Motion Planning With Constraints Using Configuration Space Approximations

Ioan A. Şucan and Sachin Chitta

***Abstract*— Robots executing practical tasks in real environments are often subject to multiple constraints. These constraints include orientation constraints: e.g., keeping a glass of water upright, torque constraints: e.g., not exceeding the torque limits for an arm lifting heavy objects, visibility constraints: e.g., keeping an object in view while moving a robot arm, etc. Rejection sampling, Jacobian projection techniques and optimization-based approaches are just some of the methods that have been used to address such constraints while computing motion plans for robots performing manipulation tasks. In this work, we present an approach to handling certain types of constraints in a manner that significantly increases the efficiency of existing methods. Our approach focuses on the sampling step of a motion planner. We implement this step as the drawing of samples from a set that has been computed in advance instead of the direct sampling of constraints. We show how our approach can be applied to different constraints: orientation constraints on the end-effector of an arm, visibility constraints and dual-arm constraints. We present simulated results to validate our method, comparing it to approaches that use direct sampling of constraints.**

I. Introduction

Robots operating in real environments need to plan their motions to operate safely. A motion plan is a continuous path between the start state of the robot and a goal region, subject to a set of constraints. Early research in motion planning typically required collision avoidance as the sole constraint [1], but many practical problems require imposing additional constraints. In this paper, we focus on motion planning with constraints that arise for manipulators (fixed or mobile) operating in real environments. One typical example of such constraints is the orientation constraint that needs to be maintained for an end-effector carrying a glass of water.

Sampling-based algorithms [2], [3] have been widely used to plan motions for manipulators due to their ability to plan efficiently in high-dimensional configuration spaces (e.g., [4]–[7]). Furthermore, sampling-based planners provide a generic framework that can handle a variety of constraints and compute plans for a variety of robots (e.g., [8], [9]). For the remainder of the paper, we will consider sampling-based motion planners used to compute motion plans subject to two types of constraints: collision avoidance constraints and task constraints. Collision avoidance refers to computing motion plans such that the robot does not collide with the environment or with itself. In this paper, task constraints are ones that have the potential to reduce the dimensionality of the valid part of configuration space in a manner that does not easily allow reparameterization (i.e., a different configuration space cannot be explicitly defined such that it inherently satisfies the given task constraints). The implicit subspace defined by a task constraint is also referred to as a "constraint manifold". The specific task constraints we consider in this paper can be verified geometrically, i.e., the constraints do not depend on velocities, accelerations, etc. When such constraints reduce the number of degrees of freedom of the system, the term "kinematic constraints" is often used. Thus, motion planning is performed under geometric constraints only (sometimes called "path planning" [2]).

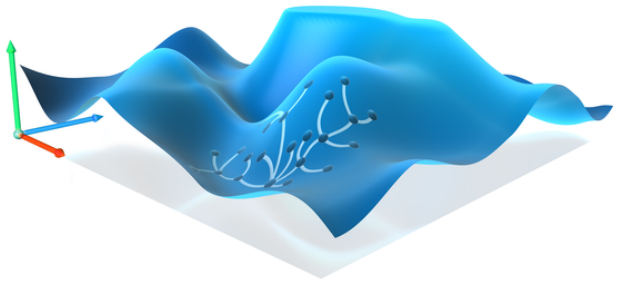

Fig. 1. A representation of a sampling-based planner exploring a lower dimensional constraint manifold.

In general, respecting task constraints is significantly more difficult than respecting collision avoidance constraints because the process of generating samples that satisfy task constraints can be more complex when the dimensionality of the constraint manifold is lower than that of the configuration space itself. To address this issue, specialized algorithms can be used [10]–[19].

In this paper we present a new approach to address task constraints during motion planning in a manner that is complementary to previous work. Our approach starts with computing an approximation of the constraint manifold offline, using already existing specialized sampling algorithms (e.g., [10]–[15]). Then, we use this approximation to plan on the constraint manifold directly instead of planning in the full configuration space (see Figure 1). Our approach can be used without modifying the planning algorithm itself. By construction, the constraint manifold satisfies task constraints, so the motion planner will effectively have to address only the collision avoidance constraint. The advantage of our method is that instead of sampling the configuration space using a relatively slow, specialized sampling algorithm, the constraint manifold can be sampled very quickly online using data structures computed offline. We show, through simulated experiments, that our approach is able to reduce planning times by orders of magnitude in some cases, across different planning algorithms.

Ioan Şucan and Sachin Chitta are with Willow Garage Inc., Menlo Park, USA. (isucan,sachinc)@willowgarage.com

The offline computation part of our approach relies on previous work, which is discussed in Section II. Details on our approach are presented in Section III and the experimental evaluation is in Section IV. Conclusions follow in Section V.

II. Background and Related Work

A. Sampling-based Motion Planning

Sampling-based algorithms are state-of-the-art techniques for solving the motion planning problem. Such algorithms operate using the concept of a configuration space $\mathcal{C}$ (or state space, when dynamics are considered) [2], [3], a space in which the entire robotic system is represented as a point. For the case of a robot arm, points in $\mathcal{C}$ usually represent the joint values (the degrees of freedom) of the robot arm. The set of points in the configuration space that corresponds to valid configurations of the robot is referred to as the valid part of the configuration space: $\mathcal{C}_v \subseteq \mathcal{C}$. Motion planning algorithms are used to find continuous paths between given start and goal configurations, such that the paths lie entirely in $\mathcal{C}_v$. Sampling-based planners are usually probabilistically complete [20], which means that they eventually find a solution when one exists, but do not terminate if no solution exists.

