# Task Space Regions: A Framework for Pose-Constrained Manipulation Planning

Dmitry Berenson[1], Siddhartha Srinivasa[2], and James Kuffner[1]

[1]The Robotics Institute, Carnegie Mellon University, Pittsburgh, PA, 15213, USA
[dberenso, kuffner]@cs.cmu.edu
[2]Intel Labs Pittsburgh, Pittsburgh, PA, 15213, USA
siddhartha.srinivasa@intel.com

## Abstract

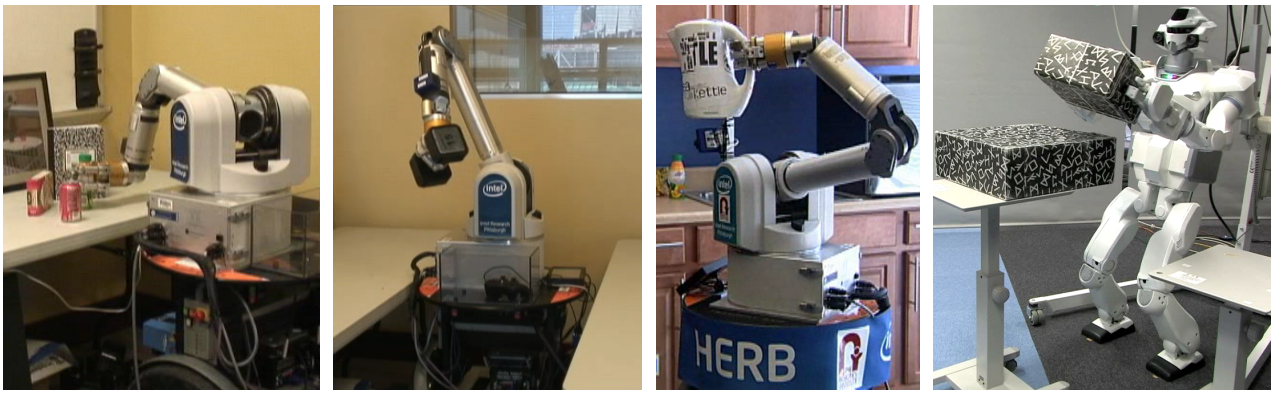
We present a manipulation planning framework that allows robots to plan in the presence of constraints on end-effector pose, as well as other common constraints. The framework has three main components: constraint representation, constraint-satisfaction strategies, and a general planning algorithm. These components come together to create an efficient and probabilistically complete manipulation planning algorithm called the Constrained BiDirectional RRT (CBiRRT2). The underpinning of our framework for pose constraints is our Task Space Regions (TSRs) representation. TSRs are intuitive to specify, can be efficiently sampled, and the distance to a TSR can be evaluated very quickly, making them ideal for sampling-based planning. Most importantly, TSRs are a general representation of pose constraints that can fully describe many practical tasks. For more complex tasks, such as those needed to manipulate articulated objects, TSRs can be chained together to create more complex end-effector pose constraints. TSRs can also be intersected, a property that we use to represent pose uncertainty. We provide a detailed description of our framework, prove probabilistic completeness for our planning approach, and describe several real-world example problems that illustrate the efficiency and versatility of the TSR framework.[1]

## 1 Introduction

Constraints involving the pose of a robot's end-effector are some of the most common constraints in manipulation planning. They arise in tasks such as reaching to grasp an object, carrying a cup of coffee, or opening a door. Although ubiquitous, these constraints present significant challenges for planning algorithms. Because the allowed configurations of the robot are not known a priori, a planning algorithm must discover valid configurations as it plans. In the context of sampling-based motion planning, this exploration is done through sampling. Yet sampling pose constraints in an efficient and probabilistically complete manner is difficult because they can produce lower-dimensional manifolds in the configuration space of the robot. It is unclear how to represent such constraints in a general way, how to ensure that the exploration is efficient, and how to guarantee probabilistic completeness.

[1]This paper unifies and extends previous work appearing in [Berenson et al., 2009d, Berenson et al., 2009c, Berenson et al., 2009b, Berenson et al., 2009a, Berenson and Srinivasa, 2010].

Researchers have developed several algorithms capable of planning with end-effector pose constraints [Stilman, 2007, Koga et al., 1994, Yamane et al., 2004, Yao and Gupta, 2005, Drumwright and Ng-Thow-Hing, 2006, Bertram et al., 2006]. Though often able to solve the problem at hand, these algorithms suffer from problems with efficiency [Stilman, 2007], probabilistic incompleteness [Koga et al., 1994, Yamane et al., 2004, Yao and Gupta, 2005], and overly-specialized constraint representations [Drumwright and Ng-Thow-Hing, 2006, Bertram et al., 2006].




(a) (b) (c) (d)

Figure 1: The HERB and HRP3 robots executing paths planned using our pose-constrained manipulation planning framework.

We present a manipulation planning framework that allows robots to plan in the presence of pose constraints. The framework has three main components: an intuitive constraint representation, constraint-satisfaction strategies, and a general planning algorithm. These three components come together to create an efficient and probabilistically complete manipulation planning algorithm called the Constrained BiDirectional RRT (CBiRRT2). The underpinning of our framework for pose-related constraints is our Task Space Regions (TSRs) representation. TSRs are intuitive to specify, can be efficiently sampled, and the distance to a TSR can be evaluated very quickly, making them ideal for sampling-based planning. Most importantly, TSRs are a general representation of pose constraints that can describe many practical tasks. TSRs can also be chained to describe constraints on articulated objects and intersected to address pose uncertainty. CBiRRT2 explores the constraints of a given problem by sampling from TSRs and TSR Chains using projection methods and direct sampling of parameters. These samples are used to determine goals for the planner and to construct paths on the manifold of configurations allowed by the constraints.

Our constrained manipulation planning framework also allows planning with multiple simultaneous constraints. For instance, collision, torque, and balance constraints can be included along with multiple constraints on end-effector pose. Closed-chain kinematics constraints can also be included as a relation between end-effector pose constraints without requiring specialized projection operators [Yakey et al., 2001] or sampling algorithms [Cortes and Simeon, 2004].

We have applied our framework to a wide range of problems for several robots (Figure 1), both in simulation and in the real world. These problems include grasping in cluttered environments, lifting heavy objects, two-armed manipulation, and opening doors, to name a few. Despite this wide range of problems, our approach only requires three parameters, all of which are constant across the examples described in this paper. We also provide a method for shortening paths generated by our planner, an important component for producing practical paths with sampling-based planners. Finally, since we are operating in the sampling-based planning paradigm, our constraint representations and constraint-satisfaction strategies can be used by other planners such as Probabilistic Roadmaps (PRMs) [Kavraki et al., 1996].

In the following sections, we first discuss previous work related to our approach (Section 2). We then formulate the constrained path planning problem and discuss three strategies for planning with constraints on configuration (Section 3). Section 4 presents TSRs, which can represent constraints on end-effector poses and goals. We then extend this representation to handle more complex constraints by chaining TSRs together (Section 5). Section 6 describes the CBiRRT2 planner, which is capable of planning with TSRs and TSR Chains, among other constraints. Section 7 describes several example problems and shows how to formulate constraints for these problems using TSRs. The performance of CBiRRT2 is also

evaluated on each example problem. We then describe an extension to the TSR framework that allows planning with object pose uncertainty in Section 8. Finally, we provide a summary of the proof of probabilistic completeness for a class of planning algorithms which includes CBiRRT2 in Appendix A.

## 2 Background

Our framework builds on several developments in control theory and motion planning research. Some of the most successful methods in control theory for manipulation have come from local controllers that seek to minimize a given function in the neighborhood of the robot's current configuration through gradient-descent. For instance, controllers have been developed for balancing two-legged robots [Sugihara and Nakamura, 2002], placing the end-effector somewhere in task space [Khatib, 1987], and collision-avoidance [Sentis and Khatib, 2005]. The technique of recursive null-space projection [Sentis and Khatib, 2005] can satisfy multiple constraints simultaneously by prioritizing the constraints and satisfying lower-priority constraints in the null-space of higher-priority ones. These controllers can succeed or fail depending on the prioritization of constraints and it is unclear which of the multiple simulatenous constraints should be prioritized ahead of which others. Even if the prioritization issue is resolved, controllers based on gradient-descent only guarantee that a local minimum of the function is found, which may not be sufficient to solve the problem.

Researchers in control theory have also pursued more global solutions, such as building control policies through dynamic programming [Bellman, 1957]. However such approaches do not scale to the high-dimensional configuration spaces of most manipulators because of the exponential cost of computing optimal policies. Another popular approach has been to learn policies from demonstration [Atkeson and Schaal, 1997, Bentivegna et al., 2004, Howard et al., 2008]. However, for our scope of manipulation problems, finding a set of examples that spans the space of tasks the manipulator is expected to perform as well as a robust way to generalize from these examples has proven quite difficult.

In motion planning, a number of efficient sampling-based planning algorithms have been developed for searching high-dimensional C-spaces [LaValle and Kuffner, 2000, Kavraki et al., 1996]. Sampling-based planners are designed to explore the space of solutions efficiently, without the exhaustive computation required for dynamic programming and without being trapped by local minima like gradient-descent controllers. This global planning ability is acutely important for manipulation because even the most common constraints (like collision-avoidance) can trap approaches based on gradient-descent. Yet sampling-based planners falter when the constraints of a problem restrict the valid configurations to a lower-dimensional manifold in the C-space. It is unclear how to sample such manifolds efficiently and how to ensure probabilistic completeness.

Our strategy is to merge the best aspects of control theory and sampling-based planning to produce an algorithm that searches globally while satisfying constraints locally. The CBiRRT2 planner uses a bi-directional RRT to explore the constraint manifold while enforcing constraints via rejection sampling and projection methods based on gradient-descent. Although CBiRRT2 is based on the Rapidly-exploring Random Tree (RRT) algorithm [LaValle and Kuffner, 2000], it is possible to adapt some of the ideas and techniques in this paper to other search algorithms such as the PRM [Kavraki et al., 1996]. We selected RRTs for their ability to explore C-space while retaining an element of "greediness" in their search for a solution. The greedy element is most evident in the bidirectional version of the RRT algorithm (BiRRT), where two trees, one grown from the start configuration and one grown from the goal configuration, take turns exploring the space and attempting to connect to each other. In this paper, we demonstrate that such a search strategy is also effective for motion planning problems involving pose constraints when it is coupled with the proper strategies for handling those constraints.

The task of the CBiRRT2 planner is to construct a C-space path that lies on the constraint manifolds induced by pose constraints (as well as constraints like collision-avoidance, balance, and torque). We distinguish this task from the task of tracking pre-scripted end-effector paths [Seereeram and Wen, 1995, Oriolo et al., 2002, Oriolo and Mongillo, 2005] because we do not assume an end-effector path is given. CBiRRT2 uses gradient-descent inverse-kinematics techniques [Sentis and Khatib, 2005, Sciavicco and Siciliano, 2000] to meet pose constraints and sample goal configurations. The algorithm plans in the full C-space of the robot, which implicitly allows it to search the null-space of pose constraints, unlike

task-space planners [Koga et al., 1994, Yamane et al., 2004, Yao and Gupta, 2005], which assign a single configuration to each task-space point (from a potentially infinite number of possible configurations). Exploration of the null-space is necessary for probabilistic completeness and can be useful for satisfying other constraints, such as avoiding obstacles or maintaining balance, though our constraint representation could be incorporated into task-space planners as well.

Algorithms similar to CBiRRT2 have been proposed by [Yakey et al., 2001] and [Stilman, 2007].[Yakey et al., 2001] proposed using Randomized Gradient Descent (RGD) to meet closed-chain kinematics constraints. RGD uses random-sampling of the C-space to iteratively project a sample towards an arbitrary constraint [Yao and Gupta, 2005]. Though [Yakey et al., 2001] showed how to incorporate RGD into a sampling-based planner and their method is quite general, it requires significant parameter-tuning and they dealt only with closed-chain kinematic constraints, which are a special case of the pose constraints used in this paper. Furthermore, [Stilman, 2007] showed that when RGD is extended to work with more general pose constraints it is significantly less efficient than Jacobian pseudo-inverse projection and it is sometimes unable to meet more stringent constraints. Our approach is similar to [Stilman, 2007] in that we use an RRT-based planner with Jacobian pseudo-inverse projection, however we differ in the specifics of the planning algorithm and use TSRs, which are a more general constraint representation.

Another key feature of CBiRRT2 is that it can sample TSRs to produce goal configurations. Other researchers have approached the problem of ambiguous goal specification by sampling some number of goals before running the planner [Stilman et al., 2007, Hirano et al., 2005], which limits the planner to a small set of solutions from a region which is really continuous. Another approach is to bias a single-tree planner toward the goal regions, however this approach usually considers single points [Drumwright and Ng-Thow-Hing, 2006, Vande Weghe et al., 2007] in the task space or is hand-tuned for specific goal regions [Bertram et al., 2006].

# 3 Constraints on Configuration

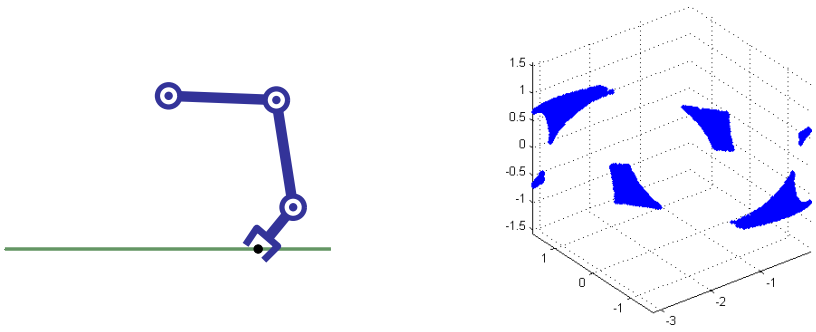
Depending on the robot and the task, many types of constraints can limit a robot's motion. This paper focuses on scleronomic holonomic constraints, which are time-invariant constraints evaluated at a given configuration of the robot. Let the configuration space of the robot be $\mathcal{Q}$. A path in 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 $\tau$. The location of $q$ in $\tau$ 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*. $C(q)$ determines whether the constraint is met at that $q$ and $s \subseteq [0,1]$ is the *domain* of this constraint, i.e. where in the path $\tau$ the constraint is active. To say that a given constraint is satisfied we require that $C(q) = 0 \quad \forall q \in \tau(s)$. 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

$$\mathcal{M}_C = \{q \in \mathcal{Q} : C(q) = 0\} \tag{1}$$

In order for $\tau$ 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 $\tau$ 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:

$$\text{find } \tau : q \in \mathcal{M}_{\mathcal{C}_i} \quad \begin{array}{l} \forall q \in \tau(\mathcal{S}_i) \\ \forall i \in \{1...n\} \end{array} \tag{2}$$




(a) (b)

Figure 2: (a) Pose constraint for a 3-link manipulator: The end-effector must be on the line with an orientation within $\pm 0.7$rad of downward. (b) The manifold induced by this constraint in the C-space of this robot.

