have been proposed. These methods prune the number of collision checks to perform using a table of links that have the potential to collide. This table can be constructed using heuristic or exhaustive search methods [9, 10]. Convex hulls can also be constructed around complex links to increase the speed of self-collision checking [11], which again usually overestimate the geometry of the links.

Some strategies used to generate collision-free configurations require not only a binary check for determining if a link is in collision but also a distance measurement between a link and the nearest obstacle. Collision libraries often include this distance-checking feature (most notably the PQP library [7]), though computing distances between triangle meshes is generally more expensive than checking for collision.

The manifold of collision-free configurations is usually assumed to have the same dimension as the C-space; thus the most common strategy for finding collision-free configurations of a robot is rejection sampling. In this strategy, a configuration generated by a planner is set on the robot using forward kinematics, and a collision query is used to determine if any link of the robot is in collision. This strategy is effective when the manifold of configuration-free configurations has significant volume in the configuration space. However, the task the robot needs to perform may require traversing a part of the C-space where the collision-free manifold narrows, i.e., a narrow passage. In such cases, specialized narrow-passage sampling methods can be used to bias the rejection sampling [12–14]. For humanoids, narrow passages can have significantly different width in different C-space dimensions. For example, one arm of the humanoid may be in a collision-free area, while the other arm must navigate through a narrow opening. In such cases, methods that estimate the variance of samples in different dimensions can be used to create more effective samplers [15].

Researchers have also investigated using projection-based methods to generate configurations that meet collision constraints. In this approach, if a configuration is in collision, local projection methods can be used to retract the configuration to the obstacle boundary. Computing the closest configuration on the obstacle boundary can be expensive for an articulated robot due to the need to compute the C-space penetration depth (though penetration depth can be computed for two free bodies in 3D [16]). Instead, projection to the obstacle boundary can be accomplished locally by using the pseudo-inverse of the Jacobian. The Jacobian is computed for a point p on a link that has penetrated an obstacle, and along with a vector that points out of the obstacle from p, the Jacobian can be used to iteratively "push" the colliding configuration to the obstacle boundary [17, 18].

4 Pose Constraints

The pose of a manipulator's end-effector is represented as a point in $SE(3)$, the six-dimensional space of rigid spatial transformations. Many practical manipulation tasks, like moving a large box or opening a refrigerator door, impose constraints on the motion of a robot's end-effector(s) as well as allowing freedom in the acceptable goal pose of the end-effector. For example, consider a humanoid robot placing a large box onto a table. Although the humanoid's hands are constrained to grasp the box during manipulation, the task of placing the box on the table affords a wide range of box placements and robot configurations that achieve the goal. Sampling-based motion planning for tasks with pose constraints has been explored by several researchers [1, 19–21], with the differences among methods being in the representation of the constraints and the sampling methods used to generate configurations on constraint manifolds. Below we present the task space region (TSR) representation of pose constraints, as it has been shown to be particularly useful for planning manipulation tasks for humanoids [22]. The ability of TSRs to capture constraints relevant to humanoid manipulation is shown in Sect. 7.

4.1 Task Space Regions

TSRs are a straightforward pose constraint representation that can capture many useful tasks. For more complex tasks, we have also developed TSR Chains (Sect. 4.2), which are defined by linking a series of TSRs. TSRs build on a long history of constraint-based problem specification. Seminal theoretical work in this area was done by Ambler and Popplestone [23], who specify geometric constraints between features of two objects and then solve for the pose of a robot which assembles these objects using symbolic methods. The AL system [24, 25] encoded pose constraints on object placement as inequalities of position and rotation variables, which is similar to the bounds of a TSR. The AutoPASS system [26] allowed specifying pose constraints for primitive motions of a manipulator.

More recent work in sampling-based planning involves planning to a goal pose [27, 28] or set of goal poses [29] for the end-effector. The representations used in this work are subsumed by TSRs. Stilman [19] presented a representation for pose constraints on the robot's path, which is also subsumed by TSRs. Finally, a representation similar to TSRs was used by De Schutter et al. [30] for a controller that maintained the pose of a frame on the robot in a set defined relative to a frame in the environment.

Task space regions (TSRs) describe end-effector constraint sets as subsets of $SE(3)$. These subsets are particularly useful for specifying manipulation tasks ranging from reaching to grasp an object and placing it on a surface or in a volume to manipulating objects with constraints on their pose such as transporting a glass of water without spilling or sliding a milk jug on a table.

TSRs are specifically designed to be used with sampling-based planners. As such, it is straightforward to specify TSRs for common tasks, to compute distance from a given pose to a TSR (necessary for the projection strategy), and to sample from a TSR using direct sampling. Furthermore, multiple TSRs can be defined for a given task, which allows the specification of multiple simultaneous constraints and affordances.

TSRs are not intended to capture every conceivable constraint on pose. Instead they are meant to be simple descriptions of common manipulation tasks that are useful for planning. Section 4.2 presents a more complex representation for articulated constraints called TSR Chains.

4.1.1 TSR Definition

Throughout this section, we will be using transformation matrices of the form $\mathbf{T}_b^a$, which specifies the pose of b in the coordinates of frame a. $\mathbf{T}_b^a$, written in homogeneous coordinates, consists of a 3×3 rotation matrix $\mathbf{R}_b^a$ and a 3×1 translation vector $\mathbf{t}_b^a$:

$$\mathbf{T}_b^a = \begin{bmatrix} \mathbf{R}_b^a & \mathbf{t}_b^a \\ \mathbf{0} & 1 \end{bmatrix} \tag{6}$$

A TSR consists of three parts:

- $\mathbf{T}_w^0$: transform from the origin to the TSR frame w
- $\mathbf{T}_e^w$: end-effector offset transform in the coordinates of w
- $\mathbf{B}^w$: 6×2 matrix of bounds in the coordinates of w:

$$\mathbf{B}^w = \begin{bmatrix} x_{min} & x_{max} \\ y_{min} & y_{max} \\ z_{min} & z_{max} \\ \psi_{min} & \psi_{max} \\ \theta_{min} & \theta_{max} \\ \phi_{min} & \phi_{max} \end{bmatrix} \tag{7}$$

The first three rows of $\mathbf{B}^w$ bound the allowable translation along the x-, y-, and z-axes (in meters), and the last three bound the allowable rotation about those axes (in radians), all in the w frame. Note that this assumes the roll-pitch-yaw (RPY) Euler angle convention, which is used because it allows bounds on rotation to be intuitively specified.

In practice, the w frame is usually centered at the origin of an object held by the hand or at a location on an object that is useful for grasping. We use an end-effector offset transform $\mathbf{T}_e^w$, because we do not assume that w directly encodes the pose of the end-effector. $\mathbf{T}_e^w$ allows the user to specify an offset from w to the origin of the end-effector e, which is extremely useful when we wish to specify a TSR for an object held by the hand or a grasping location which is offset from e; for instance, in between the fingers. For some example $\mathbf{T}_e^w$ transforms, see Fig. 3.

4.1.2 Distance to TSRs

When using the projection strategy with TSRs, it will be necessary to find the distance from a given configuration q_s to a TSR (please follow the explanation below in Fig. 4). Because we do not have an analytical representation of the constraint manifold corresponding to a TSR, we compute this distance in task space. Given a q_s, we use forward kinematics to get the position of the end-effector at this configuration $\mathbf{T}_s^0$. We then apply the inverse of the offset $\mathbf{T}_e^w$ to get $\mathbf{T}_{s'}^0$, which is

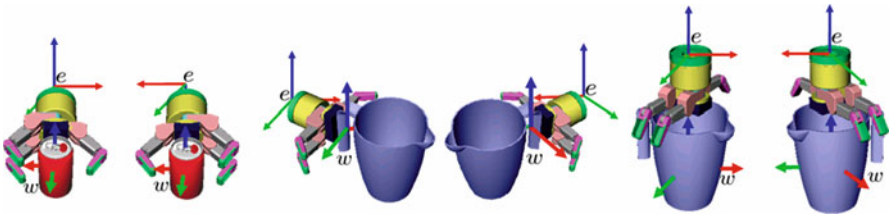

Fig. 3 The w and e frames used to define end-effector goal TSRs for a soda can and a pitcher