Sampling-based motion planners have been shown to be efficient for solving problems in high-dimensional spaces. The core idea of such motion planning algorithms is to approximate the connectivity of $\mathcal{C}_v$. From a highly abstract perspective, all sampling-based motion planning algorithms require at least the following two components that interact to construct the approximation of $\mathcal{C}_v$:

1) *Sampling*. This is the process of generating samples for the approximation of $\mathcal{C}_v$. This process usually relies on more basic functionality that allows sampling $\mathcal{C}$, followed by constraint evaluation (e.g., collision detection) and potentially constraint enforcement.
2) *Local planning*. This is the process of connecting pairs of configurations in $\mathcal{C}$ and checking whether the resulting motion segment lies in $\mathcal{C}_v$.

The abstraction above is not intended to be detailed or complete, but to point out the main components that we consider when planning with constraints.

When dealing with collision avoidance constraints only, the sampling and local planning processes are straightforward. If $\mathcal{C}_v \neq \emptyset$, sampling $\mathcal{C}$ (e.g., using a uniform distribution) will eventually produce samples that are also in $\mathcal{C}_v$ (i.e., rejection sampling usually works). The local planning process can follow a similar strategy: if the straight line between two valid configurations lies in $\mathcal{C}_v$, it can be included in the approximation to $\mathcal{C}_v$.

The statement above includes the hidden assumption that the volume of $\mathcal{C}_v$ is not negligible within $\mathcal{C}$. This assumption usually holds true for collision avoidance constraints alone, but fails if we include task constraints that make $\mathcal{C}_v$ be a lower-dimensional constraint manifold. In this case, the probability of finding configurations in $\mathcal{C}_v$ by rejection sampling from $\mathcal{C}$ tends to 0. To address this issue, a number of techniques can be employed, as described below.

B. Planning on Constraint Manifolds

The problem of planning on constraint manifolds is one that has been well studied in the robotics literature [10]–[19].

A possible approach is to change the sampling mechanism of a planner such that samples drawn from $\mathcal{C}$ are projected to the constraint manifold. One of the first projection techniques that was used within sampling-based motion planning is Randomized Gradient Descent (RGD) [10]. This method relies on sampling the robot's configuration space iteratively, towards the constraint manifold, for the specific case of closed chains. Improvements to the RGD algorithm to make it address more general constraints were developed as well [13]. The kinematics-based roadmap [11] and the Random Loop Generator (RLG) algorithm [12] were also designed to address the issue of sampling closed-loop chains. The kinematics-based roadmap breaks loops into multiple open chains and then uses inverse kinematics to enforce closure constraints. RLG relies on sampling only a subset of the joint variables that make up the closed chain and then solving for the rest.

A number of projection techniques were evaluated by Stilman [14] where it was suggested that techniques based on the pseudo-inverse of the Jacobian may be more efficient. Berenson et al. [15] have shown how Jacobian pseudo-inverse projections as well as rejection sampling can be used to address a number of task constraints applied to a robot's end-effector. The work of Suh et al. [18] expands on this approach and performs planning on the tangent bundle of $\mathcal{C}_v$, performing projections to the constraint manifold only when prescribed boundaries or error limits have been reached, thus reducing the number of projection operations, but at the cost of increased approximation error.

The work of Porta et al. [19] shows how to construct an atlas for the constraint manifold at runtime, and plan directly on that constrained space.

The idea of storing configurations that satisfy a subset of the constraints imposed during planning exists in previous work as well. Kuffner et al. [16] use precomputed sets of stable poses in their work with planning for humanoids. A similar use is also found in the work of Zacharias et al. [17], where precomputed sets of configurations that appear more human-like are used.

III. Approximating Constraint Manifolds

Given a particular set of task constraints, we can compute an approximation of the entire constraint manifold, subject to the task constraints alone (i.e., not considering collision avoidance constraints). This approximation has the form of a graph $G = (V, E)$, the *Approximation Graph*, such that all vertices in V correspond to configurations that satisfy the task constraints, and crossing any edge in E does not leave the constraint manifold. This graph is essentially the same as the roadmap data structure constructed by PRM [4].

A. Generating an Approximation Graph

Algorithm 1 describes the process of generating the Approximation Graph given a set of constraints c. Lines 1–4 of the algorithm generate a requested number of configurations

that satisfy the constraints c. The `Sample()` function can be implemented using techniques as described in previous work [10]–[15]. Depending on the type of constraint, various sampling strategies can be efficient. Once the requested number of configurations is computed, a number of valid expansion directions are computed for each configuration as well (lines 5–10). These directions are stored as edges in the Approximation Graph and encode the information that task constraints are maintained while moving along such edges. This holds true only if the local planner used by the planner is the same as the one used by `ValidEdge()` (line 8).