Note that the domains of two or more constraints may overlap in which case an element of $\tau$ may need to lie within two or more constraint manifolds.

### 3.1 Difficulties of Constrained Path Planning

Two main issues make solving the constrained path planning problem difficult. First, constraint manifolds are difficult to represent. There is no known analytical representation for many types of constraint manifolds (including pose constraints) and the high dimensional C-spaces of most practical robots make representing the manifold through exhaustive sampling 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 nonsmooth (see Figure 2). 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 pose constraints, 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 used 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.

We address the first issue by using a sampling-based planner that explores the constraint manifold in the C-space (not in the parameter space). This planner uses a variety of sampling techniques to generate samples on constraint manifolds (Section 3.2). One of these techniques is able to sample lower-dimensional constraint manifolds and we validate the probabilistic completeness of this approach in Appendix A, thus addressing the second issue.

### 3.2 Sampling on Constraint Manifolds

In order to solve the constrained path planning problem, a sampling-based planning algorithm must be able to generate configurations that lie on constraint manifolds. We describe three general strategies for generating these configurations: rejection, projection, and direct sampling (see Figure 3).

In the *rejection* strategy, we simply generate a 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

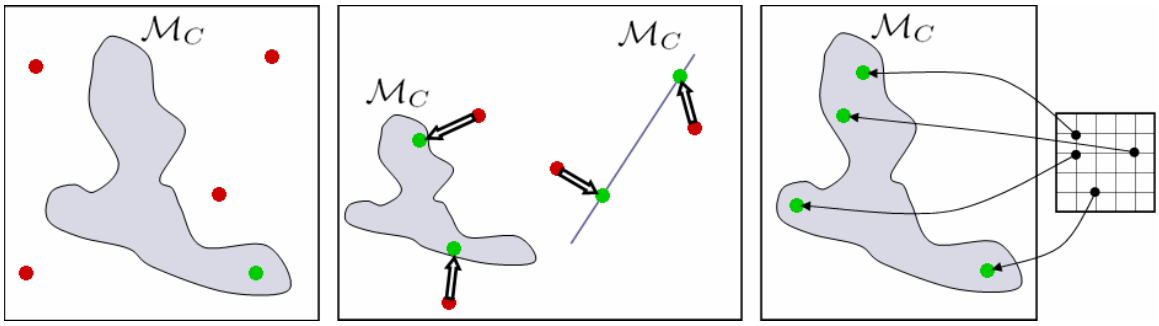



Figure 3: The three sampling strategies used in our framework. Red dots represent invalid samples and green dots represent valid ones. (Left) Rejection sampling (Center) Projection sampling (Right) Direct sampling from a parameterization of the constraint

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 $q_0 \in \mathcal{Q}$ 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 $C(q_i) = 0$. 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. 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 we will use this strategy only when sampling goals for our planner, not to build paths. We will describe how to do direct sampling with our constraint representation in Section 4.

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 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.

# 4 Task Space Regions

We now focus on a specific constraint representation that we have developed for planning paths for manipulators with end-effector 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 (see Figure 1d). 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. We propose a novel framework for pose-constrained manipulation planning which is capable of trading off constraints and affordances to produce manipulation

plans for high degree of freedom robots, like humanoids or mobile manipulators.

Our constrained manipulation planning framework uses a novel unifying representation of constraints and affordances which we term Task Space Regions (TSRs). 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. We have also developed a more complex representation for articulated constraints called TSR Chains, which is discussed in Section 5. Finally, we discuss the limitations of these representations in Section 9.

## 4.1 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 $3 \times 3$ rotation matrix $\mathbf{R}_b^a$ and a $3 \times 1$ translation vector $\mathbf{t}_b^a$.

$$\mathbf{T}_b^a = \begin{bmatrix} \mathbf{R}_b^a & \mathbf{t}_b^a \\ \mathbf{0} & 1 \end{bmatrix} \tag{3}$$

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 $w$
- $\mathbf{B}^w$: $6 \times 2$ matrix of bounds in the coordinates of $w$:

$$\mathbf{B}^w = \begin{bmatrix} x_{min} & x_{max} \\ y_{min} & y_{max} \\ z_{min} & z_{max} \\ \psi_{min} & \psi_{max} \\ \theta_{min} & \theta_{max} \\ \phi_{min} & \phi_{max} \end{bmatrix} \tag{4}$$

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 Figure 4 and Figure 5.

## 4.2 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 Figure 5). 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 the pose of the grasp location or the pose of the object held by the hand in world coordinates.

$$\mathbf{T}_{s'}^0 = \mathbf{T}_s^0 (\mathbf{T}_e^w)^{-1} \tag{5}$$

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

$$\mathbf{T}_{s'}^w = (\mathbf{T}_w^0)^{-1} \mathbf{T}_{s'}^0 \tag{6}$$

Now we convert the transform $\mathbf{T}_{s'}^w$ into a $6 \times 1$ 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 $\mathbf{B}^w$.

$$d^w = \begin{bmatrix} \mathbf{t}_{s'}^w \\ \arctan 2(\mathbf{R}_{s'_{32}}^w, \mathbf{R}_{s'_{33}}^w) \\ -\arcsin(\mathbf{R}_{s'_{31}}^w) \\ \arctan 2(\mathbf{R}_{s'_{21}}^w, \mathbf{R}_{s'_{11}}^w) \end{bmatrix} \tag{7}$$

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

$$\Delta\mathbf{x}_i = \begin{cases} d_i^w - \mathbf{B}_{i,1}^w & \text{if } d_i^w < \mathbf{B}_{i,1}^w \\ d_i^w - \mathbf{B}_{i,2}^w & \text{if } d_i^w > \mathbf{B}_{i,2}^w \\ 0 & \text{otherwise} \end{cases} \tag{8}$$

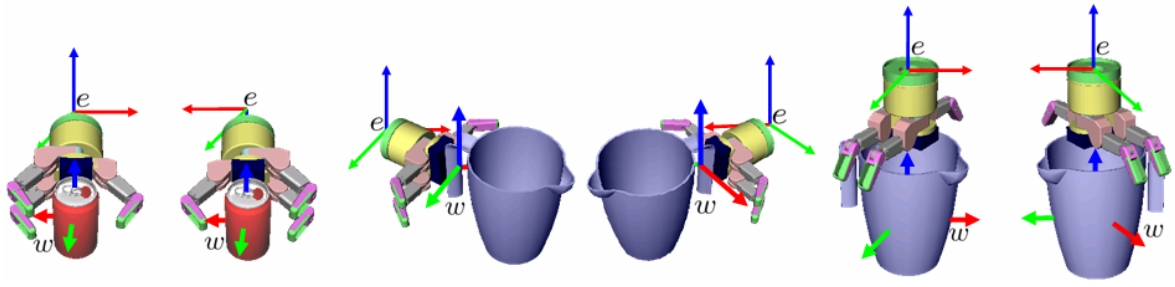



Figure 4: The $w$ and $e$ frames used to define end-effector goal TSRs for a soda can and a pitcher.

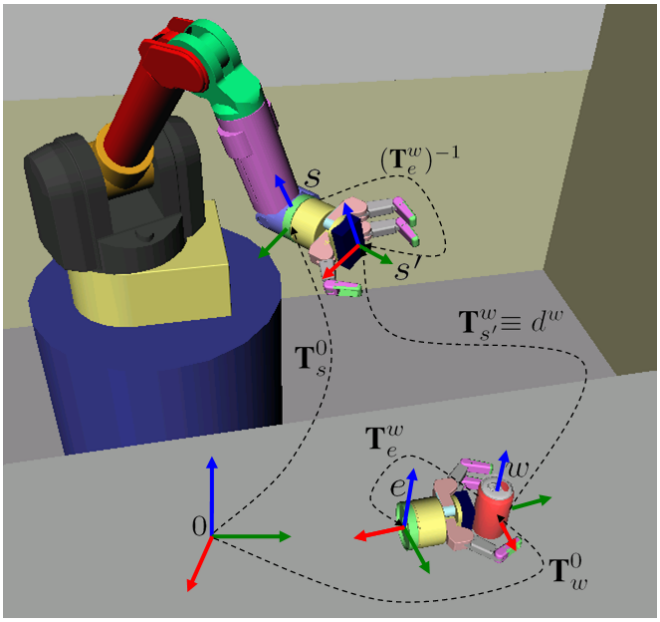



Figure 5: Transforms and coordinate frames involved in computing the distance to TSRs. The robot is in a sample configuration which has end-effector transform $s$ and the hand near the soda can at transform $e$ represents the $\mathbf{T}_e^w$ defined by the TSR.

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.

### 4.3 Direct Sampling of TSRs

When using TSRs to specify goal end-effector poses, it will be necessary to draw 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 $d_{sample}^w$ and converted into the transformation $\mathbf{T}_{sample}^w$. We can then convert this sample into world coordinates after applying the end-effector transformation.

$$\mathbf{T}_{sample'}^0 = \mathbf{T}_w^0 \mathbf{T}_{sample}^w \mathbf{T}_e^w \tag{9}$$

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.

$$\zeta_i = \sum_{j=1}^{6} \left( \mathbf{B}_{j,2}^{w_i} - \mathbf{B}_{j,1}^{w_i} \right) \tag{10}$$

where $\zeta_i$ and $\mathbf{B}^{w_i}$ are the weight and bounds of the $i$th TSR, respectively. Sampling proportional to $\zeta_i$ 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 non-zero probability of selecting any TSR.

## 4.4 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:

$$\{C(q) = \text{DistanceToTSR}(q), \quad s = [1]\} \tag{11}$$

Where the DistanceToTSR function implements the method of Section 4.2 and $s$ refers to the domain of the constraint (Section 3).

To generate valid configurations in the $\mathcal{M}_C$ corresponding to this constraint, we can use direct sampling of TSRs (Section 4.3) and pass the sampled pose to an IK solver to generate a valid configuration. In order to ensure that we don't 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.

## 4.5 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 Equation 11 in the domain of the constraint:

$$\{C(q) = \text{DistanceToTSR}(q), \quad s = [0, 1]\} \tag{12}$$

Since the domain of this constraint spans the entire path, the planning algorithm must ensure that each configuration it