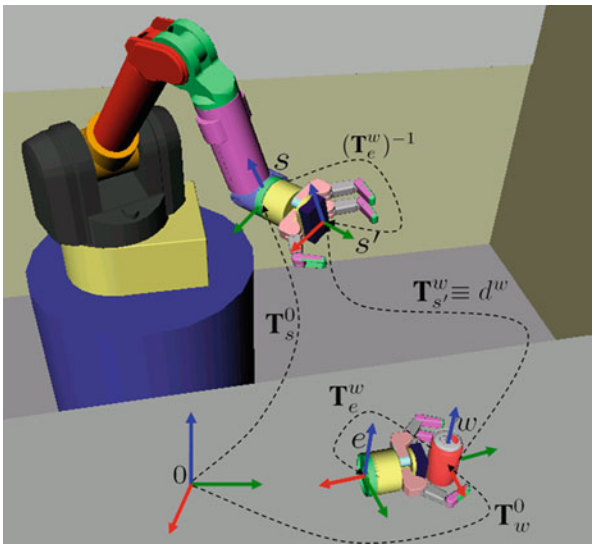

Fig. 4 Transforms and coordinate frames involved in computing the distance to TSRs. The robot is in a sample configuration which has end-effector transform s and the hand near the soda can at transform e represents the $\mathbf{T}_e^w$ defined by the TSR

the pose of the grasp location or the pose of the object held by the hand in world coordinates:

$$\mathbf{T}_{s'}^0 = \mathbf{T}_s^0 (\mathbf{T}_e^w)^{-1} \tag{8}$$

We then convert this pose from world coordinates to the coordinates of w:

$$\mathbf{T}_{s'}^w = (\mathbf{T}_w^0)^{-1} \mathbf{T}_{s'}^0 \tag{9}$$

Now we convert the transform $\mathbf{T}_{s'}^w$ into a 6×1 displacement vector from the origin of the w frame. This displacement represents rotation in the RPY convention so it is consistent with the definition of $\mathbf{B}^w$:

$$d^w = \begin{bmatrix} \mathbf{t}_{s'}^w \\ \arctan 2(\mathbf{R}_{s'_{32}}^w, \mathbf{R}_{s'_{33}}^w) \\ -\arcsin(\mathbf{R}_{s'_{31}}^w) \\ \arctan 2(\mathbf{R}_{s'_{21}}^w, \mathbf{R}_{s'_{11}}^w) \end{bmatrix} \tag{10}$$

Taking into account the bounds of $\mathbf{B}^w$, we get the 6×1 displacement vector to the TSR $\Delta\mathbf{x}$:

$$\Delta\mathbf{x}_i = \begin{cases} d_i^w - \mathbf{B}_{i,1}^w & \text{if } d_i^w < \mathbf{B}_{i,1}^w \\ d_i^w - \mathbf{B}_{i,2}^w & \text{if } d_i^w > \mathbf{B}_{i,2}^w \\ 0 & \text{otherwise} \end{cases} \tag{11}$$

where i indexes through the six rows of $\mathbf{B}^w$ and six elements of $\Delta\mathbf{x}$ and d^w. $\|\Delta\mathbf{x}\|$ is the distance to the TSR. Note that we implicitly weigh rotation in radians and translation in meters equally when computing $\|\Delta\mathbf{x}\|$, but the two types of units can be weighed in an arbitrary way to produce a distance metric that considers one or the other more important. Because of the inherent redundancy of the RPY Euler angle representation, there are several sets of angles that represent the same rotation. To find the minimal distance by our metric, we evaluate the norm of each of the possible RPY angle sets capable of yielding the minimum displacement. This set consists of the $\{\Delta\mathbf{x}_4, \Delta\mathbf{x}_5, \Delta\mathbf{x}_6\}$ defined above as well as the eight equivalent rotations $\{\Delta\mathbf{x}_4 \pm \pi, -\Delta\mathbf{x}_5 \pm \pi, \Delta\mathbf{x}_6 \pm \pi\}$.

If we define multiple TSRs for a given manipulator, we extend our distance computation to evaluate distance to all relevant TSRs and return the smallest.

4.1.3 Direct Sampling of TSRs

When using TSRs to specify goal end-effector poses, it will be necessary to sample poses from TSRs. Sampling from a single TSR is done by first sampling a random value between each of the bounds defined by $\mathbf{B}^w$ with uniform probability. These values are then compiled in a displacement d^w_{sample} and converted into the transformation $\mathbf{T}^w_{\text{sample}}$. We can then convert this sample into world coordinates after applying the end-effector transform:

$$\mathbf{T}^0_{\text{sample}'} = \mathbf{T}^0_w \mathbf{T}^w_{\text{sample}} \mathbf{T}^w_e \tag{12}$$

We observe that while our method ensures a uniform sampling in the bounds of $\mathbf{B}^w$, it could produce a biased sampling in the subspace of constrained spatial displacements $SE(3)$ that $\mathbf{B}^w$ parameterizes. However this bias has not had a significant impact on the runtime or success rate of our algorithms.

In the case of multiple TSRs specified for a single task, we must first decide which TSR to sample from. If the bounds of all TSRs enclose six-dimensional volumes, we can choose among TSRs in proportion to their volume. However a volume-proportional sampling will ignore TSRs that encompass volumes of less than six dimensions because they have no volume in the six-dimensional space. To address this issue, we use a weighted sampling scheme that samples TSRs proportional to the *sum* of the differences between their bounds:

$$\zeta_i = \sum_{j=1}^{6} \left(\mathbf{B}^{w_i}_{j,2} - \mathbf{B}^{w_i}_{j,1} \right) \tag{13}$$

where ζ_i and $\mathbf{B}^{w_i}$ are the weight and bounds of the ith TSR, respectively. Sampling proportional to ζ_i allows us to sample from TSRs of any dimension except 0 while giving preference to TSRs that encompass more volume. TSRs of dimension 0, i.e., points, are given a fixed probability of being sampled. In general, any sampling scheme for selecting a TSR can be used as long as there is a nonzero probability of selecting any TSR.

4.1.4 Planning with TSRs as Goal Sets

TSRs can be used to sample goal end-effector placements of a manipulator, as would be necessary in a grasping or object placement task. The constraint for using TSRs in this way is

$$\{C(q) = \text{DistanceToTSR}(q), \quad s = [1]\}. \tag{14}$$

where the DistanceToTSR function implements the method of Sect. 4.1.2 and s refers to the domain of the constraint (Sect. 2).

To generate valid configurations in the $\mathcal{M}_C$ corresponding to this constraint, we can use direct sampling of TSRs (Sect. 4.1.3) and pass the sampled pose to an IK solver to generate a valid configuration. In order to ensure that we do not exclude any part of the constraint manifold, the IK solver used should not exclude any configurations from consideration. This can be achieved using an analytical IK solver for manipulators with six or fewer DOF. For manipulators with more than six DOF, we can use a pseudo-analytical IK solver, which discretizes or samples all but six joints.

Alternatively, we can use the projection strategy to sample the manifold. This would take the form of an iterative IK solver, which starts at some initial configuration. This configuration should be randomized to ensure exploration of the constraint manifold. Note that this strategy is prone to local minima and can be relatively slow to compute, so we use it only when an analytical or pseudo-analytical IK solver is not available (for instance, with a humanoid).

Of course the same definition and strategies apply to sampling starting configurations as well as goal configurations.

4.1.5 Planning with TSRs as Pose Constraints

TSRs can also be used for planning with constraints on end-effector pose for the entire path. The constraint definition for such a use of TSRs differs from Eq. 14 in the domain of the constraint:

$$\{C(q) = \text{DistanceToTSR}(q), \quad s = [0, 1]\}. \tag{15}$$

Since the domain of this constraint spans the entire path, the planning algorithm must ensure that each configuration it deems valid lies within the constraint manifold. While the rejection strategy can be used to generate valid configurations for TSRs whose bounds encompass a six-dimensional volume, the projection strategy can be used for all TSRs.

One method of projection for TSRs is shown in Algorithm 1. This method uses the Jacobian pseudo-inverse ($\mathbf{J}^+$) [31] to iteratively move a given configuration to the constraint manifold defined by a TSR.

The DisplacementFromTSR function returns the displacement from q to a TSR, i.e., the result of Eq. 11. The GetJacobian function computes the Jacobian of the manipulator at q. Though Algorithm 1 describes the projection conceptually, in practice we must also take into account the issues of step-size, singularity

Algorithm 1: $\mathbf{J}^{+}$Projection(q)

```
while true do
    Δx ← DisplacementFromTSR(q);
    if ‖Δx‖ < ϵ then
        return q;
    end
    J ← GetJacobian(q);
    Δq_error ← J^T (JJ^T)^{-1} Δx;
    q ← (q − Δq_error);
end
```

avoidance, and joint limits when projecting configurations. We showed, in [1], that the distribution of samples generated on the constraint manifold by this projection operator covers the manifold, which is a necessary property for probabilistic completeness of sampling-based planners.