Algorithm 1 Generate Constraint Manifold Approximation

Input: c: task constraint; n_s: # configs; n_e: # edges/config
Output: Approximation Graph

```
1:  Approx = EmptyApproximation()
2:  while ConfigCount(Approx) < n_s do
3:    if Sample(c, x) then
4:      AddSample(Approx, x)
5:  for i = 0 to ConfigCount(Approx)-1 do
6:    j = 0
7:    while OutEdges(Approx, i) < n_e and
              j < ConfigCount(Approx) do
8:      if i ≠ j and ValidEdge(Approx, i, j) then
9:        AddEdge(Approx, i, j)
10:     j = j + 1
11: return Approx
```

B. Planning Directly on the Constraint Manifold

Consider a problem instance that asks for a motion plan in space $\mathcal{C}_v \subseteq \mathcal{C}$, where $\mathcal{C}_v$ corresponds to a set of constraints c. Instead of running the planner on the space $\mathcal{C}$ and sampling configurations that satisfy c, our approach runs that planner on the space $\mathcal{C}' \subseteq \mathcal{C}$, $\mathcal{C}_v \subseteq \mathcal{C}'$, subject to the same set of constraints c. $\mathcal{C}'$ is the constraint manifold that satisfies $c' \subseteq c$, a subset of the original constraints. The typical setup would be to have c' correspond to the task constraints and c to include additional constraints such as collision avoidance. Since $\mathcal{C}'$ cannot be represented explicitly, we use the corresponding Approximation Graph representation, computed by Algorithm 1. If the constraints $c \backslash c'$ further reduce the dimensionality of the constraint manifold, projection techniques as mentioned in Section II-B would still need to be used and the usefulness of the Approximation Graph is severely reduced.

C. Sampling a Constraint Manifold using the Approximation Graph

Sampling-based algorithms use a variety of distributions for sampling the configuration space they search. At a low level, most of these distributions can be implemented in terms of a few core functions. The planning algorithms used in this work require only two such core functions: (1) sampling configurations uniformly at random, and (2) sampling configurations uniformly at random within a ball around a particular configuration. We will now show how to implement these functions using a constraint manifold approximation. Similar techniques can be developed for other sampling functions that implement different distributions (e.g., normal distribution), if needed.

The process of sampling configurations uniformly at random is simply replaced by sampling an integer value $i \in [0, n_s - 1]$, where n_s is the number of configurations in the Approximation Graph. The generated sample is then the configuration at index i in the Approximation Graph. This process is significantly faster than actually generating a new sample on the constraint manifold. The uniformity of this sampling process is contingent on the `Sample()` function from Algorithm 1 producing samples uniformly at random. Unfortunately, due to the use of projection techniques, that uniformity is not always guaranteed.

Sampling configurations uniformly in a ball is more involved. For efficiency reasons we chose to attach a tag to every configuration in the Approximation Graph. That tag is in fact the index position of the configuration in the Approximation Graph. When a planner requests sampling nearby a configuration ns, Algorithm 2 is called. If ns was previously sampled from the Approximation Graph, it has an associated tag. Using that tag, the edges from the Approximation Graph that correspond to ns can be found (lines 2,3). Then sampling a state in the vicinity of ns can be implemented as choosing one of the edges (lines 4,5) and sampling a configuration along that edge (lines 6–8), such that the sampled configuration is within the requested ball. Since the sampled configuration is along an edge in the Approximation Graph, it has higher probability of being valid. If no tag is found for ns (e.g., if the configuration ns was computed by the planner using interpolation between other previously sampled configurations), a random direction remains to be used (towards a random configuration in the Approximation Graph) (line 1). The sampling process described by Algorithm 2 is efficient, but it is not uniform for multiple reasons. We use a source of samples that may not be uniform to begin with (due to the `Sample()` function not necessarily producing uniform samples) and the step at line 7 does not respect the fact that the volume of the space is concentrated near the surface of the ball in higher dimensional spaces. Even with this caveat, experimental results indicate this sampling procedure can work well.

Algorithm 2 Sample Inside Ball

Input: ns: config. to sample around ; d: ball radius
Output: x: a config. at distance $\leq d$ from config. ns

```
1: x = RandomElement(Configurations(Approx))
2: if Tag(ns) ∈ [0, ConfigCount(Approx)−1] then
3:   nbh = Neighbors(Approx, Tag(ns))
4:   if UniformRand(0, 1) > 1/|nbh| then
5:     x = RandomElement(nbh)
6: if Distance(ns, x) > d then
7:   d' = UniformRand(0, d)
8:   x = run local planner from ns to x up to distance d'
9: return x
```

D. Parameterizing the Approximation Graph

The same task constraints can often arise in different robot applications. In some cases, however, there may be

complex constraints that vary slightly. For example, when grasping objects such as a tray using two arms, the distance between the end-effectors has to match the size of the grasped object. This constraint can be parameterized using a single parameter, the distance between the two end-effectors along the length of the tray.

For constraints whose differences can be described using one parameter, it is possible to construct an Approximation Graph for a range of values for that parameter. The set of configurations can then be sorted with respect to that parameter to make look-up easier. When using the Approximation Graph during planning, a binary search can be performed to identify the usable range of configurations in the Approximation Graph for the value of the parameter specified by the problem. The size of this range depends on the error that is allowed in the value of the parameter for the constraint. This approach works only if the allowed error is above 0. The same sorting idea can be used to identify usable ranges of Approximation Graph edges when sampling with Algorithm 2.