**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
```

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 ($\mathbf{J}^+$) [Sciavicco and Siciliano, 2000] 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 Equation 8. 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 avoidance, and joint limits when projecting configurations. We will show, in Appendix A, 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.

It is important to note that we can also use the method of Section 4.3 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 [Yao and Gupta, 2005]). However, we prefer the completeness properties of C-space planners, so we focus on those in this paper.

# 5 Task Space Region Chains

While we showed that TSRs are intuitive to specify, 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 closed-chain kinematics and manipulating articulated objects, this section 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 Figure 6). 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 while the door is moving) 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.

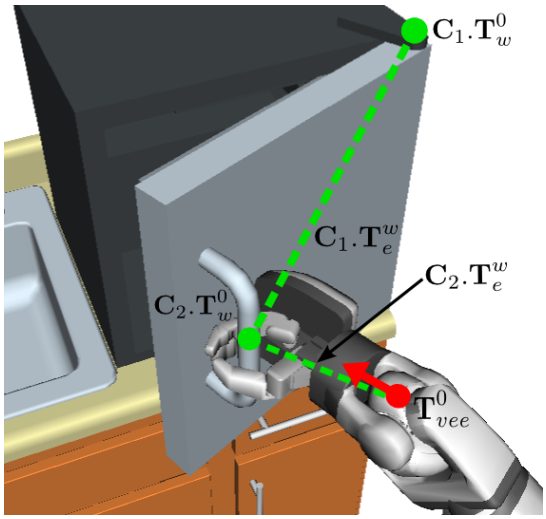



Figure 6: The virtual manipulator for the door example. The green dotted lines represent the links of the virtual manipulator and the red dot and arrow represent the virtual end-effector, which is at transform $\mathbf{T}_{vee}^0$.

## 5.1 TSR Chain Definition

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

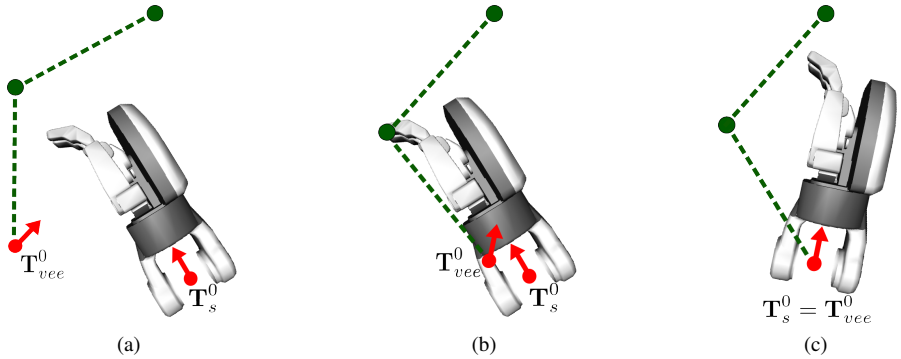
$$\mathbf{C}_i.\mathbf{T}_w^0 = (\mathbf{C}_{i-1}.\mathbf{T}_w^0)(\mathbf{C}_{i-1}.\mathbf{T}_{sample}^w)(\mathbf{C}_{i-1}.\mathbf{T}_e^w) \tag{13}$$

for $i = \{2...n\}$ where $\mathbf{C}_i$ corresponds to the $i$th TSR in the chain and $\mathbf{C}_i.\{\cdot\}$ refers to an element of the $i$th TSR. Of course a TSR Chain can consist of only one TSR, in which case it is identical to a normal TSR. $\mathbf{C}_i.\mathbf{T}_{sample}^w$ can be any transform obtained by sampling from inside the bounds of $\mathbf{C}_i.\mathbf{B}^w$. Thus we do not know $\mathbf{C}_i.\mathbf{T}_w^0$ until we have determined $\mathbf{T}_{sample}^w$ 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 Figure 6). $\mathbf{C}_1.\mathbf{T}_{sample}^w$ 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 $\mathbf{T}_{sample}^w$ 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 $\mathbf{B}^w$.

## 5.2 Direct Sampling From TSR Chains

To directly sample a TSR Chain we first sample from within $\mathbf{C}_1.\mathbf{B}^w$ to obtain $\mathbf{C}_1.\mathbf{T}_{sample}^w$. This is done by sampling uniformly between the bounds in $\mathbf{B}^w$, compiling the sampled values into a displacement $d_{sample}^w = [x\ y\ z\ \psi\ \theta\ \phi]$ and converting that displacement to the transform $\mathbf{C}_1.\mathbf{T}_{sample}^w$. We then use this sample to determine $\mathbf{C}_2.\mathbf{T}_w^0$ via Equation 13. We repeat this process for each TSR in the chain until we reach the $n$th TSR. We then obtain a sample in the world frame:




(a) (b) (c)

Figure 7: 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.

$$\mathbf{T}^0_{sample'} = (\mathbf{C}_n.\mathbf{T}^0_w)(\mathbf{C}_n.\mathbf{T}^w_{sample})(\mathbf{C}_n.\mathbf{T}^w_e) \tag{14}$$

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 $\mathbf{T}^w_{sample}$ 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 $\mathbf{T}^0_{sample'}$ 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.

## 5.3 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 $\mathbf{T}^w_{sample}$ 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 Figure 7a). Finding the correct $\mathbf{T}^w_{sample}$ 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 Figure 7b). Once we obtain the joint angles of the virtual manipulator, we convert them to $\mathbf{T}^w_{sample}$ values and forward-chain to obtain the virtual end-effector position $\mathbf{T}^0_{vee}$. We then convert $\mathbf{T}^0_s$ to the virtual end-effector's frame:

$$\mathbf{T}^{vee}_s = (\mathbf{T}^0_{vee})^{-1}\mathbf{T}^0_s \tag{15}$$

and then convert to the displacement form:

$$d_s^{vee} = \begin{bmatrix} \mathbf{t}_s^{vee} \\ \arctan 2(\mathbf{R}_{s_{32}}^{vee}, \mathbf{R}_{s_{33}}^{vee}) \\ -\arcsin(\mathbf{R}_{s_{31}}^{vee}) \\ \arctan 2(\mathbf{R}_{s_{21}}^{vee}, \mathbf{R}_{s_{11}}^{vee}) \end{bmatrix} \tag{16}$$

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

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 $\mathbf{T}_{vee}^0$ to meet the constraint specified by this TSR Chain (Figure 7c). 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.

## 5.4 Physical Constraints

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 (see Section 6.3).

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 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 it's initial pose to $\mathbf{T}_{vee}^0$ as it is being pulled by the robot to find the resting configuration of the chain.

## 5.5 Notes on Implementation

Whenever we create a TSR Chain, we also create its virtual manipulator in simulation so that we can perform IK on this manipulator and get the location of the virtual end-effector. When we refer to the joint values of a TSR Chain, we are actually referring to the joint values of that TSR Chain's virtual manipulator. Also, to differentiate whether a TSR Chain should be used for sampling goals or constraining configurations or both, we specify how the chain should be used in its definition. When inputing TSR Chains into our planner, we specify which manipulator of the robot they correspond to as well as any physical DOFs that correspond to elements of the chain.

# 6 The CBiRRT2 Algorithm

This section describes the Constrained BiDirectional RRT (CBiRRT2) planner, which is capable of planning with TSR Chains among other constraints. Since the representation of TSR chains subsumes that of TSRs, the planner can incorporate the uses of TSRs already described. In this section we describe the operations of the planner. Several example problems as well as CBiRRT2's performance on these problems are shown in Section 7. We prove the probabilistic completeness of CBiRRT2 when planning with pose constraints in Appendix A.

## 6.1 Planner Operation

CBiRRT2 takes into account constraints on the configuration of the robot during its path as well as constraints on the goal configuration of the robot. Constraints on the poses and goal locations of the robot's end-effectors are specified as TSR Chains.

**Algorithm 2**: CBiRRT2($Q_s$, $Q_g$)

```
1  T_a.Init(Q_s); T_b.Init(Q_g);
2  while TimeRemaining() do
3      T_goal = GetBackwardTree(T_a, T_b);
4      if size(T_goal) = 0 or rand(0,1) < P_sample then
5          AddRoot(T_goal);
6      else
7          q_rand ← RandomConfig();
8          q^a_near ← NearestNeighbor(T_a, q_rand);
9          q^a_reach ← ConstrainedExtend(T_a, q^a_near, q_rand);
10         q^b_near ← NearestNeighbor(T_b, q^a_reached);
11         q^b_reach ← ConstrainedExtend(T_b, q^b_near, q^a_reach);
12         if q^a_reach = q^b_reach then
13             P ← ExtractPath(T_a, q^a_reach, T_b, q^b_reach);
14             return ShortenPath(P);
15         else
16             Swap(T_a, T_b);
17         end
18     end
19 end
20 return ∅;
```

**Algorithm 3**: **ConstrainedExtend**($T$, $q_{near}$, $q_{target}$)

```
1  q_s ← q_near; q_s^old ← q_near;
2  while true do
3      if q_target = q_s then
4          return q_s;
5      else if ||q_target − q_s|| > ||q_s^old − q_target|| then
6          return q_s^old;
7      end
8      q_s^old ← q_s;
9      q_s ← q_s + min(Δq_step, ||q_target − q_s||) (q_target − q_s)/||q_target − q_s||;
10     c ← GetConstraintValues(T, q_s^old);
11     {q_s, c} ← ConstrainConfig(q_s^old, q_s, c, ∅);
12     if q_s ≠ ∅ then
13         T.AddVertex(q_s, c);
14         T.AddEdge(q_s^old, q_s);
15     else
16         return q_s^old;
17     end
18 end
```

CBiRRT2 operates by growing two trees in the C-space of the robot (please follow the explanation below in Algorithm 2). At each iteration, CBiRRT2 chooses between one of two modes: exploration of the C-space using the two trees or direct sampling from a set of TSR Chains. The probability of choosing to sample is defined by the parameter $P_{sample}$.

If the algorithm chooses to sample, it calls the AddRoot function, which tries to inject a goal configuration into the backward tree $T_{goal}$. If the algorithm chooses to explore the C-space, one of the trees grows a branch toward a randomly-sampled configuration $q_{rand}$ using the ConstrainedExtend function. The branch grows as far as possible toward $q_{rand}$ but may be stalled due to collision or constraint violation and will terminate at $q^a_{reach}$. The other tree then grows a branch toward $q^a_{reach}$, again growing as far as possible toward this configuration. If the other tree reaches $q^a_{reach}$, the trees have connected and a path has been found. If not, the trees are swapped and the above process is repeated.

The ConstrainedExtend function (see Algorithm 3) iteratively moves from a configuration $q_{near}$ toward a configuration

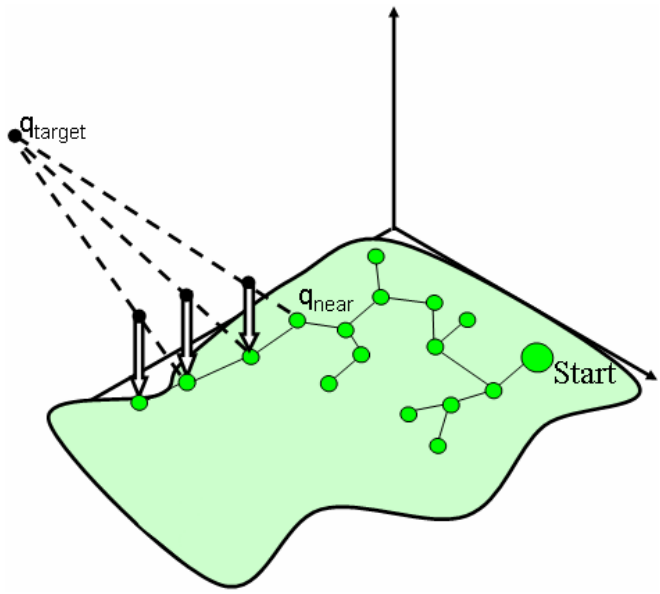



Figure 8: Depiction of one **ConstrainedExtend** operation. The operation starts at $q_{near}$, which is a node of a search tree on the constraint manifold and iteratively moves toward $q_{target}$, which is a configuration sampled from the C-space. Each step toward $q_{target}$ is constrained using the ConstrainConfig function to lie on the constraint manifold.

$q_{target}$ with a step size of $\Delta q_{step}$. After each step toward $q_{target}$, the function checks if the new configuration $q_s$ has reached $q_{target}$ or if it is moving farther from $q_{target}$, in either case the function terminates. If the above conditions are not true then the algorithm takes a step toward $q_{target}$ and passes the new $q_s$ to the ConstrainConfig function, which is problem-specific. If ConstrainConfig is able to project $q_s$ to a constraint manifold, the new $q_s$ is added to the tree and the stepping process is repeated. Otherwise, ConstrainedExtend terminates (see Figure 8 for an illustration). ConstrainedExtend always returns the last configuration reached by the extension operation. The $c$ vector is a vector of TSR Chain joint values of all TSR Chains. Every $q_s$ has a corresponding $c$ which is stored along with $q_s$ in the tree. We store the $c$ vector so that the ConstrainConfig function has a good initial guess of the TSR chain joint values when taking subsequent steps. This greatly decreases the time used by the inverse-kinematics solver inside ConstrainConfig.

After a path is found, we shorten it using the ShortenPath function. This function implements the popular "short-cut" method to iteratively shorten the path. However, instead of using straight lines which would violate constraints, we use the ConstrainedExtend function (Algorithm 4) for each short-cut. Using ConstrainedExtend guarantees that constraints will be met along the shortened path. Also, it is important to note that a short-cut generated by ConstrainedExtend between two nodes is not necessarily the shortest path between them because the nodes may have been projected in an arbitrary way. This necessitates checking whether $\mathrm{Length}(P_{shortcut})$ is shorter than the original path between $i$ and $j$.

Note that CBiRRT2 can also be seeded with multiple start and goal configurations ($Q_s$/$Q_g$). If no goals are specified, the AddRoot function will insert the first goal into the backward tree. In fact, the AddRoot function can be called for both the start and the goal trees, if this is desired.

## 6.2 Planning with TSR Chains

Accounting for TSR Chains is done in the AddRoot and ConstrainConfig functions. When CBiRRT2 chooses to sample a goal configuration, it calls the AddRoot function (see Algorithm 5). This function retrieves the relevant set of TSR Chains for each manipulator and samples a target transform for each manipulator using the SampleFromTSRChain function, which is an implementation of the methods described in Section 5.2. It then forms an initial guess of the robot's joint values and

**Algorithm 4**: ShortenPath($P$)

```
1  while TimeRemaining() do
2      T_shortcut ← {};
3      i ← RandomInt(1, size(P) − 1);
4      j ← RandomInt(i, size(P));
5      q_reach ← ConstrainedExtend(T_shortcut, P_i, P_j);
6      if q_reach = P_j and
       Length(T_shortcut) < Length(P_i ··· P_j) then
7          P ← [P_1 ··· P_i, T_shortcut, P_{j+1} ··· P.size];
8      end
9  end
10 return P;
```

**Algorithm 5**: AddRoot($T$)

```
1  for i = 1...m do
2      ℂ ← GetTSRChainsForManipulator(i);
3      {T^0_targ, c} ← SampleFromTSRChains(ℂ);
4      Targets.AddTarget(T^0_targ, i);
5  end
6  {q_s, c} ← GetInitialGuess();
7  {q_s, c} ← ConstrainConfig(∅, q_s, c, Targets);
8  if q_s ≠ ∅ then
9      T.AddVertex(q_s, c);
10 end
```

**Algorithm 6**: ConstrainConfig($q_s^{old}$, $q_s$, $c$, Targets)