It is important to note that we can also use the method of Sect. 4.1.3 to generate samples directly from TSRs and then compute IK to obtain configurations that place the end-effector at those samples. Such a strategy would be especially useful when planning in task space, i.e., the parameter space of pose constraints, instead of C-space because it would allow the task space to be explored while providing configurations for each task space point (similar to [32]). However, planning in C-space with projection methods allows us to compute configurations that satisfy multiple constraints at once, while at the same time guaranteeing probabilistic completeness.

4.2 Task Space Region Chains

While TSRs are intuitive to specify and can be quickly sampled, and the distance to TSRs can be evaluated efficiently, a single TSR, or even a finite set of TSRs, is sometimes insufficient to capture the pose constraints of a given task. To describe more complex constraints such as manipulating articulated objects, this chapter introduces the concept of TSR Chains, which are defined by linking a series of TSRs. Though direct sampling of TSR Chains follows clearly from that of TSRs, the distance metric for TSR Chains is extremely different.

To motivate the need for a more complex representation, consider the task of opening a door while allowing the end-effector to rotate about the door handle (see Fig. 5). It is straightforward to specify the rotation of the door about its hinge as a single TSR and to specify the rotation of the end-effector about the door's handle as a single TSR if the door's position is fixed. However, the product of these two constraints (allowing the end-effector to rotate about the handle for any angle of the hinge) cannot be completely specified with a finite set of TSRs. In order to allow more complex constraint representations in the TSR framework, we present TSR Chains, which are constructed by linking a series of TSRs.

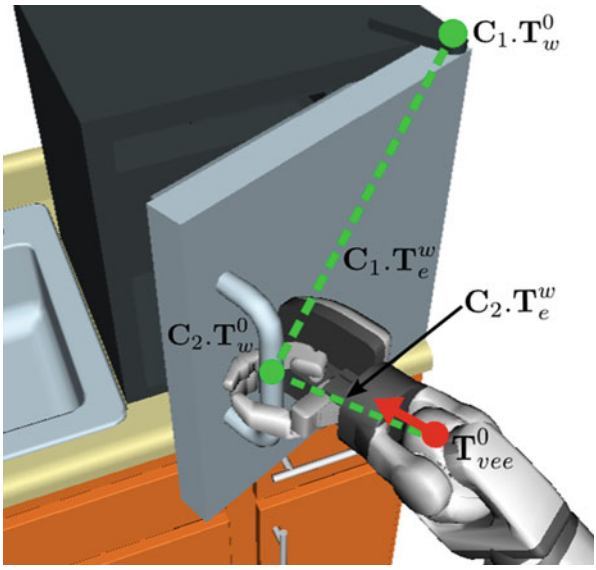

Fig. 5 The virtual manipulator for the door example. The *green dotted lines* represent the links of the virtual manipulator, and the *red dot* and *arrow* represent the virtual end-effector, which is at transform $\mathbf{T}^0_{vee}$

4.2.1 TSR Chain Definition

A TSR chain $\mathbf{C} = \{\mathbf{C}_1, \mathbf{C}_2, \ldots, \mathbf{C}_n\}$ consists of a set of n TSRs with the following additional property:

$$\mathbf{C}_i.\mathbf{T}^0_w = (\mathbf{C}_{i-1}.\mathbf{T}^0_w)(\mathbf{C}_{i-1}.\mathbf{T}^w_{\text{sample}})(\mathbf{C}_{i-1}.\mathbf{T}^w_e) \tag{16}$$

for $i = \{2..n\}$ where $\mathbf{C}_i$ corresponds to the ith TSR in the chain and $\mathbf{C}_i.\{\cdot\}$ refers to an element of the ith TSR. Of course a TSR Chain can consist of only one TSR, in which case it is identical to a normal TSR. $\mathbf{C}_i.\mathbf{T}^w_{\text{sample}}$ can be any transform obtained by sampling from inside the bounds of $\mathbf{C}_i.\mathbf{B}^w$. Thus we do not know $\mathbf{C}_i.\mathbf{T}^0_w$ until we have determined $\mathbf{T}^w_{\text{sample}}$ values for all previous TSRs in the chain. By coupling TSRs in this way, the TSR Chain structure can represent constraints that would otherwise require an infinite number of TSRs to specify.

A TSR chain can also be thought of as a virtual serial-chain manipulator. Again consider the door example. To define the TSR chain for this example, we can imagine a virtual manipulator that is rooted at the door's hinge. The first link of the virtual manipulator rotates about the hinge and extends from the hinge to the handle. At the handle, we define another link that rotates about the handle and extends to where a robot's end-effector would be if the robot were grasping the handle (see Fig. 5). $\mathbf{C}_1.\mathbf{T}^w_{\text{sample}}$ would be a rotation about the door's hinge corresponding to how much the door had been opened. In this way, we could see the $\mathbf{T}^w_{\text{sample}}$ values for each TSR as transforms induced by the "joint angles" of the virtual manipulator. The joint limits of these virtual joints are defined by the values in $\mathbf{B}^w$.

4.2.2 Direct Sampling from TSR Chains

To directly sample a TSR Chain, we first sample from within $\mathbf{C}_1.\mathbf{B}^w$ to obtain $\mathbf{C}_1.\mathbf{T}^w_{\text{sample}}$. This is done by sampling uniformly between the bounds in $\mathbf{B}^w$, compiling the sampled values into a displacement $d^w_{\text{sample}} = [x\ y\ z\ \psi\ \theta\ \phi]$, and converting that displacement to the transform $\mathbf{C}_1.\mathbf{T}^w_{\text{sample}}$. We then use this sample to determine $\mathbf{C}_2.\mathbf{T}^0_w$ via Eq. 16. We repeat this process for each TSR in the chain until we reach the nth TSR. We then obtain a sample in the world frame:

$$\mathbf{T}^0_{\text{sample}'} = (\mathbf{C}_n.\mathbf{T}^0_w)(\mathbf{C}_n.\mathbf{T}^w_{\text{sample}})(\mathbf{C}_n.\mathbf{T}^w_e). \tag{17}$$

Note that the sampling of TSR Chains in this way is biased, but the sampling will cover the entire set. To see this, imagine a virtual manipulator with many links. It can be readily seen that many sets of different joint values (essentially $\mathbf{T}^w_{\text{sample}}$ values) of the virtual manipulator will map to the same end-effector transform. However, if the virtual manipulator's end-effector is at the boundary of the virtual manipulator's reachability, only one set of joint values maps to the end-effector pose (when the manipulator is fully outstretched). Thus some $\mathbf{T}^0_{\text{sample}'}$ values can have a higher chance of being sampled than others, depending on the definition of the TSR Chain. Clearly a uniform sampling would be ideal, but we have found that this biased sampling is sufficient for the practical tasks we consider.

If there is more than one TSR Chain defined for a single manipulator, this means that we have the option of drawing a sample from any of these TSR Chains. We choose a TSR Chain for sampling with probability proportional to the sum of the differences between the bounds of all TSRs in that chain.

4.2.3 Distance to TSR Chains

Though the sampling method for TSR Chains follows directly from the sampling method for TSRs, evaluating distance to a TSR Chain is fundamentally different from evaluating distance to a TSR. This is because we do not know which $\mathbf{T}^w_{\text{sample}}$ values for each TSR in the chain yield the minimum distance to a query transform $\mathbf{T}^0_s$ (derived from a query configuration q_s using forward kinematics).