When describing constraints by more than one parameter that is allowed to vary, it may be more difficult to quickly identify the usable range of configurations. One option is to sort offline by one parameter only and sort by additional parameters online, after the range satisfying the first parameter has been identified. After the subsequent online sorting step, a further reduced range can be identified.

With this simple idea, a continuum of similar constraints can be handled directly. There are two factors to keep in mind when constructing Approximation Graphs for parameterized constraints: (1) the smaller the variation allowed in the parameters of a constraint, the larger the Approximation Graph needs to be and (2) the number of configurations in the Approximation Graph is expected to increase exponentially with the number of constraint parameters considered.

E. Probabilistic Completeness

As described, the method we presented is not probabilistically complete because it limits the sampling of the configuration space to a fixed set. A simple approach to regain probabilistic completeness is to run a procedure that generates a cache of configurations on the constraint manifold in parallel, while the planner is running. When the planner needs to sample a configuration, it draws it from the cache generated at runtime, if such a sample is available, otherwise it draws it from the Approximation Graph. The procedure that fills the cache of configurations is likely slower than just drawing configurations from the Approximation Graph, but allows the planner to be probabilistically complete.

IV. Experiments

A. Task Constraints

We evaluated the benefits of using an Approximation Graph for three sets of constraints. We provide here a description of each constraint. We compare our approach (using the Approximation Graph) for motion planning with these task constraints to a baseline approach for dealing with these constraints without using the Approximation Graph. In this context, we also describe the sampling and local planning techniques used for each constraint. These techniques are used both for offline computation of the Approximation Graph (in our approach) and as part of motion planning itself (in the baseline approach). The sampling techniques themselves are not a contribution of this work and can be replaced by any means introduced in previous work.

The constraints we consider are applied to the arms of the PR2 robot, which have 7 degrees of freedom each.

1) End-effector orientation constraint: When manipulating objects in the environment it is often necessary to maintain their orientation fixed, or approximately fixed. For example, when moving a glass of water, its content should not be spilled. To respect this constraint, the orientation of the end-effector on an arm carrying the glass needs to stay fixed: i.e., the roll and pitch are set to specific values, depending on the grasp, while the yaw is allowed to vary (with respect to a global frame of reference). For this paper, we consider *upright* constraints, i.e., constraints where the roll and pitch of the end-effector are 0.

A joint configuration that satisfies these constraints can be sampled using inverse kinematics: the pose of the end-effector is sampled (values for the roll and pitch being always the same) and inverse kinematics is executed to compute the corresponding joint values. This procedure leads to a success rate of 99.5%, i.e., 99.5% of the time inverse kinematics succeeds at finding a configuration that lies on the constraint manifold. If an analytical inverse kinematics solver is not available, numerical solvers can be used instead. As mentioned in Section II-B, Jacobian-based projection techniques can also be used to project points that have been directly sampled in joint space onto the constraint manifold.

Because this constraint is imposed on the end-effector, the local planner performs linear interpolation between positions of the end-effector and spherical-linear interpolation between the orientations of the end-effector. The interpolated poses are mapped onto joint configurations for the arm using inverse kinematics. Interpolating between configurations in this manner is more likely to maintain the validity of the constraint along a motion segment.

2) Visibility cone constraint: When analyzing a grasped object in more detail, it is often necessary to move that object while keeping it within a sensor's field of view. This is useful, for example, for robots like the PR2 which need to move a checker-board target in their field of view during calibration. This constraint can be specified using a *visibility constraint* consisting of two parts: (1) the visibility cone to a specified area (labelled the cone base) on the grasped object's surface needs to remain unobstructed by robot links, and (2) the axis of the visibility cone is within a maximum angle with respect to the sensor axis, i.e., the object remains in the field of view of the sensor. In this paper we used a cone base of radius of 0.1m and a maximum angle of 30 degrees with respect to the sensor.

This constraint does not reduce the dimension of the constraint manifold. Sampling of valid configurations can

thus be performed with rejection sampling. The sampling success rate in this case is 0.7%. Thus, the sampling step is a relatively expensive operation when performed online as part of motion planning. In our approach though, this sampling step is primarily employed offline in the generation of the Approximation Graph. Online, the motion planner samples directly from the Approximation Graph, saving a large amount of effort. The local planner performs linear interpolation between configurations in joint space.

3) Dual arm grasping constraint: For larger objects, robotic systems with multiple arms may grasp an object with two arms: one on each side of the object. Depending on the type of object, there may be additional orientation constraints for each end-effector. We consider upright constraints, fixing the roll and pitch for each end-effector to be 0.

The sampling process consists of sampling the pose of one end-effector, then computing the pose for the other end-effector (using information about the grasp on the object) and then performing inverse kinematics for both end-effectors. Since this is a closed-loop chain, a number of other different methods for sampling could have been used as well, as discussed in Section II-B. The success rate of the sampling method we used is 13.3%. Again, as for visibility constraints, sampling will be expensive when performed online. The local planner is the same as for the orientation constraints described in *1)*.