```
1  CheckDist = False;
2  if Targets = ∅ then
3      CheckDist = True;
4      for i = 1...m do
5          ℂ ← GetTSRChainsForManipulator(i);
6          T^0_s ← GetEndEffectorTransform(q_s, i);
7          {T^0_targ, c} ← GetClosestTransform(ℂ, T^0_s, c);
8          Targets.AddTarget(T^0_targ, i);
9      end
10 end
11 q_s ← UpdatePhysicalConstraintDOF(q_s, c);
12 q_s ← ProjectConfig(q_s, Targets);
13 if (q_s = ∅ or
14     (CheckDist and |q_s − q_s^old| > 2Δq_step)) then
15     return ∅;
16 end
17 return {q_s, c};
```

$c$ and calls the ConstrainConfig function. In practice we usually use the initial configuration of the robot and vector of zeros for $c$ as the guess but these can be randomized as well. If the ConstrainConfig does not return $\emptyset$, the resulting $q_s$ and corresponding $c$ are added to the tree.

The ConstrainConfig function is problem-specific, an example of a ConstrainConfig function that considers *only* TSR Chains is given in Algorithm 6. If ConstrainConfig is not passed a set of targets (i.e. it is called from ConstrainedExtend instead of AddRoot), then it generates a set of targets for each manipulator using the GetClosestTransform function, which is an implementation of the methods described in Section 5.3. Note that this function also updates the $c$ vector with the joint values of the TSR Chain that generated the closest transform. The $c$ values for the TSR Chains that did not yield the closest transform to $\mathbf{T}_s^0$ are not updated. After the target transforms for each manipulator are obtained, ProjectConfig projects the configuration of the robot using standard inverse-kinematics algorithms based on the Jacobian pseudo-inverse to produce a $q_s$ which meets the constraints represented by the TSR Chains. This completes the IK handshaking process described in Section 5.3.

If ConstrainConfig was called by AddRoot, the distance between $q_s^{old}$ and $q_s$ is unimportant. However, we do not wish for $q_s$ to be too far from $q_s^{old}$ when extending using ConstrainedExtend because the intermediate configurations are not likely to meet the constraints. Thus we enforce a small step size to reduce deviation from constraints between nodes.

In most situations, we are also interested in satisfying other constrains such as balance and collision using the rejection strategy. Checks for these constraints should be inserted at line 14 of ConstrainConfig.

### 6.3 Augmenting Configuration with States of Physical DOF

Because TSR chains can specify constraints corresponding to physical degrees of freedom of objects in the world (such as the hinge of a door) as well as purely virtual constraints, we have to account for physical DOF when checking collision and measuring distances in the C-space. To achieve this, we include the configuration of all physical DOFs in the configuration vector $q$. We set these DOF by extracting their values from the vector of all the TSRChains' virtual manipulator joint values $c$ using the UpdatePhysicalConstraintDOF function. This is done on line 11 of the ConstrainConfig function. Note that these DOF are not affected by the ProjectConfig function.

### 6.4 Parameters

One of the strengths of CBiRRT2 is that it uses only three parameters, all of which require minimal tuning. The first parameter encodes the RRT step size $\Delta q_{step}$. $\Delta q_{step}$ can be increased to speed up planning or decreased to allow finer motions but we have found that tuning this parameter is rarely necessary for the manipulation tasks we consider.

The numerical error allowed in meeting a pose constraint $\epsilon$ (in Algorithm 1) is necessitated by the numerical nature of our projection operator. Our projection method is quite accurate, so CBiRRT2 performs well even for small values of $\epsilon$.

Finally, the third parameter $P_{sample}$ is only used when goal sampling is required. A higher $P_{sample}$ biases CBiRRT2 toward goal sampling, a lower one biases it toward building paths. We showed in [Berenson et al., 2009c] that the algorithm performs well for a wide range of values for $P_{sample}$, though we recommend setting the value to be low because building paths usually requires more computation than sampling an adequate goal in our problem domain.

## 7 Example Problems

This section describes six example problems and the constraints specified for those problems as well as results for running CBiRRT2 in simulation and experiments on physical hardware. These examples illustrate the use of TSRs and TSR Chains for common problems in manipulation planning. They are also meant to show the wide range of problems and robots which can be handled by the TSR framework.

The first three examples are implemented on a 7DOF Barrett WAM and the last three on 28DOF of the HRP3 humanoid. Since the TSR Chain representation subsumes the TSR representation, each problem can be implemented using TSR Chains. However we do not describe a chained implementation when only chains of length one are used so that the explanation is clearer. Unless otherwise noted, we use the ConstrainConfig function of Algorithm 6 and include collision and balance (for HRP3) constraints on line 14 so all paths produced by the planner are guaranteed to be collision-free and quasi-statically balanced. We set the allowable error for meeting a constraint to $\epsilon = 0.001$ and the RRT stepsize $\Delta q_{step} = 0.05$. $P_{sample}$ is only used in the first and fifth examples, where its value is $0.1$. All experiments were performed on a 2.4 GHz Intel CPU with 4 GB of RAM.

### 7.1 Clearing a Table

In this problem the robot's task is to clear a table (see Figure 9). Each object on the table has a set of TSRs associated with it, similar to the TSRs defined in Figure 4. For the soda can and juice bottle we define allowable grasp poses similar to those used for the soda can in Figure 4. For the notebook and rice box, we define a TSR for each side of the object with the hand facing that side. Note that we do not specify which object to grasp, we simply input all the TSRs for all the objects

Figure 9: Snapshots from three runs of three trajectories planned using CBiRRT2. Top Row: Grasping and throwing away a box of rice. Middle Row: Grasping and throwing away a juice bottle. Bottom Row: Grasping and throwing away a soda can.

into the planner and execute the first path the planner returns. After the robot grasps one of the objects, we plan a path to throw it away using a TSR placed over the recycling bin. This TSR allows translation freedom over the top of the bin and full rotation freedom. Snapshots from the execution of this task are shown in Figure 9. The objects were recognized and localized using a vision algorithm based on SIFT feature matching [Collet et al., 2009]. Seven objects were picked up and dropped into the recycling bin. The average time for planning a reaching path was 3.50s and the average planning time for planning a path to the recycling bin was 2.44s (all runs were successful).

## 7.2 The Maze Puzzle

In this problem, the robot must solve a maze puzzle by drawing a path through the maze with a pen (see Figure 10). 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 $\alpha$ 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 $\mathbf{B}^w$ is

$$\mathbf{B}^w = \begin{bmatrix} -\infty & \infty \\ -\infty & \infty \\ 0 & 0 \\ -\alpha & \alpha \\ -\alpha & \alpha \\ -\pi & \pi \end{bmatrix} \tag{17}$$

This example is meant to demonstrate that CBiRRT2 is capable of solving multiple narrow passage problems while still moving on a constraint manifold. It is also meant to demonstrate the generality of CBiRRT2; no special-purpose planner is needed even for such a specialized task.

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 7.2 represent the average of 10 runs for different $\alpha$ values. Runtimes with a "$>$" denote that there was at least one run that did not terminate before 120 seconds. For such runs, 120 was used in computing the average. No goal sampling is performed in this example.

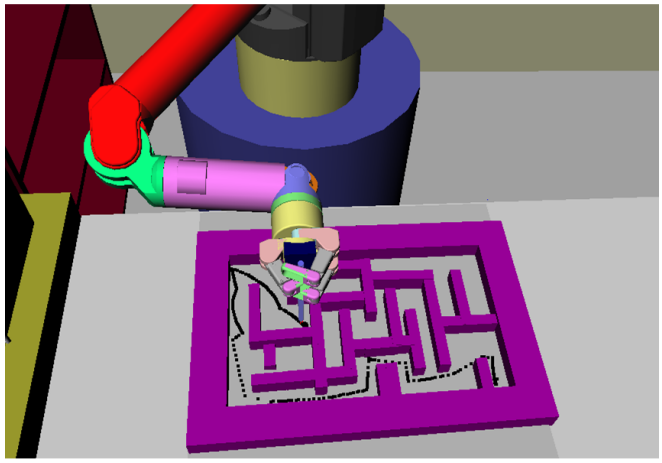


Figure 10: A trajectory found for the Maze Puzzle using $\alpha = 0.4$rad. The black points represent positions of the tip of the pen along the trajectory.

| $\alpha$(rad.) | 0.0 | 0.1 | 0.2 | 0.3 | 0.4 | 0.5 |
|---|---|---|---|---|---|---|
| Avg. Runtime(s) | >83.5 | >58.8 | >49.0 | 19.5 | 14.3 | 15.2 |
| Success Rate | 40% | 60% | 90% | 100% | 100% | 100% |

TABLE 7.2: SIMULATION RESULTS FOR MAZE PUZZLE

The shorter runtimes and high success rates for larger $\alpha$ 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. 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 Figure 10 for an example trajectory of the tip of the pen.

## 7.3 Heavy Object with Sliding Surfaces

In this problem the task is to move a heavy dumbbell from a start position to a given goal position (see Figure 12). The weight of the dumbbell is known but we do not know what configurations allow acceptable torques a priori. Sliding surfaces are also provided so that the planner may use these to support the object if necessary. The planner is allowed to slide the object along a sliding surface or to hold the object if the torques in the holding configuration are within torque limits. The constraint on torque is formulated as:

$$\left\{ C(q) = \begin{cases} 1 & \text{if InTorqueLimits}(q) \\ 0 & \text{otherwise} \end{cases}, \quad s = [0, 1] \right\} \tag{18}$$

We employ the rejection strategy with respect to torque constraints, so we need to calculate the torques on the joints in a given $q$. This is done using standard Recursive Newton-Euler techniques [Walker and Orin, 1982]. We will refer to the total end-effector mass as $m$. Note that this formulation only takes into account the torque necessary to maintain a given $q$, i.e. it assumes the robot's motion is quasi-static. The end-effector frame is defined to be at the bottom of the dumbbell.

Each sliding surface is a rectangle of known width and length with an associated surface normal. In general, the surfaces may be slanted so they may only support part of the objects's weight, which is taken into account when calculating joint torques. Each sliding surface gives rise to a constraint manifold and there can be any number of sliding surfaces. To

represent the sliding surfaces, we define pose constraint TSRs at the centers of each sliding surface with $\mathbf{T}_w^0$ at the center with the z axis oriented normal to the surface and $\mathbf{T}_e^w$ being identity. The $\mathbf{B}^w$ for each of these TSRs is:

$$\mathbf{B}^w = \begin{bmatrix} -\text{length}/2 & \text{length}/2 \\ -\text{width}/2 & \text{width}/2 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ -\pi & \pi \end{bmatrix} \tag{19}$$

The ConstrainConfig function for this example differs slightly from the one in Algorithm 6. Instead of always satisfying one of the pose constraint TSR, the ConstrainConfig function for this example first checks if the configuration meets the torque constraint and if it does not, attempts to satisfy the closest pose constraint TSR in the same way as Algorithm 6 (see Figure 11). Finally, ConstrainConfig for this example checks if the projected configuration supports enough weight to ensure the torque constraint is met.

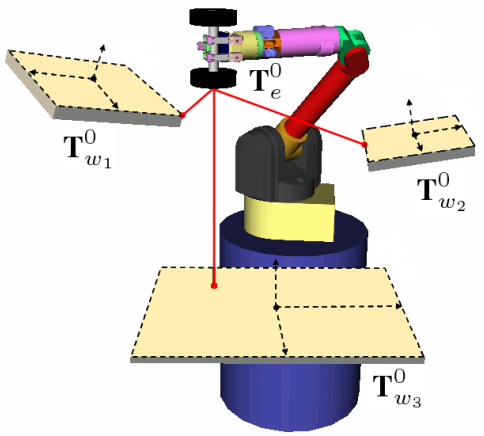



Figure 11: Finding the closest pose constraint TSR within ConstrainConfig. The shortest distance from $\mathbf{T}_e^0$ to $\mathbf{T}_{w_i}^0$ (computed using the DistanceToTSR function of Section 4.2) determines which TSR is chosen for projection.

We generate $Q_s$ and $Q_g$ the same way as in the Maze Puzzle. The values in Table 7.3 represent the average of 10 runs for different weights of the dumbbell. The weight of the dumbbell was increased until the algorithm could not find a path within 120 seconds for one of the 10 runs. No goal sampling is performed in this example.

| Weight | 7kg | 8kg | 9kg | 10kg | 11kg | 12kg | 13kg | 14kg |
|---|---|---|---|---|---|---|---|---|
| Avg. Runtime(s) | 1.89 | 2.06 | 3.84 | 5.51 | 7.29 | 12.4 | 27.5 | >53.9 |
| Success Rate | 100% | 100% | 100% | 100% | 100% | 100% | 100% | 80% |

TABLE 7.3: SIMULATION RESULTS FOR HEAVY OBJECT SLIDING

The shorter runtimes and higher success rates for lower weights of the dumbbell match our expectations about the constraints induced by torque limits. As the dumbbell becomes heavier, the manifold of configurations with valid torque becomes smaller and thus finding a path through this manifold becomes more difficult.