To approach this problem, it is again useful to think of the TSR chain as a virtual manipulator (see Fig. 6a). Finding the correct $\mathbf{T}^w_{\text{sample}}$ values for each TSR is equivalent to finding the joint angles of the virtual manipulator that bring its virtual end-effector as close to $\mathbf{T}^0_s$ as possible. Thus we can see this distance-checking problem as a form of the standard IK problem, which is to find the set of joint angles that places an end-effector at a given transform. Depending on the TSR Chain definition and $\mathbf{T}^0_s$, the virtual manipulator may not be able to reach the desired transform, in which case we want the virtual end-effector to get as close as possible. Thus we can apply standard iterative IK techniques based on the Jacobian pseudo-inverse to move the virtual end-effector to a transform that is as close as possible to $\mathbf{T}^0_s$ (see Fig. 6b). Once we obtain the joint angles of the virtual manipulator, we convert them to $\mathbf{T}^w_{\text{sample}}$ values and forward-chain to obtain the virtual end-effector position $\mathbf{T}^0_{\text{vee}}$. We then convert $\mathbf{T}^0_s$ to the virtual end-effector's frame:

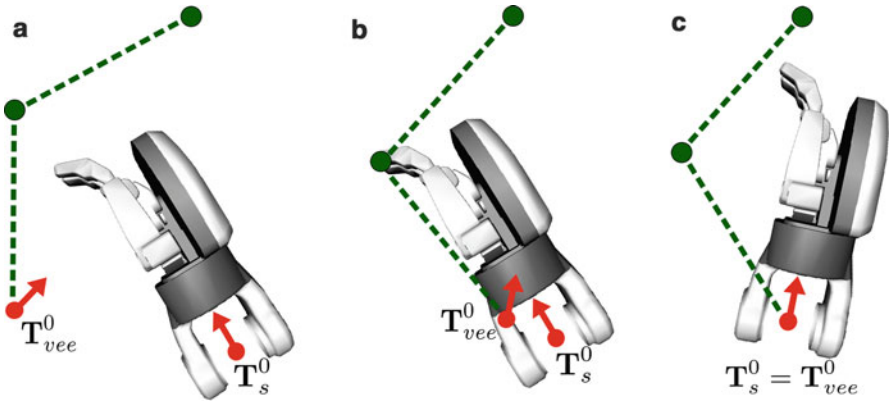

Fig. 6 Depiction of the IK handshaking procedure. (**a**) The virtual manipulator starts in some configuration. (**b**) Finding the closest configuration of the virtual manipulator. (**c**) The robot's manipulator moves to meet the constraint

$$\mathbf{T}_s^{\text{vee}} = (\mathbf{T}^0_{\text{vee}})^{-1}\mathbf{T}^0_s \tag{18}$$

and then convert to the displacement form

$$d_s^{vee} = \begin{bmatrix} \mathbf{t}_s^{\text{vee}} \\ \arctan 2(\mathbf{R}^{\text{vee}}_{s_{32}}, \mathbf{R}^{\text{vee}}_{s_{33}}) \\ -\arcsin(\mathbf{R}^{\text{vee}}_{s_{31}}) \\ \arctan 2(\mathbf{R}^{\text{vee}}_{s_{21}}, \mathbf{R}^{\text{vee}}_{s_{11}}) \end{bmatrix}. \tag{19}$$

$\|d_s^{\text{vee}}\|$ is the distance between $\mathbf{T}^0_s$ and $\mathbf{T}^0_{\text{vee}}$.

Once the distance is evaluated, we can employ the projection strategy by calling the IK algorithm for the robot's manipulator to move the robot's end-effector to $\mathbf{T}^0_{\text{vee}}$ to meet the constraint specified by this TSR Chain (Fig. 6c). We term this process of calling IK for the virtual manipulator and the robot in sequence *IK Handshaking*.

Just as with TSR Chains used for sampling, we may define more than one TSR Chain as a constraint for a single manipulator. This means that we have the option of satisfying any of these TSR Chains to produce a valid configuration. To find which chain to satisfy, we perform the distance check from our current configuration to each chain and choose the one that has the smallest distance.

4.2.4 Physical Joints and TSR Chains

In the door example, the first TSR corresponds to a physical joint of a body in the environment, but the second one is purely virtual, i.e., defining a relation between two frames that is not enforced by a joint in the environment (in this case the relation is between the robot's end-effector and the handle of the door). It is important to note that TSR Chains inherently accommodate such mixing of real and virtual

constraints. In fact a TSR Chain can consist of purely virtual or purely physical constraints. However, when planning with TSR Chains, special care must be taken to ensure that any physical joints (such as the door's hinge) be synchronized with their TSR Chain counterparts. This is done by including the configuration of any physical joints corresponding to elements of TSR Chains in the configuration space searched by the planner.

In the case that the physical constraints included in the TSR Chain form a redundant manipulator, the inverse-kinematics algorithm for the TSR Chain should be modified to account for the physical properties of the chain. For instance, if the chain is completely passive, a term that minimizes the potential energy of the chain should be applied in the null space of the Jacobian pseudo-inverse to find a local minimum-energy configuration of the chain. In general, chains can have various physical properties that may not be easy to account for using an IK solver. In that case, we recommend a physical simulation of the movement of the end-effector from its initial pose to $\mathbf{T}^0_{\text{vee}}$ as it is being pulled by the robot to find the resting configuration of the chain.

5 Closed-Chain Kinematics Constraints

Another important constraint to address in whole-body or two-arm manipulation is closed-chain kinematics constraints. For instance, a closed-kinematic chain is formed whenever a humanoid is in the double-support phase or whenever both the humanoid's arms are holding an object. The closed-chain constraint is difficult to plan with because it induces a lower-dimensional constraint manifold in the configuration space that cannot be sampled using rejection sampling. What is worse, the closed-chain configuration manifold consists of parts that are of varying dimensionality; the manifold is "pinched" at kinematic singularities [33] (see Fig. 7).

Approaches for sampling these kinds of constraint manifolds fall into two categories: direct sampling and projection methods. In direct sampling, the end-effector poses can be generated first (e.g., relative to sampled positions of the object, both hands are holding), and IK is solved to generate the robot configuration. However, the IK solver may generate very different configurations for nearby poses

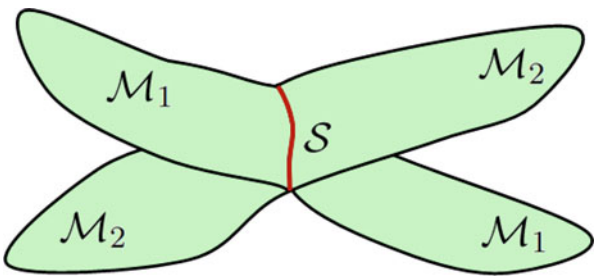

Fig. 7 Illustration of a singularity in a closed-chain constraint manifold. $\mathcal{M}_1$ and $\mathcal{M}_1$ are pieces of the closed-chain constraint manifold. $\mathcal{M}_1$ and $\mathcal{M}_2$ intersect at a singular part of the manifold $\mathcal{S}$, which is of a lower dimension than $\mathcal{M}_1$ and $\mathcal{M}_2$

(e.g., elbow up vs. elbow down), making it difficult to connect samples into a graph/tree. Nevertheless, effective direct sampling methods have been produced, such as the random loop generator [34] and [33]. However, it is may be difficult to integrate these specialized methods with other simultaneous constraints the humanoid must obey.

The second approach is projection sampling. In fact, one of the earliest projection sampling approaches in motion planning was applied to the problem of generating configurations that obeyed closed-chain kinematics constraints [35]. It is also possible to apply projection sampling to closed-chain constraints by representing them using the TSR framework described above. This can be done in two ways: For end-effector poses that do not move (e.g., the feet during a reaching motion), we can specify TSRs that keep the end-effectors at a certain pose by setting $\mathbf{B}^w = \mathbf{0}_{6\times 2}$. For end-effector poses that do move, we specify a TSR for one end-effectors with respect to the other end-effector involved in the closed chain. For example, consider that the robot is lifting a large box with two hands. To encode the closed-chain constraint, we specify that the $\mathbf{T}_w^0$ of the second arm be defined with respect to the end-effector pose of the first arm instead of the world frame. When we sample a new configuration of the robot in the ambient C-space, the arms will naturally not obey the closed-chain constraint, and when we apply the projection for TSRs, the second arm will move to obey the closed-chain constraint, while the first arm stays fixed. In this way closed-chain constraints can be encoded in the same way as pose constraints (and can be specified alongside pose constraints) in the TSR framework. This construction will allow us to sample the generic parts of the constraint manifold, but specialized sampling methods will be needed to explicitly explore the singular parts of the manifold (such as those described above). Though we have not observed the need to incorporate these specialized methods in practice, they may be useful when the only way to reach a goal is to move through a singular configuration.

6 Balance Constraints

Unlike fixed-base robots or many mobile manipulators, motion planning for humanoid robots requires taking into account the balance constraints of the robot. This chapter focuses on quasi-static motion planning, so here we consider quasi-static balance constraints. Many methods in humanoid walking consider dynamic balance criteria such as zero moment point (ZMP) [36]; however in the quasi-static case, the constraint on the robot reduces to keeping its center of mass (CoM) above its support polygon. When the contacts between the robot and the environment are restricted to a single plane (e.g., two feet on flat ground), the support polygon is the convex hull of the contacts. Support polygons can also be computed for nonplanar contacts, as shown in work on climbing robots [37,38]. Recent work [39] has shown how to generalize stability computation to nonplanar contacts without computing a support polygon. Instead they use methods similar to those developed for checking the force-closure criterion in grasping [40], which operate on the contacts directly.