B. Experimental Setup

We used a number of state-of-the-art sampling-based planners (implemented in OMPL [21]): RRT (Rapidly-exploring Random Trees) [22], RRT-Connect (a bi-directional search version of RRT) [6], an implementation of KPIECE (Kinodynamic Planning by Interior-Exterior Cell Exploration) [23] under geometric constraints solely, LBKPIECE (a lazy bi-directional implementation of KPIECE) [23] and SBL (Single-query Bi-directional probabilistic roadmap planner with Lazy collision checking) [7].

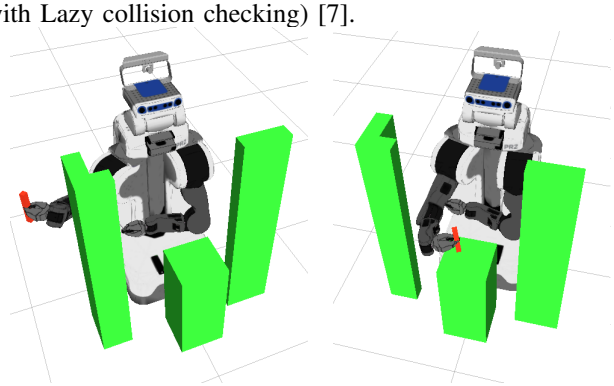

Fig. 2. PR2 operating among obstacles while keeping roll and pitch fixed for the right arm end-effector. Left: Start configuration. Right: Goal configuration.

We tested these algorithms on three different environments using the PR2, each environment using a corresponding task constraint from Section IV-A. Environment 1, shown in Figure 2, asks for the right arm to move an object from the right side of the environment to a stool in front of the robot, while respecting the end-effector orientation constraint. Environment 2, shown in Figure 3, asks for the right arm to move a small board from the left side to the right side of a pole, while maintaining the visibility cone constraint with respect to the head stereo camera of the PR2. Environment 3, shown in Figure 4, asks for the robot to use both its arms to move a tray from the robot's left side to its right side, so it can be placed on the table in front of the robot, while maintaining the dual arm constraint.

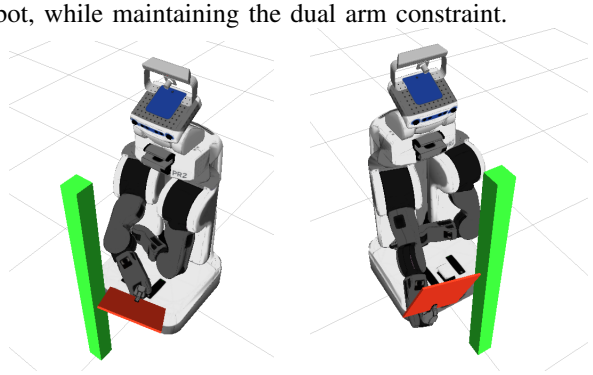

Fig. 3. PR2 operating among obstacles, maintaining visibility from head stereo camera to a grasped object. Left: Start configuration. Right: Goal configuration.

For each of the environments, 10 different approaches were tested: (1) A baseline approach where an Approximation Graph was not used but specialized algorithms were used for sampling the constraint manifold, and (2) 9 approaches where approximations of the constraint manifold were employed. We used Approximation Graphs with 1000, 5000 and 10000 configurations and with 0, 20 or 100 edges per configuration. Note that a different Approximation Graph needs to be created for each type of constraint, but the same Approximation Graph can be used across different environments as long as the constraint stays the same. For each experiment, all values were averaged over 100 planner executions. Experiments 1 and 2 have a time limit of 15 s per planner execution, and Experiment 3 has a time limit of 300 s per planner execution. The machine we used had an Intel i7-2600 CPU (3.4Ghz) and 16 GB RAM. All planners are single threaded, but the database computation is multi-threaded (8 threads).

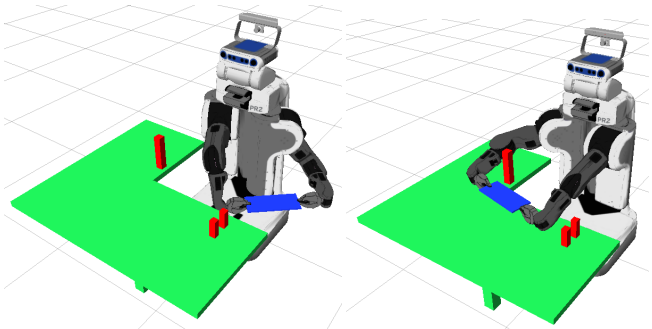

Fig. 4. PR2 operating among obstacles, keeping roll and pitch fixed for both arms while grasping a tray. Left: Start configuration. Right: Goal configuration.

C. Experimental Results

The results from the experiments are shown in Table I for Environment 1, Table II for Environment 2 and Table III for Environment 3. The first column in each table indicates

TABLE I

"AVERAGE TIME (S)" / "SUCCESS RATE" IN ENVIRONMENT 1.