We also implemented this problem on our physical WAM robot. Snapshots from three trajectories for three different weights are shown in Figure 12. As with the simulation environment, the robot slid the dumbbell more when the weight was heavier and sometimes picked up the weight without any sliding for the mass of 4.98kg. Note that we take advantage of the compliance of our robot to help execute these trajectories but in general such trajectories should be executed using an appropriate force-feedback controller.

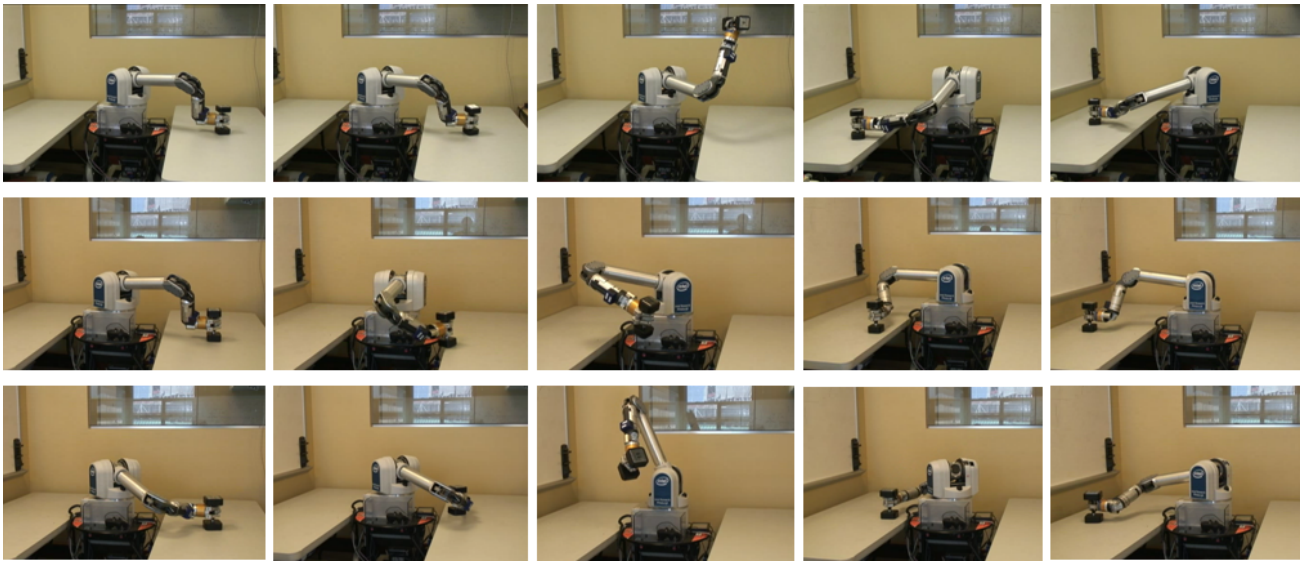


Figure 12: Experiments on the 7DOF WAM arm for three dumbbells. Top Row: $m = 4.98$kg. Middle Row: $m = 5.90$kg, and Bottom Row: $m = 8.17$kg. The trajectory for the lightest dumbbell requires almost no sliding, whereas the trajectories for the heavier dumbbells slide the dumbbell to the edge of the table.

## 7.4 Closed Chain Kinematics

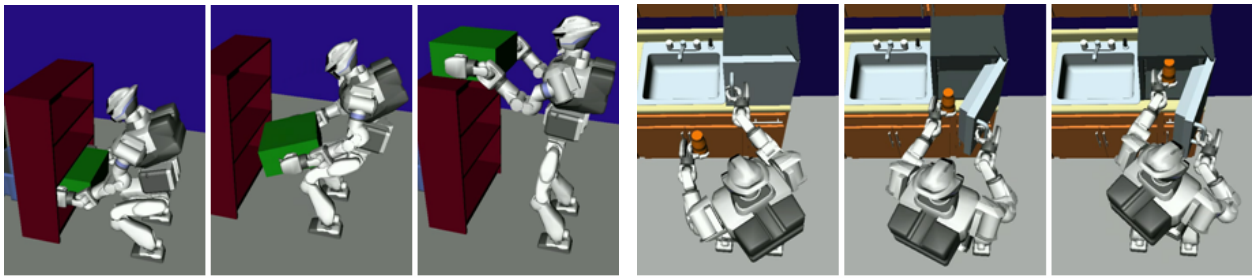
One of the major research areas in two-arm manipulation is the handling of closed chain kinematics constraints. Some researchers approach the problem of planning with closed-chain kinematics by implementing specialized projection operators [Yakey et al., 2001] or sampling algorithms [Cortes and Simeon, 2004]. However, in the TSR framework no special additions are required. In fact, closed chain kinematics can be enforced using straightforward TSR definitions.

Consider the problem shown in Figure 13a. The task for the HRP3 humanoid is to pick up a box from the bottom of the bookshelf and place it on top. Note that there are two closed chains which must be enforced by the planner; the legs and arms form two separate loops. We root the kinematic tree of the robot at the right foot, though the floating-base formulation can also be used.

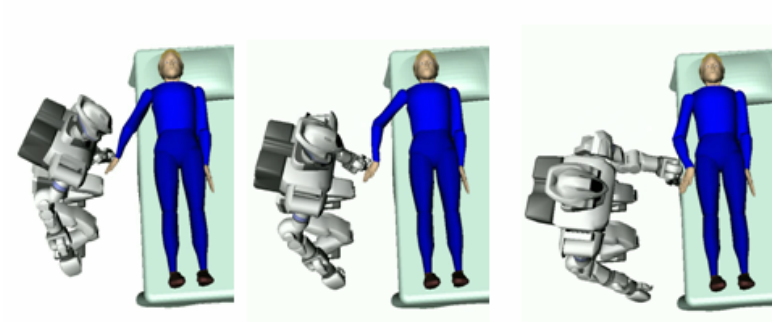
We define three pose constraint TSRs. The first TSR is assigned to the left leg of the robot and allows no deviation from the current left-foot location (i.e. $\mathbf{B}^w = \mathbf{0}_{6\times 2}$). The second and third TSRs are assigned to the left and right arms and are defined relative to the location 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. We get the goal configuration of the robot from inverse kinematics on the target box position; no goal sampling is performed in this example.

The result of this construction is the following: When ConstrainedExtend generates a new $q_s$, the box moves with the right hand and the frame of the box changes thus breaking the closed-chain constraint. This $q_s$ is passed to ConstrainConfig, which projects $q_s$ to meet the constraint (i.e. moving the left arm). The right hand is constrained to not deviate from its pose in $q_s$ by its TSR Chain, which ensures that the box does not move during the projection operation. The same process happens simultaneously for the left leg of the robot as well.

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 7.6. On the real robot, the task was to stack two boxes in succession, snapshots from the execution of the plan can be seen in Figure 14. The experiments on the robot show that we can enforce stringent closed-chain constraints using the CBiRRT2 planner and TSRs.



(a) (b)



(c)

Figure 13: 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.

Figure 14: Snapshots from the execution of the box stacking task on the HRP3 robot.

## 7.5 Simultaneous Constraints and Goal Sampling

The task in this problem is to place a bottle held by HRP3 into a refrigerator (see Figure 13b). 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 (as in [Diankov et al., 2008]) 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. The second TSR Chain(2 element) is defined for the right arm and is described in Section 5.1. This chain is also marked for both sampling goals and constraining pose. The third TSR Chain(1 element) is defined for the left arm and constrains the robot to disallow titling of the bottle during the robot's motion. This chain is used only as a pose constraint. Its bounds are:

$$\mathbf{B}^w = \begin{bmatrix} -\infty & \infty \\ -\infty & \infty \\ -\infty & \infty \\ 0 & 0 \\ 0 & 0 \\ -\pi & \pi \end{bmatrix} \tag{20}$$

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.

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 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 ConstrainConfig function. Theoretically, a TSR Chain defined for goal sampling need only be a super-set of the goal configurations that meet all constraints (as long as it is of the same dimensionality). However, as the probability of sampling a goal from this TSR chain which meets all constraints decreases, the planner will require more time to generate a goal configuration, thus slowing down the algorithm.

Runtimes for 30 runs of this problem in simulation can be seen in Table 7.6.

## 7.6 Manipulating a Passive Chain

The task in this problem is for the robot to assist in placing a disabled person into bed (see Figure 13c). 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. In this problem, we get the goal configuration of the robot from inverse kinematics on the target pose of the person's hand. The robot's grasp of the person's hand is assumed to be rigid. No goal sampling is performed in this example.

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.

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.

This example shows that the TSR framework is capable of handling complex chains of constraints in addition to the simpler constraints of the previous problems. Runtimes for 30 runs of this example in simulation can be seen in Table 7.6.

| | Mean | Std. Dev | % Success |
|---|---|---|---|
| Closed Chain Kinematics | 4.21s | 2.00s | 100% |
| Simultaneous Constraints and Goal Sampling | 1.54s | 0.841s | 100% |
| Manipulating a Passive Chain | 1.03s | 0.696s | 100% |

TABLE 7.6: RUNTIMES FOR EXAMPLE PROBLEMS USING HRP3

# 8 Extension: Addressing Pose Uncertainty with Task Space Regions

In an effort to broaden the applicability of our framework to larger classes of real-world problems, we have been studying how to compute safe plans in the presence of uncertainty. A common assumption when planning for robotic manipulation tasks is that the robot has perfect knowledge of the geometry and pose of objects in the environment. For a robot operating in a home environment it may be reasonable to have geometric models of the objects the robot manipulates frequently and/or the robot's work area. However these objects and the robot often move around the environment, introducing uncertainty into the pose of the objects relative to the robot. Laser-scanners, cameras, and sonar sensors can all be used to help resolve the poses of objects in the environment, but these sensors are never perfect and usually localize the objects to be within some hypothetical set of pose estimates. Planning without regard to this set of estimates can violate task specifications.

Suppose that a robot arm is to pick up an object by placing its end-effector at a particular pose relative to the object and closing the fingers. If there is any uncertainty in the pose of the object, generally no guarantee can be made that the end-effector will reach a specific point relative to the true pose of object. Depending on the task, this lack of precision may range from being the source of minor disturbances to being the cause of critical failure.

TSRs allow planning for manipulation tasks in the presence of pose uncertainty by ensuring that the given task requirements are satisfied for all hypotheses of an object's pose. TSRs also provide a way to quickly reject tasks which cannot be guaranteed to be accomplished given the current pose uncertainty estimates. In this section, we show how to modify the TSRs of a given task to account for pose uncertainty and guarantee that samples drawn from the modified TSRs will meet task specifications. The methods presented in this section apply only to the TSR representation; we have not yet generalized them to account for TSR Chains because we do not yet have a method for intersecting the implicit sets of poses defined by TSR Chains.

## 8.1 Intersecting TSRs

Let the set of pose hypotheses for a given object be a set of transformation matrices $\mathcal{H}$. Also, let the set of TSRs defined for this object be $\mathcal{T}$. The process for generating a new set of TSRs $\mathcal{T}_{new}$ that takes into account $\mathcal{H}$ is shown in Algorithm 7. The goal of this process is to find the intersection of copies of each TSR corresponding to each pose hypothesis. Any point sampled from within the volume of intersection is guaranteed to meet the task specification despite pose uncertainty.

This algorithm first splits every TSR $t \in \mathcal{T}$ to take into account the rotation uncertainty in $\mathcal{H}$, generating a set $\mathcal{T}_{split}$ for each $t$. See Figure 15 for an illustration of this process. It then places a duplicate of each $t_s \in \mathcal{T}_{split}$ at every location defined by the transforms in $\mathcal{H}$. Next, it computes the volume of intersection of all duplicates for every $t_s$ (see Figure 16). Recall that TSR bounds define a cuboid in pose space. The volume of intersection between multiple cuboids is computed by first converting all faces of all cuboids into linear constraints via the FacesToLinInequalities function and then converting those linear constraints into vertices $P$ of a 6D polytope via the GetVerticesInequalities function. Since TSRs are convex we know that the polytope of intersection must be convex as well. If the uncertainty is too great (i.e. there is no 6D point where all duplicates intersect), $P$ will be empty. If $P$ is not empty, we place an axis-aligned bounding box around $P$, set this as the new bounds of $t_s$, and add $t_s$ to $\mathcal{T}_{new}$. Note that it is irrelevant which element of $\mathcal{H}$ is used as $\mathbf{T}^0_{h_0}$ because the results will always be the same in the world frame.

**Algorithm 7**: ApplyUncertainty($\mathcal{T}$,$\mathcal{H}$)

$\mathbf{T}^0_{h_0} \leftarrow$ Any element of $\mathcal{H}$;
$\mathcal{T}_{new} \leftarrow \emptyset$;
**for** $t \in \mathcal{T}$ **do**
  $\mathcal{T}_{split} \leftarrow$ SplitRotations($t$, $\mathbf{T}^0_{h_0}$, $\mathcal{H}$);
  **for** $t_s \in \mathcal{T}_{split}$ **do**
    $A \leftarrow \emptyset$; $b \leftarrow \emptyset$;
    **for** $\boldsymbol{T}^0_h \in \mathcal{H}$ **do**
      $V \leftarrow$ GetVertices($t_s$);
      $V_{xyz} \leftarrow (\mathbf{T}^0_{h_0})^{-1}\mathbf{T}^0_h V_{xyz}$;
      $F \leftarrow$ GetFaces($V$);
      $\{A_{temp}, b_{temp}\} \leftarrow$ FacesToLinInequalities($F$);
      $A \leftarrow A \cup A_{temp}$;
      $b \leftarrow b \cup b_{temp}$;
    **end**
    $P \leftarrow$ GetVerticesFromInequalities($A$, $b$);
    **if** $P = \emptyset$ **then**
      $t_s.\mathbf{T}^0_w \leftarrow h_0$;
      $t_s.\mathbf{B}^w \leftarrow$ BoundingBox($P$);
      $t_s.\mathbf{T}^w_e \leftarrow t.\mathbf{T}^w_e$;
      $t_s.\mathbf{LI} \leftarrow \{A, b\}$;
      $\mathcal{T}_{new} \leftarrow \mathcal{T}_{new} \cup t_s$;
    **end**
  **end**
**end**
**return** $\mathcal{T}_{new}$;

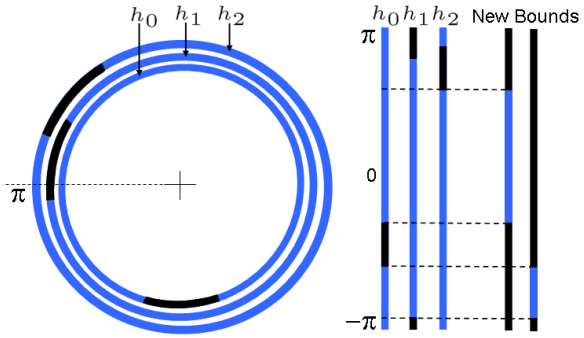



Figure 15: Process for splitting TSRs to take into account rotation uncertainty. Only one dimension of rotation is shown here. The three concentric circles correspond to a *single* TSR's bound in Roll that has been rotated by transforms $\mathbf{T}^0_{h_0}$, $\mathbf{T}^0_{h_1}$, and $\mathbf{T}^0_{h_2}$. Blue regions correspond to allowable rotations and black ones to unallowable rotations. The circles are cut at $\pi = -\pi$ and overlaid on the right. The strips where all rotations are valid (there are no black regions) are extracted as new separate bounds for this dimension. This process is identical for Roll, Pitch, and Yaw. The cartesian product of the new bounds for Roll, Pitch, and Yaw along with the original x, y, and z bounds produces a new set of TSRs $\mathcal{T}_{split}$.

## 8.2 Direct Sampling from the Volume of Intersection

In order to guarantee that a directly sampled 6D point meets the uncertainty specification of the problem, samples drawn from $t_s$ must lie inside the polytope defined by $P$. Ideally, we would like to generate uniformly random samples from within $P$ directly. Indeed, this is always possible because the polytope defined by $P$ is convex. Because the polytope is convex, it can always be divided into simplices using Delaunay Triangulation. To generate a uniformly random sample from a collection of simplices, we first select a simplex proportional to its area and then sample within that simplex by generating a random linear combination of its vertices [Devroye, 1986]. For simple polytopes, this method is quite efficient, however as the polytope defined by $P$ grows more complex, the Delaunay Triangulation becomes more costly, thus this method usually does not scale well with the number of hypotheses in $\mathcal{H}$.