In the case of planar contacts, unless the support polygon is degenerate (i.e., reduces to a line), the manifold of configurations that are in balance is of the same dimension as the C-space. Thus, we can use rejection sampling to find configurations that are in balance by sampling a configuration, performing forward kinematics to determine the pose of all links, computing the center of mass, and checking if the projection of the center of mass onto the ground plane (in the case of planar contact) is within the support polygon or not.

Projection can also be used to find configurations that are in balance. Projecting a configuration to the balance constraint manifold requires computing a Jacobian for the CoM [41]. Let $\mathbf{J}_i$ be the Jacobian for the CoM of link i of the robot and m_i be the mass of this link. Then the Jacobian for the CoM of the robot is

$$\mathbf{J}_{\text{com}} = \frac{\sum_i m_i \mathbf{J}_i}{\sum_i m_i}. \tag{20}$$

Using the pseudo-inverse of $\mathbf{J}_{\text{com}}$, we can servo the center of mass toward a given target, such as the center of the support polygon or the closest boundary of the polygon, which effectively projects the configuration of the robot to the constraint manifold. It is also possible to combine this projection operation with projections to other constraints, e.g., to satisfy pose constraints for the end-effectors using recursive null-space projection [17].

7 Example Problems

To demonstrate how the above constraints can be used for practical tasks involving robot arms and humanoids, this section describes several example problems described in terms of the constraints used. We also present results from running the CBiRRT2 motion planning algorithm [1] on these problems. CBiRRT2 is a sampling-based planner that uses projection sampling to satisfy pose and closed-chain kinematics constraints and rejection sampling to satisfy constraints on balance and collision.

The first four examples are implemented on a 7DOF Barrett arm and the last three on the 28DOF of the HRP3 robot. The first three examples describe how to use TSRs for goal pose specification. The next example shows how to use TSRs as pose constraints throughout the path. The last three examples show how to mix goal and pose TSRs and TSR Chains. At the end of this chapter, we analyze the computational cost of the operations of CBiRRT2 on a door-opening task for both robots.

Since the TSR Chain representation subsumes the TSR representation, each example problem can be implemented using TSR Chains. However we do not describe a chained implementation when only chains of length 1 are used so that the explanation is clearer. All experiments were performed on a 2.4 GHz Intel CPU with 4 GB of RAM using the OpenRAVE simulation and planning environment [42]. The

planner takes two parameters, $\Delta q_{\text{step}} \in \mathbb{R}$, which is the step size between nodes in the RRT extension operation, and $P_{\text{sample}} \in [0, 1]$, which is the probability of the algorithm attempting to sample a new goal configuration (vs. extending the tree) in a given iteration. The P_{sample} parameter is only relevant when a constraint with $s = 1$ has been defined. The numerical error allowed in meeting a pose constraint was $\epsilon = 0.001$.

7.1 Reaching to Grasp an Object

Our goal in this problem is to grasp an object for which we can define a continuum of acceptable grasp poses. These grasp poses can be encoded into TSRs and passed to our planner. We define four TSRs for the pitcher we wish to grasp (see Fig. 8a): two for the top of the pitcher and two for the handle. The $\mathbf{T}_w^0$ and $\mathbf{T}_e^w$ transforms of these TSRs are shown in Fig. 3. The two bounds for the top TSRs are identical, as are the two bounds for the handle TSRs:

$$\mathbf{B}_{\text{top}}^w = \begin{bmatrix} \mathbf{0}_{5\times 2} \\ -0.3 \;\; 0.3 \end{bmatrix} \qquad \mathbf{B}_{\text{handle}}^w = \begin{bmatrix} \mathbf{0}_{2\times 2} \\ -0.03 \;\; 0.02 \\ \mathbf{0}_{3\times 2} \end{bmatrix} \tag{21}$$

The top TSRs allow the robot to grasp the pitcher from the top with limited hand rotation about the z-axis. The handle TSRs allow the robot to grasp the pitcher anywhere along the handle but do not allow any offset in hand rotation. A trajectory produced by CBiRRT2 is shown in Fig. 8a. The RRT step-size Δq_{step} was set to 0.05 and $P_{\text{sample}} = 0.1$. The average planner runtime for 15 trials was 0.04 s.

7.2 Reaching to Grasp Multiple Objects

In this problem the robot's task is to reach and grasp one of the seven randomly placed soda cans on a table (see Fig. 8b). Each soda can is treated as a cylinder and

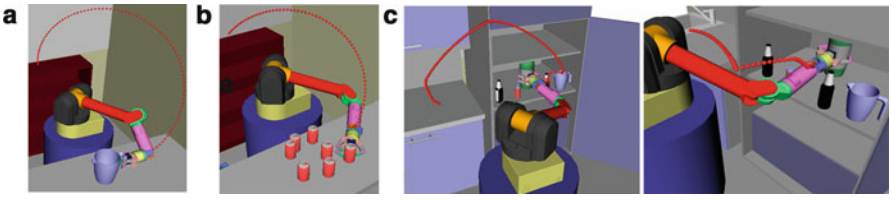

Fig. 8 Paths of the end-effector produced by CBiRRT2 for the three goal TSR examples. (**a**) Reaching to grasp a pitcher. (**b**) Reaching to grasp one of many soda cans. (**c**) Placing a bottle into the refrigerator *left*: fixed base, *right*: mobile base. The paths shown have been smoothed with 500 iterations of the shortcut smoothing algorithm

two TSRs are defined for each can. The $\mathbf{T}_w^0$ and $\mathbf{T}_e^w$ transforms are shown in Fig. 3. Both TSRs for each can have identical bounds:

$$\mathbf{B}^w = \begin{bmatrix} \mathbf{0}_{5\times 2} \\ -\pi \quad \pi \end{bmatrix}. \tag{22}$$

These bounds allow the grasp to rotate about the z-axis of the can, thus allowing it to grasp the can from any direction in the plane defined by the x- and y-coordinates of the can's center. Note that we do not specify which soda can to grasp; this choice is made within the planner when sampling from the TSRs. A trajectory produced by CBiRRT2 is shown in Fig. 8b. $\Delta q_{step} = 0.05$ and $P_{sample} = 0.1$. The average planner runtime for 15 trials was 0.21 s.

7.3 Placing an Object into a Cluttered Space

The task in this problem is for the robot to place the bottle it is holding into a very cluttered location (see Fig. 8c). The bottles in the refrigerator and the upper refrigerator shelf make it difficult for the robot to find a path that places the large bottle it is holding onto the middle refrigerator shelf. $\mathbf{T}_w^0$ is defined at the center of the middle shelf, and $\mathbf{T}_e^w$ is defined as an end-effector position pointing along the y-axis (away from the robot) that is holding the bottle at $\mathbf{T}_w^0$:

$$\mathbf{B}^w = \begin{bmatrix} -0.24 \; 0.24 \\ -0.34 \; 0.34 \\ \mathbf{0}_{4\times 2} \end{bmatrix} \tag{23}$$

This $\mathbf{B}^w$ defines a plane on the shelf where the bottle can be placed. $\Delta q_{\text{step}} = 0.05$ and $P_{\text{sample}} = 0.1$. The average planner runtime for 15 trials was 93 s.

7.4 The Maze Puzzle

In this problem, the robot arm must solve a maze puzzle by drawing a path through the maze with a pen (see Fig. 9). The constraint is that the pen must always be touching the table; however the pen is allowed to pivot about the contact point up to an angle of α in both roll and pitch. We define the end-effector to be at the tip of the pen with no rotation relative to the world frame. To specify the constraint in this problem, we define one pose constraint TSR with $\mathbf{T}_w^0$ to be at the center of the maze with no rotation relative to the world frame (z being up). $\mathbf{T}_e^w$ is identity and

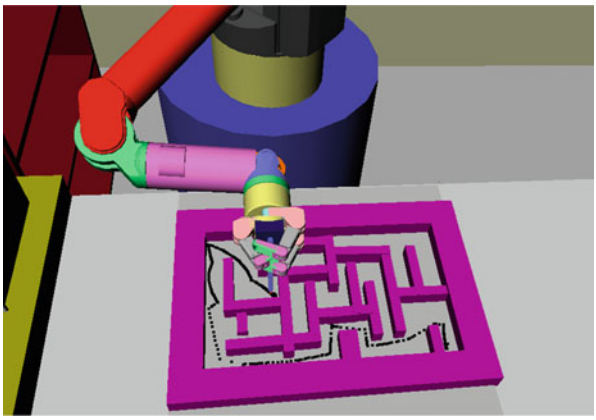

Fig. 9 A path found for the maze puzzle using $\alpha = 0.4$ rad. The *black points* represent positions of the tip of the pen along the path

Table 1 Simulation results for Maze Puzzle

α(rad.)	0.0	0.1	0.2	0.3	0.4	0.5
Avg. runtime	>83.5 s	>58.8 s	>49.0 s	19.5 s	14.3 s	15.2 s
Success rate	40%	60%	90%	100%	100%	100%

$$\mathbf{B}^w = \begin{bmatrix} -\infty & \infty \\ -\infty & \infty \\ 0 & 0 \\ -\alpha & \alpha \\ -\alpha & \alpha \\ -\pi & \pi \end{bmatrix}. \tag{24}$$

IK solutions were generated for both the start and goal position of the pen using the given grasp and input as Q_s and Q_g. The values in Table 1 represent the average of ten runs for different α values. Runtimes with a ">" denote that there was at least one run that did not terminate before 120 s. For such runs, 120 was used in computing the average. The RRT step-size Δq_{step} was set to 0.05. No goal sampling is performed in this example.