	RRTConnect	LBKPIECE	SBL	RRT	KPIECE
no approx.	1.70 / 100%	3.98 / 100%	2.86 / 100%	3.02 / 94%	4.03 / 86%
1000/0	0.99 / 99%	0.55 / 100%	0.68 / 100%	0.90 / 84%	2.29 / 96%
1000/20	0.96 / 100%	0.49 / 100%	0.62 / 100%	1.10 / 100%	2.23 / 99%
1000/100	1.01 / 93%	0.51 / 100%	0.67 / 100%	0.89 / 91%	2.43 / 98%
5000/0	0.71 / 100%	0.57 / 100%	0.69 / 100%	1.14 / 93%	2.43 / 99%
5000/20	0.93 / 100%	0.54 / 100%	0.63 / 100%	1.33 / 82%	2.96 / 94%
5000/100	0.55 / 100%	0.55 / 100%	0.66 / 100%	1.05 / 100%	2.64 / 91%
10000/0	0.70 / 100%	0.54 / 100%	0.67 / 100%	1.28 / 85%	3.41 / 90%
10000/20	0.66 / 100%	0.59 / 100%	0.67 / 100%	1.12 / 91%	3.00 / 93%
10000/100	0.83 / 100%	0.58 / 100%	0.65 / 100%	1.19 / 92%	2.52 / 94%

TABLE II

"AVERAGE TIME (S)" / "SUCCESS RATE" IN ENVIRONMENT 2.

	RRTConnect	LBKPIECE	SBL	RRT	KPIECE
no approx.	0.95 / 100%	3.90 / 100%	2.47 / 100%	4.71 / 100%	0.45 / 100%
1000/0	0.36 / 100%	0.59 / 100%	1.01 / 100%	0.42 / 100%	3.46 / 88%
1000/20	0.41 / 100%	0.89 / 100%	1.42 / 100%	0.49 / 100%	2.29 / 91%
1000/100	0.31 / 100%	0.84 / 100%	1.32 / 100%	0.37 / 100%	3.16 / 92%
5000/0	0.31 / 100%	0.76 / 100%	1.17 / 100%	0.38 / 100%	3.01 / 91%
5000/20	0.29 / 100%	0.89 / 100%	1.36 / 100%	0.37 / 100%	4.08 / 89%
5000/100	0.32 / 100%	0.99 / 100%	1.35 / 100%	0.44 / 100%	4.03 / 66%
10000/0	0.33 / 100%	0.78 / 100%	1.25 / 100%	0.43 / 100%	5.27 / 66%
10000/20	0.29 / 100%	0.99 / 100%	1.20 / 100%	0.44 / 100%	4.13 / 70%
10000/100	0.37 / 100%	0.95 / 100%	1.25 / 100%	0.43 / 100%	3.75 / 84%

TABLE III

"AVERAGE TIME (S)" / "SUCCESS RATE" IN ENVIRONMENT 3.

	RRTConnect	LBKPIECE	SBL	RRT	KPIECE
no approx.	156.9 / 100%	N/A / 0%	211.7 / 15%	182.9 / 90%	76.1 / 100%
1000/0	0.26 / 100%	1.00 / 100%	2.22 / 100%	0.43 / 100%	0.62 / 100%
1000/20	0.21 / 100%	0.40 / 100%	0.76 / 100%	0.41 / 100%	1.22 / 100%
1000/100	0.39 / 100%	0.51 / 100%	0.80 / 100%	0.67 / 100%	1.25 / 100%
5000/0	0.27 / 100%	1.29 / 100%	2.96 / 100%	0.45 / 100%	0.60 / 100%
5000/20	0.27 / 100%	0.51 / 100%	0.79 / 100%	0.48 / 100%	0.78 / 100%
5000/100	0.28 / 100%	0.45 / 100%	0.68 / 100%	0.61 / 100%	0.74 / 100%
10000/0	0.24 / 100%	1.49 / 100%	3.29 / 100%	0.54 / 100%	0.76 / 100%
10000/20	0.25 / 100%	0.52 / 100%	0.89 / 100%	0.49 / 100%	0.77 / 100%
10000/100	0.27 / 100%	0.39 / 100%	0.72 / 100%	0.47 / 100%	0.67 / 100%

the size of the Approximation Graph: "*nr. configurations / nr. edges per configuration*". Subsequent columns show the "*average runtime (seconds) / success rate*". The time per thread taken for offline generation of the Approximation Graphs using 8 threads, as well as the size (in MB) of the resulting data structure are shown in Table IV.

As we can see from Tables I, II and III, the runtimes shown in the top rows (corresponding to the baseline approach for planning without using the Approximation Graph) are much higher than the runtimes shown in subsequent rows (corresponding to the use of Approximation Graphs of different sizes). One exception is for the KPIECE planner operating in Environment 2. The fact that samples with a higher chance of being valid are sampled leads to overall faster execution for the cases where the Approximation Graph was employed.

The one exception for KPIECE happens when using the visibility constraint, which does not reduce the dimensionality of the configuration space. The exploration strategy KPIECE employs performs better when not limited by the number of configurations in the Approximation Graph. The behaviour of KPIECE with the Approximation Graph could possibly be improved with some parameter tuning, but we made no attempt to tune the parameters of any of the planners – automatic settings provided by OMPL were used.