Rejection sampling can also be used to sample from the polytope defined by $P$. When using rejection sampling, we sample a point $x$ uniformly at random from the bounding-box of $P$ until we find an $x$ which satisfies $b - Ax \geq 0$, where the matrix $A$ and the vector $b$ describe the hyperplanes and offsets, respectively, that define the faces of $P$. This method is quite fast in practice and does not require triangulating the polytope defined by $P$, thus it is more suitable for use in an online planning scenario.

In order to accommodate rejection sampling with TSRs, we add another element to our TSR definition (Section 4.1) for tasks with pose uncertainty:

- **LI**: Linear inequalities of the form $b - Ax \geq 0$

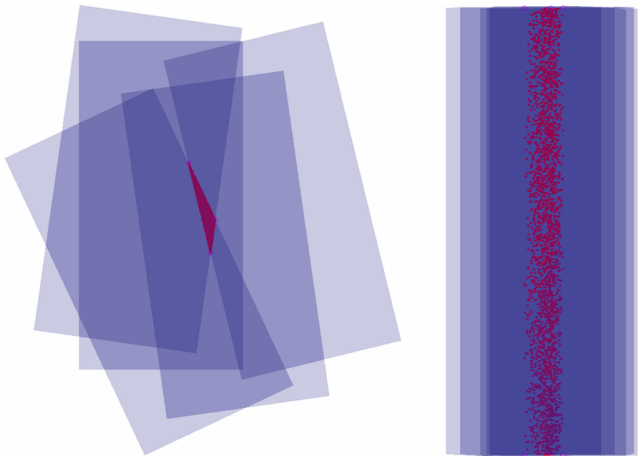


Figure 16: Intersection of five instances of a TSR. *Left*: x-y view. *Right*: y-z view. The red points are sampled within the polytope of intersection using rejection sampling.

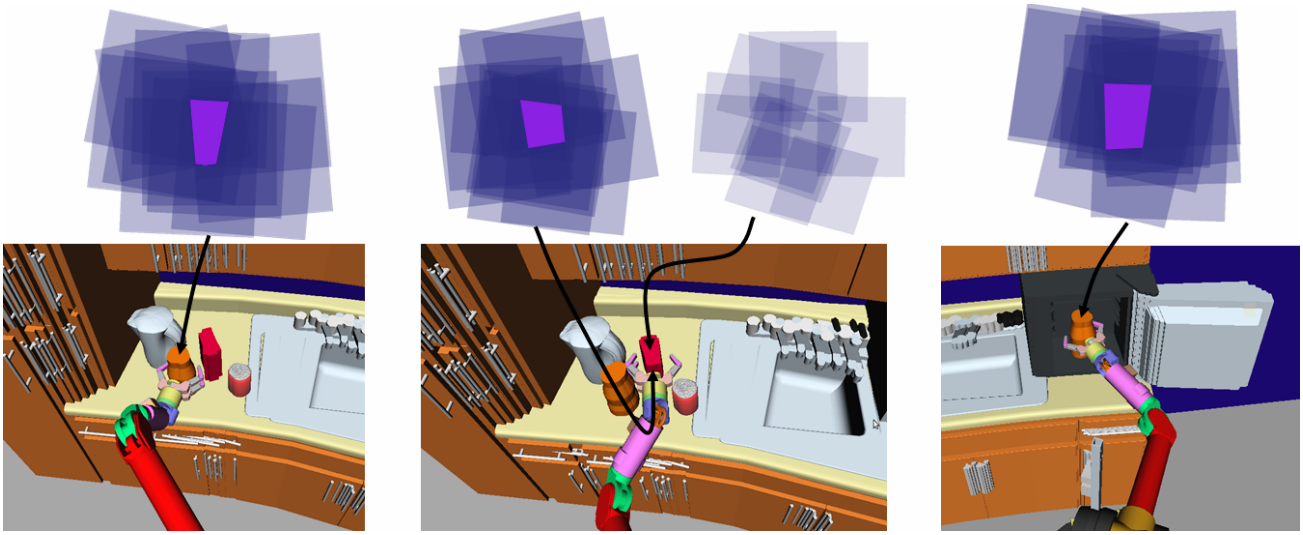


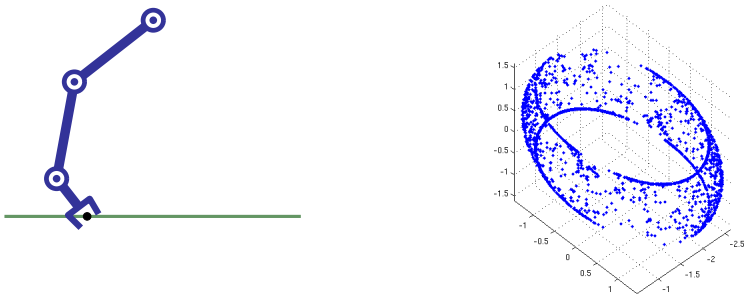
Figure 17: Sampled goal configurations of the WAM arm that are guaranteed to meet task specifications despite uncertainty for three reaching tasks. The intersecting boxes above show several of the intersecting TSRs for these tasks. In the task shown in the center the TSR for grasping the box from the top is eliminated by uncertainty (there is no point where all the boxes intersect) while the one for grasping it from the side is not.

If the $\mathcal{T}_{new}$ returned by ApplyUncertainty($\mathcal{T}$, $\mathcal{H}$) is empty, then we know that it is impossible to accomplish this task with the uncertainty in $\mathcal{H}$. We can thus reject this task without calling the planner, a key advantage of this approach.

Several example reaching tasks and associated TSRs are shown in Figure 17. The scene contains duplicates of every object at its pose estimates to ensure that the path generated by the planner is collision-free for all hypotheses of object pose.

# 9 Discussion

The main advantage of our framework is the generality in the constraint representation and planning method. We are able to tackle a wide range of problems without resorting to highly specialized techniques. This is evidenced in part by the



(a) (b)

Figure 18: The depiction of a TSR and the samples on the corresponding manifold generated using the CBiRRT2. (a) The end-effector must be on the line with an orientation within $\pm 0.7$rad of downward. (b) The sampling is biased toward the boundaries of the manifold.

low number of parameters in our algorithm and that, despite a wide range of problems and robots, the values of these parameters were constant across all the example problems.

Another advantage of our approach is that short-cutting paths produced by our planner uses the same process as growing branches. This allows us to easily shorten our paths in the presence of multiple constraints, which was left as an open problem in previous work [Stilman, 2007, Yakey et al., 2001].

Since our framework is built around a sampling-based planner we inherit many of the difficulties of this approach to planning. For instance, it would be difficult to incorporate non-holonomic constraints and dynamics into our framework because these constraints would disrupt the distance metric used by the RRT. On the other hand, we can employ a wide range of techniques that speed up sampling-based planning and repair paths in the presence of moving obstacles. These techniques can be incorporated into our planner to make it faster and more robust.

One criticism of the TSR framework 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 expressivity. 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 representation is surely possible, we have yet to find one that is as straightforward to specify and as convenient for sampling-based planning.

Even if we were to adopt a different constraint representation to solve a specific problem, many parts of our framework would still be applicable. For instance, we would still be able to use CBiRRT2 as long as the representation provided for fast sampling and distance-checking methods. The proof of probabilistic completeness (Appendix A) would also apply, because it addresses pose constraints in general, not only TSRs.

One drawback of the TSR framework is that we have not yet devised a method to prioritize constraints. As a result, we can only specify simultaneous constraints using an OR structure if we use the projection strategy. This means that the planner has the option of satisfying constraint 1 OR constraint 2 OR constraint 3, and so on. One way to address this issue would be to attempt to satisfy constraints in order of their priority, though this is not equivalent to a truly prioritized framework. Another approach would be to use projection operators that allow prioritization, like those discussed in Section A.7.2, however it is unclear how to retain probabilistic completeness while using those operators. We are very interested in investigating how to incorporate prioritization into the framework in the future.

Finally, a practical issue with our method of planning on constraint manifolds is that the distribution of samples may sometimes be undesirable, i.e. the projection strategy biases samples toward the boundaries of the manifold (Figure 18). 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 algorithm to perform slowly if an interior point of the manifold is needed to complete a path. On the other hand, the algorithm is much faster at finding configurations on the boundary, which can be useful for problems such as the weight-lifting example, where the robot must slide the dumbbell to the edge of the table in order to lift it.

# 10 Conclusion

We have presented a practical framework for representing and planning with end-effector pose constraints, among others. The underpinnings of our approach are the TSR and TSR Chain representations, which allow us to easily specify constraints for a given problem. These representations are designed specifically for sampling-based planners and as a result possess fast sampling and distance-checking methods. The TSR framework uses these representations to characterize and solve many real-world problems ranging from reaching to grasp to manipulating articulated objects. Most importantly, the framework is designed for real-world robots, such as the mobile manipulator and humanoid shown earlier. We discuss the completeness properties of planning with end-effector pose constraints and present a proof for a class of planning algorithms in Appendix A. This provides a theoretical foundation for our framework and reinforces its generality. Finally, we have begun to apply our framework to planning with uncertainty, where we developed a method that exploits freedoms in task specification to compensate for uncertainty in object pose.

# 11 Acknowledgements

This research was partially supported by Intel Labs Pittsburgh and by the National Science Foundation under Grant No. EEC-0540865. Thanks to Ross Knepper, Julius Ziegler, Joel Chestnut, Nico Blodow, and David Handron for helpful discussions.

# A Appendix: Probabilistic Completeness

We now discuss a proof of probabilistic completeness for our approach to planning with end-effector pose constraints. Since the proof is quite lengthy, we provide only a summary here (for details see [Berenson and Srinivasa, 2010]). We emphasize that this proof is valid for a class of algorithms that plan with constraints on end-effector pose, not only CBiRRT2. We also note that, while we focus on the TSR representation in the rest of this paper, this section makes no assumptions about the representation of pose constraints. Our only restriction is that the dimensionality of the manifold described by the constraint is fixed for a given problem.

Depending on the definition of an end-effector pose constraint, it can induce a variety of manifolds in the robot's configuration space. If these manifolds have non-zero volume in the C-space (see Figure 19d) it is straightforward to show that an RRT-based algorithm is probabilistically complete because rejection sampling in the C-space will eventually place samples inside of the manifold. However, if a pose constraint induces a lower-dimensional manifold, i.e. one that has zero volume in the C-space (see Figures 19e and 19f), rejection sampling in the C-space will not generate a sample on the constraint manifold.

RRT-based algorithms can overcome this problem by using the projection strategy: sampling coupled with a projection operator to move configuration space samples onto the constraint manifold. However, it is not clear whether the projection strategy produces adequate coverage of the constraint manifold to guarantee probabilistic completeness. The proof presented in this section guarantees probabilistic completeness for a class of RRT-based algorithms given an appropriate projection operator. This proof is valid for constraint manifolds of any fixed dimensionality.

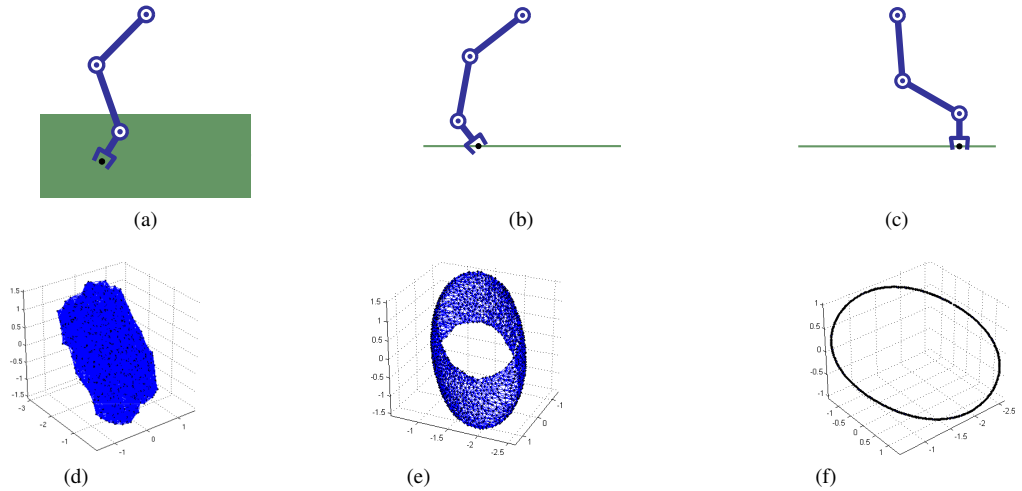



Figure 19: Three pose constraints and their corresponding C-space constraint manifolds for a 3-link manipulator. (a) The end-effector must be in the green rectangle with an orientation $\pm 0.7$rad of downward. (b) The end-effector must be on the line with an orientation $\pm 0.7$rad of downward. (c) The end-effector must be on the line pointing downward. (d-f) show graphs created from sampling on the manifolds corresponding to the constraints in (a-c), respectively. Black points are nodes and blue lines are edges. The manifold in (d) has non-zero volume in the C-space, the manifolds in (e) and (f) do not.

| Name | Symbol | Dimension |
|---|---|---|
| C-space | $\mathcal{Q}$ | $n$ |
| Configuration | $q$ | 0 |
| Constraint Manifold | $\mathcal{M}$ | $m = n - (r - d)$ |
| Reachability | $\mathcal{R}$ | $r$ |
| Pose | $x(q)$ | 0 |
| Task Constraint | $\mathcal{T}$ | $d \leq r$ |

TABLE A.1: DEFINITIONS USED THROUGHOUT THE PROOF OF PROBABILISTIC COMPLETENESS.

Our proof of probabilistic completeness has two parts: first, we present a set of properties for the projection operator and prove that these properties allow the projection strategy to cover the constraint manifold. Second, we describe a class of RRT-based algorithms (such as CBiRRT2 and TCRRT [Stilman, 2007]), which use such a projection operator and prove that they are probabilistically complete.

## A.1 Definitions

A topological *manifold* is a second-countable Hausdorff space where every point has a neighborhood homeomorphic to an open Euclidean $n$-ball, where $n$ is allowed to vary. Manifolds can be disjoint, with each piece of the manifold called a *connected component*. Manifolds that have a fixed $n$ (i.e. $n$-dimensional manifolds) are called *pure* manifolds.

Let $\mu_n$ be a measure of volume in an $n$-dimensional space. If the volume of a manifold in an embedding space is zero, then the probability of generating a sample on the manifold by rejection sampling in the embedding space is zero. Conversely if the volume of a manifold in an embedding space is *not* zero, then the probability of generating a sample on the manifold by rejection sampling in the embedding space will go to 1 as the number of samples goes to infinity.

A sampling method *covers* a manifold if it generates a set of samples such that any open $n$-dimensional ball contained in the manifold contains at least one sample.

Our proof of manifold coverage hinges on the concept of *self-motion manifolds*, which were described in [Burdick, 1989]. A self-motion manifold is the set of configurations in C-space which place the end-effector of the robot in a certain pose.

Let the reachable manifold of end-effector poses of the given manipulator be $\mathcal{R} \subseteq SE(3)$. $\mathcal{R}$ is defined by the Forward Kinematics function of the robot:

$$x : \mathcal{Q} \to \mathcal{R} \tag{21}$$

where $\mathcal{Q}$ is the C-space. $x$ is always surjective and can be one-to-one or many-to-one, depending on the manipulator. We restrict our proof to manipulators whose $\mathcal{Q}$ and $\mathcal{R}$ are both pure manifolds.