The shorter runtimes and high success rates for larger α values demonstrate that the more freedom we allow for the task, the easier it is for the algorithm to solve it. This shows a key advantage of formulating the constraints as bounds on allowable pose as opposed to requiring the pose of the object to conform exactly to a specified value, as in [19]. For problems where we do not need to maintain an exact pose for an object, we can allow more freedom, which makes the problem easier. See Fig. 9 for an example path of the tip of the pen.

7.5 Closed-Chain Kinematics

The task is for the HRP3 humanoid to pick up the box from the bottom of the bookshelf and place it on top (see Fig. 10a). There are two closed chains which must be enforced by the planner; the legs and arms form two separate loops (Fig. 10).

We define three TSRs. The first TSR is assigned to the left leg of the robot and allows no deviation from the current left-foot location (i.e., $\mathbf{B}^w = \mathbf{0}_{6\times 2}$). The second and third TSRs are assigned to the left and right arms and are defined relative to the location of the box (i.e., the 0 frame of $\mathbf{T}_w^0$ is the frame of the box). The bounds are defined such that the hands will always be holding the sides of the box at the same locations ($\mathbf{B}^w = \mathbf{0}_{6\times 2}$). The geometry of the box is "attached" to the right hand.

Balance for a given configuration of the robot is checked after projection has been performed to meet the pose and closed-chain constraints. Projected configurations that are not in balance or are in collision are rejected. $\Delta q_{\text{step}} = 0.05$. In this problem we compute the goal configuration of the robot using inverse kinematics on the box target; no goal sampling is performed in this example.

The result of this construction is the following: When a new configuration q_s is generated through sampling, the box moves with the right hand and the frame of the box changes, thus breaking the closed-chain constraint. This q_s is projected to meet the constraint (i.e., moving the left arm). Meanwhile, the TSR for the right hand ensures the box does not move during the projection. The same process happens simultaneously for the left leg of the robot as well since the kinematic chain is rooted at the right leg.

We implemented this example in simulation and on the physical HRP3 robot. Runtimes for 30 runs of this problem in simulation can be seen in Table 2. On the physical robot, the task was to stack two boxes in succession; snapshots from the execution of the plan can be seen in Fig. 11.

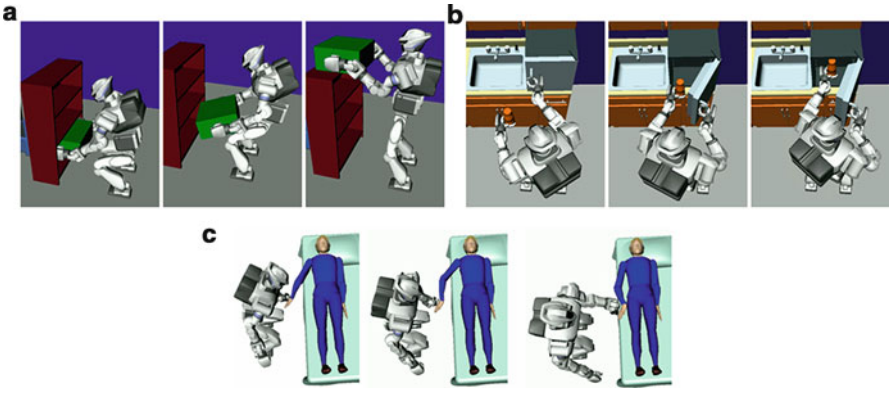

Fig. 10 Snapshots from paths produced by our planner for the three examples using HRP3 in simulation. (**a**) Closed chain kinematics example. (**b**) Simultaneous constraints and goal sampling example. (**c**) Manipulating a passive chain example

Fig. 11 Snapshots from the execution of the box stacking task on the HRP3 robot

7.6 Simultaneous Constraints and Goal Sampling

The task in this problem is to place a bottle held by the robot into a refrigerator (see Fig. 10b). Usually, such a task is separated into two parts: first open the refrigerator and then place the bottle inside. However, with TSRs, there is no need for this separation because we can implicitly sample how much to open the refrigerator and where to put the bottle at the same time. The use of TSR Chains is important here, because it allows the right arm of the robot to rotate about the handle of the refrigerator, which gives the robot more freedom when opening the door. We assume that the grasp cages the door handle so the end-effector can rotate about the handle without the door escaping.

There are four TSR Chains defined for this problem. The first is the TSR Chain (1 element) for the left leg, which is the same as in the previous example. This TSR Chain is marked for both sampling goals and constraining pose throughout the path. The second TSR Chain (2 element) is defined for the right arm and is described in Sect. 4.2.1. This chain is also marked for both sampling goals and constraining pose throughout the path. The third TSR Chain (1 element) is defined for the left arm and constrains the robot to disallow tilting of the bottle during the robot's motion. This chain is used only as a pose constraint. Its bounds are

$$\mathbf{B}^w = \begin{bmatrix} -\infty & \infty \\ -\infty & \infty \\ -\infty & \infty \\ 0 & 0 \\ 0 & 0 \\ -\pi & \pi \end{bmatrix}. \tag{25}$$

The final TSR Chain (1 element) is also defined for the left arm and represents the allowable placements of the bottle inside the refrigerator. Its $\mathbf{B}^w$ has freedom in x and y corresponding to the refrigerator width and length and no freedom in any other dimension. This chain is only used for sampling goal configurations. $\Delta q_{\text{step}} = 0.05$ and $P_{\text{sample}} = 0.1$ for this example.

The result of this construction is that the robot simultaneously samples a target bottle location and wrist position for its right arm when sampling goal configurations; thus it can perform the task in one motion instead of in sequence. Another important point is that we can be rather sloppy when defining TSRs for goal sampling. Observe that many samples from the right arm's TSR chain will

Table 2 Runtimes for example problems using HRP3

	Mean	Std. Dev
Closed chain kinematics	4.21 s	2.00 s
Simultaneous constraints and goal sampling	1.54 s	0.841 s
Manipulating a passive chain	1.03 s	0.696 s

leave the door closed or marginally open, thus placing the left arm into collision if it is reaching inside the refrigerator. However, this is not an issue for the planner because it can always sample more goal configurations and the collision constraint is included in the problem. Theoretically, a TSR Chain defined for goal sampling need only be a super set of the goal configurations that meet all constraints. However, as the probability of sampling a goal from this TSR chain which meets all constraints decreases, the planner will usually require more time to generate a feasible goal configuration, thus slowing down the algorithm.

Runtimes for 30 runs of this problem in simulation can be seen in Table 2.

7.7 Manipulating a Passive Chain

The task in this problem is for the robot to assist in placing a disabled person into bed (see Fig. 10c). The robot's task is to move the person's right hand to a specified point near his body. The person's arm is assumed to be completely passive, and the kinematics of the arm (as well as joint limits) are assumed to be known. The robot's grasp of the person's hand is assumed to be rigid.

There are two TSR Chains defined for this problem, both of which are used as pose constraints. The first is the TSR Chain (1 element) for the left leg, which is the same as the previous example. The second is a TSR Chain (6 element) defined for the person's arm. Every element of this chain corresponds to a physical DOF of the person's arm. Note that since the arm is not redundant, we do not need to perform any special IK to ensure that the configuration of the person matches what it would be in the real world.

In this problem we get the goal configuration of the robot from inverse kinematics on the target pose of the person's hand; no goal sampling is performed. $\Delta q_{\text{step}} = 0.05$.

The result of this construction is that the person's arm will follow the robot's left hand. Since the configuration of the person's arm is included in q, there cannot be any significant discontinuities in the person's arm configuration (i.e., elbow up to elbow down) because such configurations are distant in the C-space. Runtimes for 30 runs of this example in simulation can be seen in Table 2.