The RRT and RRTConnect algorithms do not use sampling in the vicinity of configurations, so their runtime when using the Approximating Graphs does not vary when different numbers of edges are used. The other algorithms, however, do use this functionality. It is still difficult to see the benefits of using edges in the Approximation Graphs, since the use of Approximation Graphs without edges already makes the problems much easier to solve. For the first two environments, the probability that a motion segment will respect task constraints, given that its endpoints respect the constraints, is relatively high, due to the choice of local planner. For Environment 3 we see a clear improvement when using edges in the Approximation Graph for the planners that use lazy collision checking (SBL and LBKPIECE). This is to be expected since including the edges in the Approximation Graph leads to a higher density of valid motions in the data structures constructed by lazy planners.

The significant difference in execution time between planning with and without and Approximation Graph for experiment 3 (over two orders of magnitude) is consistent with the difference in sampling speed: generating samples for experiment 3 can be done at a rate of approximately 300 samples per second while just copying samples from the Approximation Graph can be done at over 70000 samples per second. The reduced success rate of SBL and LBKPIECE is also expected because these planners typically generate many more samples than non-lazy algorithms.

To test the performance when constraint parameters vary, we performed additional experiments for Environment 3, with objects of different sizes. In this case, the constraint is parameterized by a single parameter (d): the distance between the wrists of the end-effectors. This parameter can be adjusted for grasping objects of different sizes. We generated an Approximation Graph of 10^5 configurations and 0 edges, allowing this distance to vary, resulting in a 22.9 MB data structure computed in 35.26 seconds (runtimes are lower than those in Table IV because the enforced constraint is simpler). We did not include edges in this Approximation Graph because the required number of edges per configuration would have to be fairly large to be useful.

We ran tests for desired values of d between 0.6m to 0.9m, at increments of 0.1m (shown in Table V). For each desired value of d, a tolerance of 5mm was allowed. Since the configurations in the Approximation Graph were already

TABLE IV

COMPUTATIONAL RESOURCES FOR APPROXIMATION GRAPH COMPUTATION USING 8 THREADS.

Approximation graph	Environment 1 time (s)	Environment 1 MB	Environment 2 time (s)	Environment 2 MB	Environment 3 time (s)	Environment 3 MB
1000/0	0.4	0.1	0.3	0.1	6.5	0.2
1000/20	0.5	0.3	1.0	0.2	10.3	0.4
1000/100	1.3	0.8	3.5	0.8	15.3	0.7
5000/0	0.9	0.6	1.1	0.3	31.0	1.1
5000/20	1.9	1.4	4.5	1.1	62.4	1.9
5000/100	5.7	4.4	17.7	4.1	126.2	4.4
10000/0	1.7	1.2	2.1	0.7	59.6	2.3
10000/20	3.7	2.8	9.0	2.2	134.5	3.9
10000/100	11.4	8.8	35.6	8.3	309.7	9.3

TABLE V

DIFFERENT CONSTRAINT PARAMETERS FOR THE DUAL ARM CONSTRAINT USING A SORTED APPROXIMATION GRAPH.

d	n_s	RRTConnect	LBKPIECE	SBL	RRT	KPIECE
0.6	2162	0.25 / 100%	0.70 / 100%	1.32 / 100%	0.90 / 100%	0.86 / 100%
0.7	1731	0.32 / 100%	1.46 / 100%	2.92 / 100%	0.68 / 100%	0.88 / 100%
0.8	1475	0.52 / 100%	1.71 / 100%	3.49 / 100%	0.80 / 97%	1.81 / 99%
0.9	1351	0.51 / 100%	2.17 / 100%	5.96 / 100%	0.87 / 95%	4.33 / 95%

sorted in accordance to the distance between wrists, it is easy to identify the range of configurations that can be used for this constraint with a particular offset between the end-effectors. Table V shows the distance between the wrists of the end-effectors (column d), the number of configurations in the allowed range around the desired value of d (column n_s) and the averaged runtimes and success rates for the evaluated planners. We notice that success rates are maintained high (always 100% for bi-directional planners) and runtimes are still very small, thus suggesting that parameterization of constraints with a few parameters is feasible.

D. Discussion

The results in the previous section show that our approach can provide significant benefits for motion planning with task constraints. It also shows that our approach would be feasible for use with constraints that are parameterized by a few parameters. For parameterized constraints like the dual-arm constraint, the scalability of our approach could be improved by associating Approximation Graphs directly to an object. This is a natural step since grasp databases are already used to store possible grasps for different objects. Thus, it makes sense to also associate a small Approximation Graph for each object in the grasp database. Furthermore, it is likely that the benefits of including valid edges in the Approximation Graph would be more apparent if the local planners used did not tend to satisfy the planning constraints (as was the case for our experiments).

The approach we introduce in this work relies on the discretization of continuous spaces. As such, probabilistically complete motion planners that plan using the space representation we discuss in this work will no longer be probabilistically complete, but this caveat can be removed if the procedure described in Section III-E is also implemented.

V. CONCLUSIONS

Using offline computed approximations of constraint manifolds is useful for sampling-based motion planning because the sampling speed can be increased by as much as two orders of magnitude and the probability of sampling valid configurations is significantly higher. Furthermore, our approach does not require changing existing motion planners and can use any of the previously existing techniques for generating samples on the constraint manifold. In future work, we intend to explore the application of our approach to more constraints including torque constraints and constraints arising from human-robot interaction.

ACKNOWLEDGEMENTS

The authors would like to thank Marius Şucan for creating Figure 1.