Let $\mathcal{Q}$ be $n$-dimensional, where $n$ is the number of DOF of the manipulator. We will parameterize $SE(3)$ locally using three variables for translation and three for rotation, i.e. a pose will be a vector in $\mathbb{R}^6$. Let $\mathcal{R}$ be $r$-dimensional, where $r \leq 6$.

Let the manifold of end-effector poses allowable by the task be $\mathcal{T} \subseteq \mathcal{R}$. Let this manifold have dimensionality $d \leq r$. We will assume that $d$ is fixed, though we will discuss the implications of allowing $d$ to vary in Section A.7. Let the manifold of configurations that place the robot's end-effector in $\mathcal{T}$ be $\mathcal{M} \subseteq \mathcal{Q}$. $\mathcal{M}$ is the union of all self-motion manifolds that map to a pose in $\mathcal{T}$:

$$\mathcal{M} = \bigcup_{t \in \mathcal{T}} \{q \in \mathcal{Q} \mid x(q) = t\} \tag{22}$$

$\mathcal{M}$ has dimensionality $m = n - (r - d)$.

We will be using the (weighted-)Euclidian distance metric on $SE(3)$, which we will denote as $\mathrm{dist}$. This distance metric has the property that each pose in $\mathcal{T}$ has at least an $(r - d)$-dimensional *Voronoi cell* in $\mathcal{R}$. A Voronoi cell is the set of points that are closer to a certain point than to any other given a distance metric [LaValle, 2006].

Table A.1 summarizes the definitions and dimensionalities of manifolds used in the proof.

## A.2 Proof of manifold coverage by the projection strategy

The projection strategy first produces a sample in the C-space and then projects that sample onto $\mathcal{M}$ using a *projection operator* $P : \mathcal{Q} \to \mathcal{M}$. In order to show that an algorithm using the projection strategy is probabilistically complete, we must first show that this method covers $\mathcal{M}$.

This section will show that the projection strategy does indeed cover $\mathcal{M}$ if the projection operator has the following properties:

1. $P(q) = q$ if and only if $x(q) \in \mathcal{T}$
2. If $x(q_1)$ is closer to $x(q_2) \in \mathcal{T}$ than to any other point in $\mathcal{T}$ and $\mathrm{dist}(x(q_1), x(q_2)) < \epsilon$ for an infinitesimal $\epsilon > 0$, then $x(P(q_1)) = x(q_2)$.

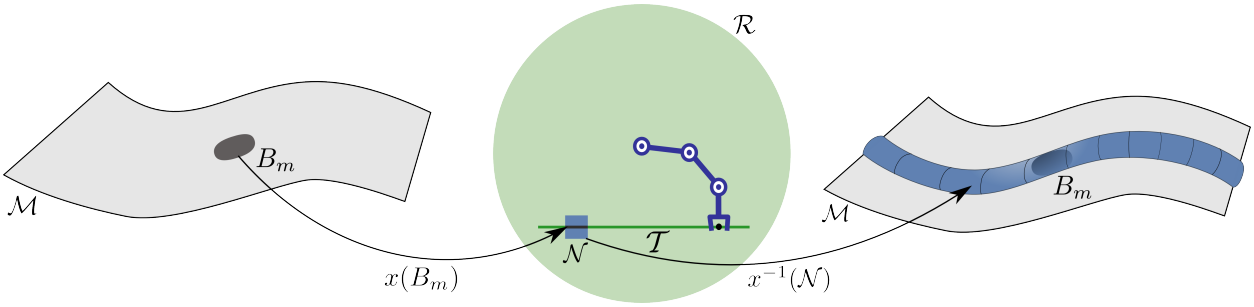



Figure 20: An example showing the process used to define $x^{-1}(\mathcal{N})$. $d = 1$, $r = 2$, and $n = 3$.

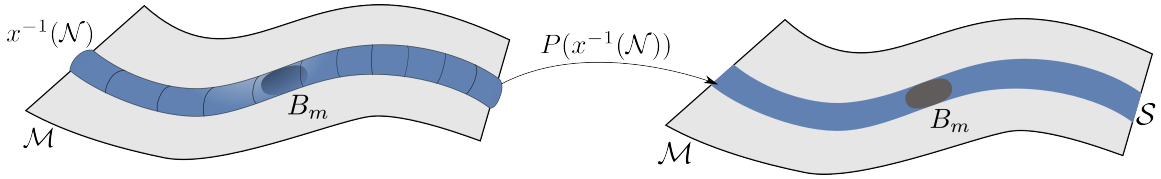



Figure 21: An example of $P(x^{-1}(\mathcal{N})) = \mathcal{S}$. $d = 1$, $r = 2$, and $n = 3$.

The first property guarantees that a configuration that is already in $\mathcal{M}$ will project to itself. The second property ensures that any pose in $\mathcal{T}$ can be chosen for projection. We will describe the underlying mechanics of the projection operator in Section A.4.

If $d = r$ then $\mu_n(\mathcal{M}) > 0$. By the first property of $P$, a sample placed in $\mathcal{M}$ will project to itself. Thus the projection strategy will cover $\mathcal{M}$ by the same principle as rejection sampling when $\mathcal{M}$ is of the same dimension as the C-space.

The remainder of this section will focus on proving coverage when $d < r$, i.e. when $\mu_n(\mathcal{M}) = 0$. Consider an open $m$-dimensional ball $B_m(q) \subseteq \mathcal{M}$ for any $q \in \mathcal{M}$. Note that *open* in this context refers to the openness of the set with respect to $\mathcal{M}$. For notational simplicity, let $B_m$ represent any $B_m(q)$ for any such $q$. We will show that the projection strategy places a sample in any $B_m$ as the number of iterations goes to infinity, thus covering $\mathcal{M}$.

Consider an $n$-dimensional manifold $\mathcal{C}(B_m) = \{q : P(q) \in B_m,\ q \in \mathcal{Q}\}$. If such a $\mathcal{C}$ exists for any $B_m$, the projection strategy will place a sample inside $\mathcal{C}$ with probability greater than 0 (because $\mathcal{C}$ is $n$-dimensional) and that sample will project into $B_m$. This will guarantee coverage of $\mathcal{M}$ as the number of iterations goes to infinity. But how do we guarantee that such a $\mathcal{C}$ exists for any $B_m$?

We will show that $\mathcal{C}$ can be defined as the intersection of two $n$-dimensional manifolds, $x^{-1}(\mathcal{N})$ and $\mathcal{UH}$, both of which must exist for any $B_m$ (notation will be explained in subsequent subsections). The following subsections describe each of these manifolds and show that $(x^{-1}(\mathcal{N}) \cap \mathcal{UH})$ must be $n$-dimensional and all configurations in $(x^{-1}(\mathcal{N}) \cap \mathcal{UH})$ must project into $B_m$. Thus $(x^{-1}(\mathcal{N}) \cap \mathcal{UH})$ meets the requirements of $\mathcal{C}$, which completes the proof of coverage.

## A.3 To Task Space and Back Again: Defining $x^{-1}(\mathcal{N})$

In this subsection we will define a manifold $x^{-1}(\mathcal{N})$, which projects into a set of self-motion manifolds $\mathcal{S} \subseteq \mathcal{M}$ that intersects $B_m$. We do this by mapping $B_m$ into task space, constructing an $r$-dimensional manifold of poses $\mathcal{N}$ that project into $x(B_m)$ and then mapping $\mathcal{N}$ back into C-space. We summarize these three steps in Figure 20. $\mathcal{N}$ is guaranteed to exist by the second property of the projection operator but the construction of this manifold is somewhat detailed, so we refer the reader to [Berenson and Srinivasa, 2010] for a full explanation.

The manifold produced at the end of the three-step process is $x^{-1}(\mathcal{N})$, which has the following properties:

1. $x^{-1}(\mathcal{N})$ is $n$-dimensional.
2. $x^{-1}(\mathcal{N})$ contains $B_m$
3. $P(x^{-1}(\mathcal{N})) = \mathcal{S}$

$\mathcal{S}$ is defined as the manifold of configurations which map into $x(B_m)$; i.e. $\mathcal{S} = \{q \in \mathcal{M} \mid x(q) \in x(B_m)\}$ (see Figure 21). We will use $\mathcal{S}$ to show coverage of $\mathcal{M}$ in Section A.5.

$\mathcal{S}$ has the following properties:

1. If a self-motion manifold intersects $B_m$, it is a subset of $\mathcal{S}$.
2. If a self-motion manifold does not intersect $B_m$, it is not a subset of $\mathcal{S}$.
3. $\mathcal{S} \neq \emptyset$.

## A.4 Projection into a ball on a self-motion manifold

We have shown that $P(x^{-1}(\mathcal{N})) = \mathcal{S}$ and described some properties of $\mathcal{S}$, however we have not stated where on $\mathcal{S}$ a projected configuration will go. It is possible that a $q \in x^{-1}(\mathcal{N})$ will project to a configuration outside of $B_m$ because $\mathcal{S}$ may contain configurations outside of $B_m$ (see Figure 21).

In order to show that the probability of placing a sample inside $B_m$ using the projection strategy is greater than 0, we need to show that there exists an $n$-dimensional manifold around a ball on a self-motion manifold that projects into that ball. The purpose of this subsection is to prove this property of self-motion manifolds. The remainder of this subsection will consider a self-motion manifold in isolation in order to show this property.

Let us now look closer at the mechanism of projection used by $P$. In this paper, we focus on projection operators based on Jacobian pseudo-inverse or Jacobian transpose. These kinds of projection operators step towards a pose target and this process can be written as a differential equation:

$$\frac{dq}{dt} = f(q(t)), t \in [0, \infty) \tag{23}$$

An *equilibrium point* $\bar{q}$ of Equation 23 satisfies $f(\bar{q}) = 0$. $\bar{q}$ is *asymptotically stable* if any $q(0)$ inside a neighborhood containing $\bar{q}$ will converge to $\bar{q}$ as the number of iterations of Equation 23 goes to infinity.

The set of equilibrium points for the task of placing the end-effector at a given pose is the self-motion manifold $\bar{\mathcal{Q}}$, which is $(n-r)$-dimensional. In [Berenson and Srinivasa, 2010], we showed that, for any $\bar{q} \in \bar{\mathcal{Q}}$, there exists an $r$-dimensional ball $B_r(\bar{q})$ within which the conditions for asymptotic stability hold. Thus the evolution of Equation 23 results in the projection $P(q(0)) = \bar{q}$ for all $\bar{q} \in \bar{\mathcal{Q}}$, for all $q(0) \in B_r(\bar{q})$. Note that this fact is only valid for a self-motion manifold in isolation.

Let us now consider an open $(n-r)$-dimensional ball $B_{\bar{\mathcal{Q}}}(\bar{q}) \subseteq \bar{\mathcal{Q}}$. Define $\mathcal{H}(B_{\bar{\mathcal{Q}}}(\bar{q}))$ as the manifold $B_r(\bar{q}) \times B_{\bar{\mathcal{Q}}}(\bar{q})$ embedded in $\mathcal{Q}$ (see Figure 22).

$\mathcal{H}$ has the following properties:

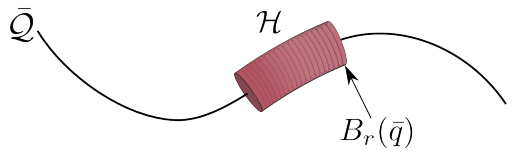



Figure 22: A self-motion manifold in C-space. $r = 2$ and $n = 3$.

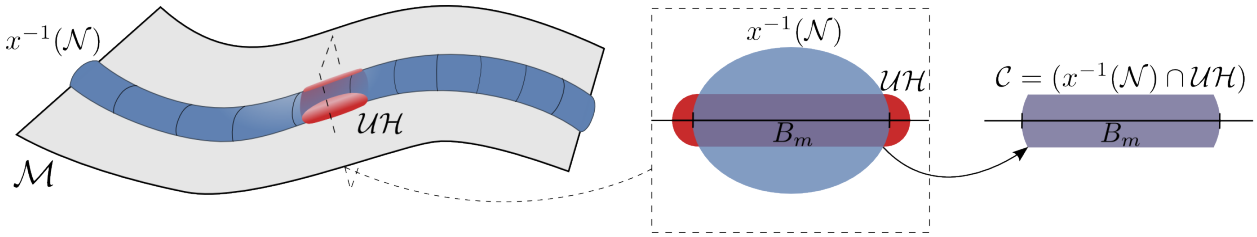



Figure 23: An example of $\mathcal{C} = (x^{-1}(\mathcal{N}) \cap \mathcal{UH})$. $d = 1$, $r = 2$, and $n = 3$.

1. $\mathcal{H}$ is $n$-dimensional.
2. $\mathcal{H}$ contains $B_{\bar{\mathcal{Q}}}(\bar{q})$.
3. $P(q) \in B_{\bar{\mathcal{Q}}}(\bar{q})$ for all $q \in \mathcal{H}$ (conditional).

It is important to note that we have only described $\mathcal{H}$ for a self-motion manifold in isolation. The third property of $\mathcal{H}$ is only valid when $x(q)$ is closer to $x(\bar{\mathcal{Q}})$ than to any $x(k)$, for a self-motion manifold $k \subseteq (\mathcal{M} - \bar{\mathcal{Q}})$. I.e. the third property holds only when $\bar{\mathcal{Q}}$ is "chosen" by the projection operator.

## A.5 Putting it all together

We will now bring the concepts developed thus far together to show coverage of $\mathcal{M}$ by the projection strategy. Recall that, to show coverage of $\mathcal{M}$, we must show that the projection strategy places a sample inside any $B_m \subseteq \mathcal{M}$.

Section A.3 showed that a configuration inside $x^{-1}(\mathcal{N})$ will project into $\mathcal{S}$, which consists of all self-motion manifolds that intersect $B_m$. Let us construct an $n$-dimensional manifold $\mathcal{UH}(B_m)$ by taking the union of the $n$-dimensional $\mathcal{H}$ manifolds around every $\bar{\mathcal{Q}} \cap B_m$ for all $\bar{\mathcal{Q}} \subseteq \mathcal{S}$:

$$\mathcal{UH}(B_m) = \bigcup_{\bar{\mathcal{Q}} \subseteq \mathcal{S}} \mathcal{H}(\bar{\mathcal{Q}} \cap B_m) \tag{24}$$

$\mathcal{UH}$ inherits the properties of $\mathcal{H}$, as well as the condition on the third property: A configuration $q \in \mathcal{UH}$ will project to a configuration inside $B_m$ by the third property of $\mathcal{H}$ if, for some $\bar{\mathcal{Q}} \subseteq \mathcal{S}$, $x(q)$ is closer to $x(\bar{\mathcal{Q}})$ than to any $x(k)$, for a self-motion manifold $k \subseteq (\mathcal{M} - \mathcal{S})$.

Let $\mathcal{C} = (x^{-1}(\mathcal{N}) \cap \mathcal{UH})$ (see Figure 23). $\mathcal{C}$ has the following properties:

1. $C$ is $n$-dimensional.
2. $P(q) \in B_m$ for all $q \in \mathcal{C}$.

**Theorem 1** *The projection strategy places a sample inside any $B_m$ as the number of samples goes to infinity.*

**Proof:** By the first property of $\mathcal{C}$, $\mu_n(\mathcal{C}) > 0$. Thus the probability of sampling a $q \in \mathcal{C}$ is greater than 0 and, as the number of samples goes to infinity, the probability of sampling a $q \in \mathcal{C}$ goes to 1. $P(q) \in B_m$ by the second property of $\mathcal{C}$. Thus we have shown that the projection strategy places a sample inside any $B_m$ as the number of samples goes to infinity, which entails that the projection strategy covers $\mathcal{M}$.

## A.6 Probabilistic completeness of RRT-Based algorithms using the projection strategy

We can use the fact that the projection strategy covers $\mathcal{M}$ to prove that RRT-based methods that plan paths on $\mathcal{M}$ are probabilistically complete. We focus on a class of RRT-based algorithms that plan with end-effector pose constraints, for example CBiRRT2 and TCRRT [Stilman, 2007]. These algorithms grow trees on $\mathcal{M}$ by sampling near an existing node on $\mathcal{M}$ and then projecting that sample to $\mathcal{M}$. An RRT-based algorithm with the following properties is probabilistically complete:

1. Given a node of the existing tree, the probability of sampling in an $n$-dimensional ball centered at that node is greater than $0$ and the sampling covers this ball.
2. The algorithm uses a projection operator with the properties of $P$ to project samples to $\mathcal{M}$.

We know that the algorithm will cover any ball in $\mathcal{M}$ centered at an existing configuration in the tree by Theorem 1. We can then use the *series-of-balls* argument to show that we place a node in any $B_m$ which is connected to the starting configuration. The basis of this argument is that we can construct a series of overlapping balls on the manifold which contains some node in the tree and overlaps with $B_m$. We can use Theorem 1 to show that an RRT node will be placed in each region of overlap between subsequent balls until the tree reaches $B_m$. For a more detailed explanation of the series-of-balls argument see [Svestka, 1996] and [Kuffner and LaValle, 2000].