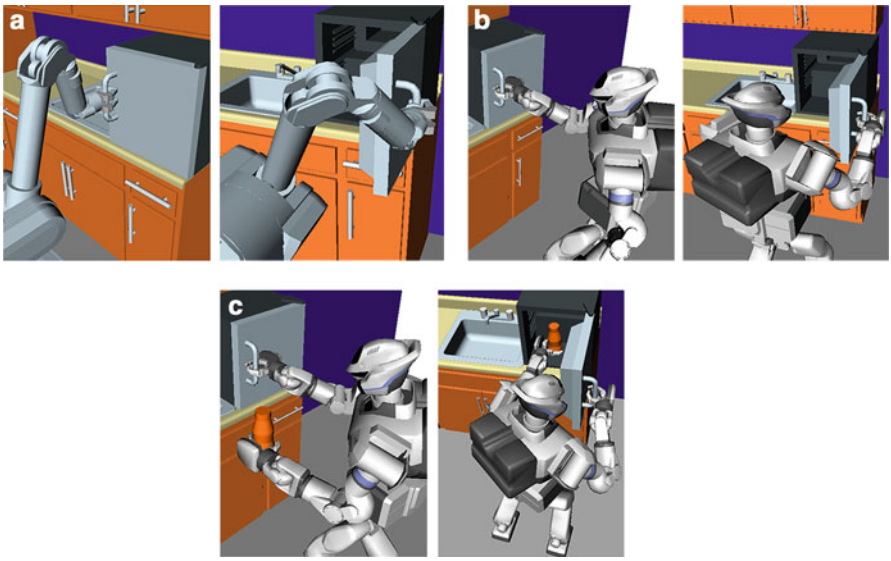

Fig. 12 Start and goal configurations for the door-opening task used for the timing experiments. (**a**) WAM (7DOF) (**b**) HRP3 (9DOF) (**c**) HRP3 (28DOF)

7.8 Runtimes for Rejection and Projection

To evaluate the computational cost of the operations of CBiRRT2, we performed a runtime comparison on a door-opening task for the WAM and HRP3 (see Fig. 12). We performed three experiments to gauge how the computation times of the main components of the algorithm (projection, collision checking, and nearest neighbor queries) scaled with the DOF of the robot.

In the first experiment, we use a 7DOF WAM to open a refrigerator door 90°. The constraint in this problem is formulated as a TSR Chain, similar to the one described in Sect. 7.6. The TSR Chain has two elements and allows the robot to rotate its hand about the door handle as well as allowing the door to rotate about its hinge.

The second experiment is identical to the first, except that we use the HRP3 robot instead of the WAM. We allow the robot to use its waist and right arm joints, for a total of 9DOF.

Finally, the third experiment is the same as the one described in Sect. 7.6 except that we have a pre-determined goal configuration where the door is opened to 90° and the bottle is placed in the refrigerator, so there is no goal sampling. In this experiment we use the arms, legs, and waist of the HRP3, for a total of 28DOF.

We ran each experiment 50 times and the total computation times averaged over all runs are shown in Table 3. We also show the average time needed to do a single projection, collision check, and nearest neighbor query averaged over all runs.

The results in Table 3 show that collision checking is the most time-consuming operation of the algorithm. However, as the number of DOF increases, the average

Table 3 Total and average computation times for the main components of CBiRRT2

	Projection (total)	Col. check (total)	NN query (total)	Projection (avg)	Col. check (avg)	NN query (avg)
WAM (7DOF)	0.1324 s	0.3650 s	7.2×10^{-6} s	0.0008 s	0.0037 s	2.1×10^{-6} s
HRP3 (9DOF)	0.1372 s	0.5276 s	1.2×10^{-5} s	0.0009 s	0.0048 s	2.5×10^{-6} s
HRP3 (28DOF)	0.8469 s	2.049 s	0.0017 s	0.0018 s	0.0052 s	2.3×10^{-5} s

projection time increases significantly. This is because the size of the matrices involved in the computation of the Jacobian pseudo-inverse, which is used to perform the projection, increases with the number of DOF. The projection also involves calling the forward kinematics function of the robot to obtain the robot's end-effector pose as well as computing the Jacobian of the end-effector, both of which become slower with increasing DOF.

8 Discussion and Future Work

This chapter has presented a framework for representing and exploring feasible configurations in the context of manipulation planning and shown how to use this framework to solve several manipulation problems for humanoids. The techniques presented in this chapter can be applied to manipulation planning tasks where the constraints are evaluated as functions of a robot's configuration. The work presented here is not meant to address tasks that require complex forceful interaction with the environment, such as the peg-in-hole problem, or tasks that require planning with dynamics, such as throwing a ball. Though we only consider scleronomic holonomic constraints and quasi-static motion, these restrictions still allow a robot to perform many useful tasks (as shown in Sect. 7) and permit a rich variety of constraints, including constraints on collision-avoidance, torque, balance, closed-chain kinematics, and end-effector pose.

Some of the most common constraints in manipulation planning involve the pose of a robot's end-effector. These constraints arise in tasks such as reaching to grasp an object, carrying a cup of coffee, or opening a door. The main advantage of the approach presented here to planning with pose constraints is the generality in the constraint representation. TSRs are able to tackle a wide range of problems without resorting to highly specialized techniques and representations.

One criticism of TSRs is that the constraint representation may not be sufficiently rich. For instance, some modifications to TSR Chains are necessary to accommodate constraints where degrees of freedom are coupled (as with screw constraints). Indeed, TSRs and TSR Chains cannot capture every conceivable constraint, nor are they intended to. Instead, these representations attempt to straddle the trade-off between practicality and expressiveness. TSRs have proven sufficient for solving a wide range of real-world manipulation problems while still remaining relatively simple and efficient to use in a sampling-based planner. While a more expressive

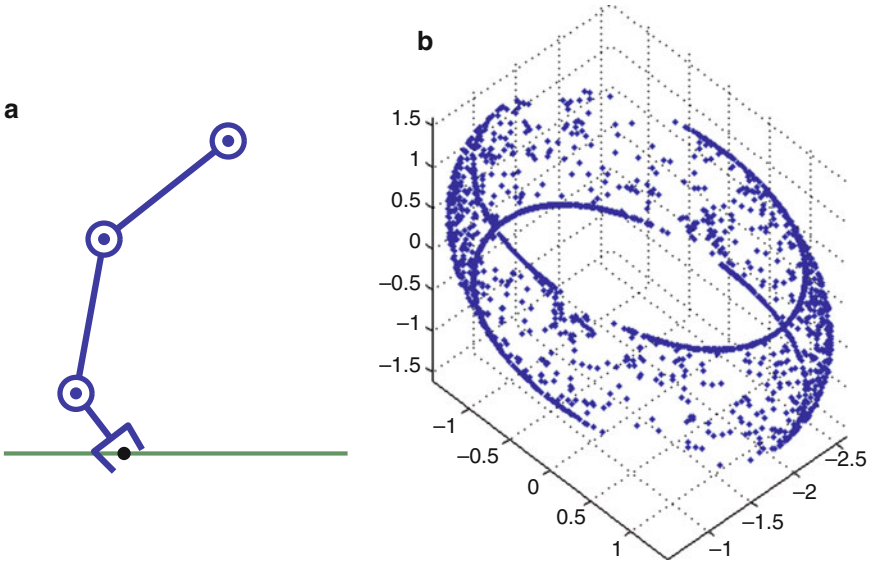

Fig. 13 Depiction of a TSR and samples on the corresponding constraint manifold (generated using CBiRRT2). (**a**) The end-effector must be on the line with an orientation within ± 0.7 rad of downward. (**b**) The sampling is biased toward the boundaries of the manifold

representation is surely possible, we have yet to find one that is as straightforward to specify and as convenient for sampling-based planning.

A practical issue with the projection sampling approach to sampling constraint manifolds is that the distribution of samples may sometimes be undesirable, e.g. the projection strategy biases samples toward the boundaries of the manifold (Fig. 13). This bias leads to an over-exploration of the boundaries of the manifold to the detriment of exploring the manifold's interior. It can cause the planning algorithm to perform slowly if an interior point of the manifold is needed to complete a path. On the other hand, the planner is much faster at finding configurations on the boundary, which may be useful in some applications.

In future work it would be interesting to explore methods that automatically generate TSRs for a given task. For instance, could a robot determine all the areas where a given object can be placed directly from the geometry of the scene? Such a task would require understanding where the object *can* be placed (through grounding the concept of placing geometrically) and also taking into account user preferences for where objects *should* be placed. Another important issue to explore would be extracting constraints from visual data and/or interaction with the environment [43, 44]. This would be useful for inferring the locations of door hinges or other articulated mechanisms, for example. Automatically creating TSRs for these constraints would greatly reduce the need for domain and robotics expertise in programming the robot to perform the useful tasks.