REFERENCES

[1] J. T. Schwartz and M. Sharir, "On the piano movers' problem: General techniques for computing topological properties of real algebraic manifolds," *Communications on Pure and Applied Mathematics*, vol. 36, pp. 345–398, 1983.

[2] H. Choset, K. M. Lynch, S. Hutchinson, G. A. Kantor, W. Burgard, L. E. Kavraki, and S. Thrun, *Principles of Robot Motion: Theory, Algorithms, and Implementations*. MIT Press, June 2005.

[3] S. M. LaValle, *Planning Algorithms*. Cambridge, U.K.: Cambridge University Press, 2006, available at http://planning.cs.uiuc.edu/.

[4] L. E. Kavraki, P. Švestka, J.-C. Latombe, and M. H. Overmars, "Probabilistic roadmaps for path planning in high dimensional configuration spaces," *IEEE Transactions on Robotics and Automation*, vol. 12, no. 4, pp. 566–580, August 1996.

[5] D. Hsu, J.-C. Latombe, and R. Motwani, "Path planning in expansive configuration spaces," in *IEEE Intl. Conference on Robotics and Automation*, vol. 3, Albuquerque, April 1997, pp. 2719–2726.

[6] J. J. Kuffner and S. M. LaValle, "RRT-connect: An efficient approach to single-query path planning," in *IEEE Intl. Conference on Robotics and Automation*, San Francisco, April 2000, pp. 995–1001.

[7] G. Sánchez and J.-C. Latombe, "A single-query bi-directional probabilistic roadmap planner with lazy collision checking," *Intl. Journal of Robotics Research*, vol. 6, pp. 403–417, 2003.

[8] D. Berenson, S. Srinivasa, D. Ferguson, A. Collet, and J. J. Kuffner, "Manipulation planning with workspace goal regions," in *IEEE Intl. Conference on Robotics and Automation*, 2009, pp. 618–624.

[9] L. Jaillet, J. Cortés, and T. Siméon, "Sampling-based path planning on configuration-space costmaps," *IEEE Transactions on Robotics*, vol. 26, no. 4, pp. 635–646, August 2010.

[10] S. LaValle, J. Yakey, and L. Kavraki, "A probabilistic roadmap approach for systems with closed kinematic chains," in *IEEE Intl. Conference on Robotics and Automation*, Detroit, Michigan, May 1999, pp. 1671–1677.

[11] H. Li and N. Amato, "A kinematics-based probabilistic roadmap method for closed chain systems," in *Algorithmic and Computational Robotics - New Directions*, 2000, pp. 233–246.

[12] J. Cortés and T. Siméon, "Sampling-based motion planning under kinematic loop-closure constraints," in *Intl. Workshop on Algorithmic Foundations of Robotics*. Springer-Verlag, 2004, pp. 59–74.

[13] Z. Yao and K. Gupta, "Path planning with general end-effector constraints," *Robotics and Autonomous Systems*, vol. 55, no. 4, pp. 316–327, 2007.

[14] M. Stilman, "Task constrained motion planning in robot joint space," in *Intl. Conference on Intelligent Robots and Systems*, San Diego, California, October 2007, pp. 3074–3081.

[15] D. Berenson, S. Srinivasa, D. Ferguson, and J. J. Kuffner, "Manipulation planning on constraint manifolds," in *IEEE Intl. Conference on Robotics and Automation*, Kobe, Japan, May 2009, pp. 625–632.

[16] J. J. Kuffner, S. Kagami, K. Nishiwaki, M. Inaba, and H. Inoue, "Dynamically-stable motion planning for humanoid robots," *Autonomous Robots*, vol. 12, no. 1, pp. 105–118, 2002.

[17] F. Zacharias, C. Schlette, F. Schmidt, C. Borst, J. Rossmann, and G. Hirzinger, "Making planned paths look more human-like in humanoid robot manipulation planning," in *IEEE Intl. Conference on Robotics and Automation*, Shanghai, May 2011, pp. 1192–1198.

[18] C. Suh, T. T. Um, B. Kim, H. Noh, M. Kim, and F. C. Park, "Tangent space RRT: A randomized planning algorithm on constraint manifolds," in *IEEE Intl. Conference on Robotics and Automation*, Shanghai, May 2011, pp. 4968–4973.

[19] J. M. Porta Pleite, L. Jalliet, and O. Bohigas Nadal, "Randomized path planning on manifolds based on higher-dimensional continuation," *Intl. Journal of Robotics Research*, vol. 31, no. 2, pp. 201–215, 2012.

[20] A. Ladd and L. Kavraki, "Measure theoretic analysis of probabilistic path planning," *IEEE Transactions on Robotics and Automation*, vol. 20, no. 2, pp. 229–242, 2004.

[21] I. A. Şucan, M. Moll, and L. E. Kavraki, "The Open Motion Planning Library," *IEEE Robotics and Automation Magazine*, 2012. [Online]. Available: http://ompl.kavrakilab.org

[22] S. M. LaValle, "Rapidly-exploring random trees: A new tool for path planning," Computer Science Dept., Iowa State University, Tech. Rep. 11, 1998.

[23] I. A. Şucan and L. E. Kavraki, "A sampling-based tree planner for systems with complex dynamics," *IEEE Transactions on Robotics*, vol. 28, no. 1, pp. 116–131, 2012.