Thus we have shown that the RRT can reach any $B_m$ connected to the starting configuration, which entails that the algorithm is probabilistically complete.

## A.7 Implications of the Proof

We now discuss the implications of our proof for sampling goals, choosing projection operators, and planning with mixed-dimensional constraint manifolds.

### A.7.1 Probabilistically Complete Goal Sampling

The above proof also applies to pose goal constraints, where we would like to ensure that projection sampling covers the manifold of configurations corresponding to some set of pose goals in task space. The same principles apply as in the above proof, except that we do not rely on a planner to explore the manifold, we simply sample it using the projection strategy. It follows clearly from Theorem 1 that this goal manifold will be covered if we sample it using the projection strategy.

### A.7.2 Projection Operators

Section A.2 describes a projection operator that guarantees probabilistic completeness for RRT-based algorithms. The regularized Jacobian pseudo-inverse or Jacobian transpose IK methods can be used to perform the projection because they possess the requisite properties. Unfortunately, there are some common iterative methods which will *not* yield probabilistic completeness.

The null-space projection method [Sentis and Khatib, 2005] operates by using the null-space of the primary task (placing the end-effector in some pose) to satisfy secondary tasks such as collision-avoidance or balancing. For this method, the $\frac{dq}{dt}$ of Equation 23 is:

$$\frac{dq}{dt} = \mathbf{J}^{+}\dot{x} + (\mathbf{I} - \mathbf{J}^{+}\mathbf{J})\dot{q}_{\text{null}} \tag{25}$$

where $\mathbf{J}^{+}$ is the generalized pseudo-inverse of the Jacobian, $\dot{x}$ is the error in pose, and $\dot{q}_{\text{null}}$ is the error in meeting a secondary objective. This type of projection attains optimal configurations by sliding along a self-motion manifold when $\dot{x} = 0$. The secondary task induces local minima on the self-motion manifold which attract configurations from the rest of the self-motion manifold. Thus, if a ball on the self-motion manifold does not contain a local minimum of the secondary task, configurations projecting to that ball may escape by sliding along the manifold. It follows that this projection operator will not cover $\mathcal{M}$. Though using this projection operator in an RRT-based algorithm may yield an effective planner, it will not be probabilistically complete.

### A.7.3 Mixed-dimensional Constraint Manifolds

The proof of coverage and probabilistic completeness assumed that the pose constraint $\mathcal{T}$ had a fixed dimensionality $d$. If we allow $d$ to vary, then $m$ (the dimensionality of $\mathcal{M}$) will vary as well. Since our proof of coverage used only local properties of $\mathcal{M}$, we can extend this proof to the case of varying $m$ simply by applying the proof to each $m$-dimensional component of $\mathcal{M}$ for every $m$. However, there is the case when a ball around a point on $\mathcal{M}$ contains components of varying dimension. In this case, the ball can be split according to the dimensionality of its components and $\mathcal{C}$ can be shown to exist for one of these components, thus guaranteeing a sample will be placed in this ball.

Though we can show that the projection strategy covers $\mathcal{M}$, the proof of probabilistic completeness for RRT-based algorithms only holds when $\mathcal{M}$ is pure. The reason is that mixed-dimensional manifolds can be constructed such that all paths between two configurations must go through a narrow passage, which is of lower dimension than any component of the manifold. For instance, suppose $\mathcal{M}$ were composed of two lines that intersect at some configuration $q_p$. To get from one line to the other, the algorithm would need to find a path which contained $q_p$. Yet there is 0 probability of generating $q_p$ exactly, thus a path may never be found, though one exists. This difficulty is not caused by the projection strategy and is not limited only to pose constraints. Rather this difficulty arises for all RRT-based planners when they must find a path through a lower-dimensional narrow passage. Coverage of $\mathcal{M}$ does not entail probabilistic completeness in this case. In order to guarantee probabilistic completeness, the algorithm must be able to generate samples in these lower-dimensional narrow passages, which can be done in a problem-specific way (as in [Hauser and Latombe, 2008]).

# References


[Atkeson and Schaal, 1997] Atkeson, C. G. and Schaal, S. (1997). Learning tasks from a single demonstration. In *Proc. IEEE International Conference on Robotics and Automation (ICRA)*, pages 1706–1712.

[Bellman, 1957] Bellman, R. (1957). *Dynamic Programming*. Princeton University Press, Princeton, NJ.

[Bentivegna et al., 2004] Bentivegna, D., Atkeson, C. G., and Cheng, G. (2004). Learning tasks from observation and practice. *Robotics and Autonomous Systems*, 47:163–169.

[Berenson et al., 2009a] Berenson, D., Chestnutt, J., Srinivasa, S. S., Kuffner, J. J., and Kagami, S. (2009a). Pose-Constrained Whole-Body Planning using Task Space Region Chains. In *Proc. IEEE-RAS International Conference on Humanoid Robots (Humanoids09)*.

[Berenson and Srinivasa, 2010] Berenson, D. and Srinivasa, S. (2010). Probabilistically complete planning with end-effector pose constraints. In *IEEE International Conference on Robotics and Automation (ICRA)*.

[Berenson et al., 2009b] Berenson, D., Srinivasa, S., and Kuffner, J. (2009b). Addressing Pose Uncertainty in Manipulation Planning Using Task Space Regions. In *Proc. IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*.

[Berenson et al., 2009c] Berenson, D., Srinivasa, S. S., Ferguson, D., Collet, A., and Kuffner, J. (2009c). Manipulation planning with workspace goal regions. In *Proc. IEEE International Conference on Robotics and Automation (ICRA)*.

[Berenson et al., 2009d] Berenson, D., Srinivasa, S. S., Ferguson, D., and Kuffner, J. (2009d). Manipulation planning on constraint manifolds. In *Proc. IEEE International Conference on Robotics and Automation (ICRA)*.

[Bertram et al., 2006] Bertram, D., Kuffner, J., Dillmann, R., and Asfour, T. (2006). An integrated approach to inverse kinematics and path planning for redundant manipulators. In *Proc. IEEE International Conference on Robotics and Automation (ICRA)*, pages 1874–1879.

[Burdick, 1989] Burdick, J. (1989). On the inverse kinematics of redundant manipulators: characterization of the self-motion manifolds. In *Proc. IEEE International Conference on Robotics and Automation (ICRA)*, pages 264–270.

[Collet et al., 2009] Collet, A., Berenson, D., Srinivasa, S. S., and Ferguson, D. (2009). Object recognition and full pose registration from a single image for robotic manipulation. In *Proc. IEEE International Conference on Robotics and Automation (ICRA)*.

[Cortes and Simeon, 2004] Cortes, J. and Simeon, T. (2004). Sampling-based motion planning under kinematic loop-closure constraints. In *Workshop on the Algorithmic Foundations of Robotics (WAFR)*.

[Devroye, 1986] Devroye, L. (1986). *Non-Uniform Random Variate Generation*, pages 567–571. Springer-Verlag, New York.

[Diankov et al., 2008] Diankov, R., Srinivasa, S. S., Ferguson, D., and Kuffner, J. (2008). Manipulation planning with caging grasps. In *Proc. IEEE-RAS International Conference on Humanoid Robots (Humanoids 08)*.

[Drumwright and Ng-Thow-Hing, 2006] Drumwright, E. and Ng-Thow-Hing, V. (2006). Toward interactive reaching in static environments for humanoid robots. In *Proc. IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*.

[Hauser and Latombe, 2008] Hauser, K. and Latombe, J. (2008). Multi-Modal Motion Planning in Non-Expansive Spaces. In *Workshop on the Algorithmic Foundations of Robotics (WAFR)*.

[Hirano et al., 2005] Hirano, Y., Kitahama, K., and Yoshizawa, S. (2005). Image-based object recognition and dexterous hand/arm motion planning using rrts for grasping in cluttered scene. In *Proc. IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*.

[Howard et al., 2008] Howard, M., Klanke, S., Gienger, M., Goerick, C., and Vijayakumar, S. (2008). Learning potential-based policies from constrained motion. In *Proc. 8th IEEE-RAS International Conference on Humanoid Robots (Humanoids 08)*.

[Kavraki et al., 1996] Kavraki, L. E., Svestka, P., Latombe, J. C., and Overmars, M. H. (1996). Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE Transactions on Robotics and Automation*, 12(4):566–580.

[Khatib, 1987] Khatib, O. (1987). A unified approach for motion and force control of robot manipulators: The operational space formulation. *IEEE Transactions on Robotics and Automation*, 3(1):43–53.

[Koga et al., 1994] Koga, Y., Kondo, K., Kuffner, J., and claude Latombe, J. (1994). Planning motions with intentions. In *SIGGRAPH*.

[Kuffner and LaValle, 2000] Kuffner, J. J., J. and LaValle, S. M. (2000). RRT-connect: An efficient approach to single-query path planning. In *Proc. IEEE International Conference on Robotics and Automation (ICRA)*.

[LaValle and Kuffner, 2000] LaValle, S. and Kuffner, J. (2000). Rapidly-exploring random trees: Progress and prospects. In *Workshop on the Algorithmic Foundations of Robotics (WAFR)*.

[LaValle, 2006] LaValle, S. M. (2006). *Planning Algorithms*. Cambridge University Press, Cambridge, U.K.

[Oriolo and Mongillo, 2005] Oriolo, G. and Mongillo, C. (2005). Motion planning for mobile manipulators along given end-effector paths. In *Proc. IEEE International Conference on Robotics and Automation (ICRA)*.

[Oriolo et al., 2002] Oriolo, G., Ottavi, M., and Vendittelli, M. (2002). Probabilistic motion planning for redundant robots along given end-effector paths. In *Proc. IEEE/RSJ International Conference on Intelligent Robots and System (IROS)*.

[Sciavicco and Siciliano, 2000] Sciavicco, L. and Siciliano, B. (2000). *Modeling and Control of Robot Manipulators*, pages 96–100. Springer, 2nd edition.

[Seereeram and Wen, 1995] Seereeram, S. and Wen, J. (1995). A global approach to path planning for redundant manipulators. *IEEE Transactions on Robotics and Automation*, 11(1):152–160.

[Sentis and Khatib, 2005] Sentis, S. and Khatib, O. (2005). Synthesis of whole-body behaviors through hierarchical control of behavioral primitives. *International Journal of Humanoid Robotics*, 2:505–518.

[Stilman, 2007] Stilman, M. (2007). Task constrained motion planning in robot joint space. In *Proc. IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*.

[Stilman et al., 2007] Stilman, M., Schamburek, J. U., Kuffner, J., and Asfour, T. (2007). Manipulation planning among movable obstacles. In *Proc. IEEE International Conference on Robotics and Automation (ICRA)*.

[Sugihara and Nakamura, 2002] Sugihara, T. and Nakamura, Y. (2002). Whole-body cooperative balancing of humanoid robot using cog jacobian. In *Proc. IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*.

[Svestka, 1996] Svestka, P. (1996). On probabilistic completeness and expected complexity of probabilistic path planning. Technical report, UU-CS-96-20. Department of Computer Science, Utrecht University, Utrecht, Netherlands.

[Vande Weghe et al., 2007] Vande Weghe, M., Ferguson, D., and Srinivasa, S. S. (2007). Randomized path planning for redundant manipulators without inverse kinematics. In *Proc. IEEE-RAS International Conference on Humanoid Robots (Humanoids 07)*.

[Walker and Orin, 1982] Walker, M. and Orin, D. (1982). Efficient dynamic computer simulation of robotic mechanisms. *ASME Journal of Dynamic Systems Measurement and Control*, 104:205–211.

[Yakey et al., 2001] Yakey, J. H., LaValle, S. M., and Kavraki, L. E. (2001). Randomized path planning for linkages with closed kinematic chains. *IEEE Transactions on Robotics and Automation*, 17(6):951–958.

[Yamane et al., 2004] Yamane, K., Kuffner, J., and Hodgins, J. (2004). Synthesizing animations of human manipulation tasks. In *SIGGRAPH*.

[Yao and Gupta, 2005] Yao, Z. and Gupta, K. (2005). Path planning with general end-effector constraints: using task space to guide configuration space search. In *Proc. IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*.