References

1. D. Berenson, S. Srinivasa, J. Kuffner, Task space regions: a framework for pose-constrained manipulation planning. Int. J. Robotics Res. **30**(12), 1435–1460 (2011)
2. D. Berenson, S. Srinivasa, Probabilistically complete planning with end-effector pose constraints, in *Proceedings of IEEE International Conference on Robotics and Automation (ICRA)*, May 2010
3. J. Kuffner, K. Nishiwaki, S. Kagami, M. Inaba, H. Inoue, Motion planning for humanoid robots under obstacle and dynamic balance constraints, in *IEEE International Conference on Robotics and Automation (ICRA)*, 2001
4. J. Kuffner, S.M. LaValle, RRT-connect: an efficient approach to single-query path planning, in *Proceedings of IEEE International Conference on Robotics and Automation (ICRA)*, 2000
5. L.E. Kavraki, P. Svestka, J.C. Latombe, M.H. Overmars, Probabilistic roadmaps for path planning in high-dimensional configuration spaces. IEEE Trans. Robot. Autom. **12**(4), 566–580 (1996)
6. P. Jiménez, F. Thomas, C. Torras, 3D collision detection: a survey. Comput. Graph. **25**(2), 269–285 (2001)
7. E. Larsen, S. Gottschalk, M. Lin, D. Manocha, Fast proximity queries with swept sphere volumes, in *Proceedings of IEEE International Conference on Robotics and Automation (ICRA)*, 2000
8. I. Sucan, S. Chitta, J. Pan, FCL: a flexible collision library (Accessed 2015). [Online]. Available: http://gamma.cs.unc.edu/FCL/fcl_docs/webpage/generated/index.html
9. F. Kanehiro, H. Hirukawa, Online self-collision checking for humanoids, in *19th Annual Conference of Robotics Society of Japan*, 2001
10. K. Okada, T. Ogura, A. Haneda, J. Fujimoto, F. Gravot, M. Inaba, Humanoid motion generation system on hrp2-jsk for daily life environment, in *IEEE International Conference Mechatronics and Automation*, 2005
11. J. Kuffner, K. Nishiwaki, S. Kagami, Y. Kuniyoshi, M. Inaba, H. Inoue, Self-collision detection and prevention for humanoid robots, in *IEEE International Conference on Robotics and Automation (ICRA)*, 2002
12. N.M. Amato, O.B. Bayazit, L.K. Dale, C. Jones, D. Vallejo, OBPRM: an obstacle-based PRM for 3D workspaces, in *Proceedings of the Third Workshop on the Algorithmic Foundations of Robotics (WAFR)*, Aug 1998, pp. 155–168
13. D. Hsu, J. Reif, The bridge test for sampling narrow passages with probabilistic roadmap planners, in *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, 2003
14. D. Hsu, G. Sanchez-Ante, H.-L. Cheng, J.-C. Latombe, Multi-level free-space dilation for sampling narrow passages in PRM planning, in *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, 2006
15. S. Dalibard, J.-P. Laumond, Control of probabilistic diffusion in motion planning, in *Proceedings of the Workshop on the Algorithmic Foundations of Robotics (WAFR)*, 2008
16. D.M. Liangjun Zhang, Y.J. Kim, A fast and practical algorithm for generalized penetration depth computation, in *Robotics: Science and Systems (RSS)*, 2007
17. L. Sentis, O. Khatib, Synthesis of whole-body behaviors through hierarchical control of behavioral primitives. Int. J. Humanoid Rob. **2**, 505–518 (2005)
18. J. Pan, L. Zhang, D. Manocha, Retraction-based RRT planner for articulated models, in *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, May 2010
19. M. Stilman, Task constrained motion planning in robot joint space, in *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2007
20. L. Jaillet, J. Cortés, T. Siméon, Sampling-based path planning on configuration-space costmaps. IEEE Trans. Robot. **26**(4), 635–646 (2010)

21. C. Suh, T.T. Um, B. Kim, H. Noh, M. Kim, F.C. Park, Tangent space RRT: a randomized planning algorithm on constraint manifolds, in *IEEE International Conference on Robotics and Automation (ICRA)* (IEEE, 2011)
22. D. Berenson, J. Chestnutt, S.S. Srinivasa, J.J. Kuffner, S. Kagami, Pose-constrained whole-body planning using task space region chains, in *Proceedings of IEEE-RAS International Conference on Humanoid Robots*, 2009
23. A. Ambler, R. Popplestone, Inferring the positions of bodies from specified spatial relationships. Artif. Intell. **6**(2), 157–174 (1975)
24. R. Finkel, R. Taylor, R. Bolles, R. Paul, J. Feldman, AL, a programming system for automation, Computer Science Department, Stanford University, Technical Report CS-456, 1974
25. R. Taylor, The synthesis of manipulator control programs from task-level specifications. Ph.D. dissertation, Computer Science Department, Stanford University, 1976
26. L.I. Lieberman, M.A. Wesley, AUTOPASS: an automatic programming system for computer controlled mechanical assembly. IBM J. Res. Dev. **21**(4), 321–333 (1977)
27. E. Drumwright, V. Ng-Thow-Hing, Toward interactive reaching in static environments for humanoid robots, in *Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2006
28. M. Vande Weghe, D. Ferguson, S.S. Srinivasa, Randomized path planning for redundant manipulators without inverse kinematics, in *Proceedings of IEEE-RAS International Conference on Humanoid Robots*, 2007
29. D. Bertram, J. Kuffner, R. Dillmann, T. Asfour, An integrated approach to inverse kinematics and path planning for redundant manipulators, in *Proceedings of IEEE International Conference on Robotics and Automation (ICRA)*, 2006
30. J. De Schutter, T. De Laet, J. Rutgeerts, W. Decre, R. Smits, E. Aertbelien, K. Claes, H. Bruyninckx, Constraint-based task specification and estimation for sensor-based robot systems in the presence of geometric uncertainty. Int. J. Robot. Res. (IJRR) **26**(5), 433–455 (2007)
31. L. Sciavicco, B. Siciliano, *Modeling and Control of Robot Manipulators*, 2nd edn. (Springer, London, 2000), pp. 96–100
32. Z. Yao, K. Gupta, Path planning with general end-effector constraints: using task space to guide configuration space search, in *Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2005
33. M. Gharbi, J. Cortes, T. Simeon, A sampling-based path planner for dual-arm manipulation, in *2008 IEEE/ASME International Conference on Advanced Intelligent Mechatronics*, July 2008, pp. 383–388
34. J. Cortes, T. Simeon, Sampling-based motion planning under kinematic loop-closure constraints, in *Proceedings of Workshop on the Algorithmic Foundations of Robotics (WAFR)*, 2004
35. J.H. Yakey, S.M. LaValle, L.E. Kavraki, Randomized path planning for linkages with closed kinematic chains. IEEE Trans. Robot. Autom. **17**(6), 951–958 (2001)
36. S. Kajita, F. Kanehiro, K. Kaneko, K. Fujiwara, K. Harada, K. Yokoi, H. Hirukawa, Biped walking pattern generation by using preview control of zero-moment point, in *IEEE International Conference on Robotics and Automation (ICRA)*, 2003
37. T. Bretl, S. Lall, Testing static equilibrium for legged robots. IEEE Trans. Robot. **24**(4), 794–807 (2008)
38. K. Hauser, Fast interpolation and time-optimization with contact. Int. J. Robot. Res. **33**(9), 1231–1250 (2014)
39. S. Caron, Q.C. Pham, Y. Nakamura, Leveraging cone double description for multi-contact stability of humanoids with applications to statics and dynamics, in *Proceedings of Robotics: Science and Systems*, Rome, July 2015
40. D. Prattichizzo, J.C. Trinkle, in *Springer Handbook of Robotics: Grasping*, ed. by B. Siciliano, O. Khatib (Springer Science & Business Media, Berlin, 2008)
41. T. Sugihara, Y. Nakamura, Whole-body cooperative balancing of humanoid robot using COG jacobian, in *Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2002

42. R. Diankov, Automated construction of robotic manipulation programs. Ph.D. dissertation, Carnegie Mellon University, Robotics Institute, Aug 2010
43. D. Katz, Y. Pyuro, O. Brock, Learning to manipulate articulated objects in unstructured environments using a grounded relational representation, in *Robotics Science and Systems (RSS)*, 2008
44. J. Sturm, V. Pradeep, C. Stachniss, Learning kinematic models for articulated objects, in *Proceedings of International Joint Conference on Artificial Intelligence (IJCAI)*, 2009