Reachable Volume RRT

Troy McMahon, Shawna Thomas, Nancy M. Amato

Abstract— **Reachable volumes are a new technique that allows one to efficiently restrict sampling to feasible/reachable regions of the planning space even for high degree of freedom and highly constrained problems. However, they have so far only been applied to graph-based sampling-based planners. In this paper we develop the methodology to apply reachable volumes to tree-based planners such as Rapidly-Exploring Random Trees (RRTs). In particular, we propose a *reachable volume RRT* called RVRRT that can solve high degree of freedom problems and problems with constraints. To do so, we develop a reachable volume stepping function, a reachable volume expand function, and a distance metric based on these operations. We also present a reachable volume local planner to ensure that local paths satisfy constraints for methods such as PRMs.**

We show experimentally that RVRRTs can solve constrained problems with as many as 64 degrees of freedom and unconstrained problems with as many as 134 degrees of freedom. RVRRTs can solve problems more efficiently than existing methods, requiring fewer nodes and collision detection calls. We also show that it is capable of solving difficult problems that existing methods cannot.

I. Introduction

High degree of freedom (dof) motion planning problems are problems in which a robot (or other deformable object) consists of hundreds, or even thousands, of rigid bodies connected by planar, prismatic, or spherical joints. Motion planning for constrained systems is a variation in which the motion of a robot is limited by constraints. These constraints could require that the robot remain in contact with a surface, that it maintain a specific clearance, or that certain joints of the robot remain in contact with each other (e.g., closed chains). Such problems are particularly difficult because the constraints form a manifold in C-space, and the probability of randomly generating samples on this manifold is zero. High dof motion planning and constrained motion planning has applications in parallel robotics [10], grasping and manipulation ([5, 11]), computational biology and molecular simulations [12], and animation [3].

In [8, 9] we presented the concept of *reachable volumes* which denote the regions of space that the joints and end effectors of a robot can reach. We presented methods for computing reachable volumes and generating constraint satisfying samples. We showed these methods can be used to generate Probabilistic RoadMaps (PRMs) [4] in a wide variety of problems. Reachable volumes allows for combinations of planar, prismatic, and spherical joints. In contrast, most previous work focuses solely on planar robots with planar (and sometimes prismatic) joints.

Reachable volumes have not been applied to Rapidly-exploring Random Tree (RRT) [6] construction. The primary difficulty is the need to expand the RRT in such a way that the joints of the new sample are in their reachable volumes. This is needed to guarantee that the joint positions are feasible and that the sample satisfies the constraints.

In this paper, we show how to apply reachable volumes to RRTs. We present a novel method for stepping reachable volume samples to generate nearby samples that are also in their reachable volumes. We use this stepping method in an RV-Expand function to grow the RRT. We also present a reachable volume-specific distance metric and local planner.

We propose a reachable volume RRT, RVRRT, which uses RV-Expand and the reachable volume distance metric to construct the RRT. It can be applied to high dof problems that RRTs have previously been unable to solve. It can also be applied to constrained systems where it generates paths that are guaranteed to adhere to the problem's constraints.

We show experimentally that RVRRTs are more efficient at solving high dof problems and problems with constraints. We present results for environments with as many as 134 dof, which demonstrate that RVRRTs require less computation time and fewer nodes to solve problems than RRTs or dynamic domain RRTs (DDRRTs) [15]. We also show that they are capable of solving a series of difficult problems that other methods cannot solve. Finally, we show that the reachable volume local planner has little overhead compared to the commonly used straight line local planner while still satisfying constraints in the context of PRMs.

The main contributions of this work include:

- a reachable volume RRT which uses reachable volume stepping to generate constraint satisfying RRT samples,
- experimental results showing that reachable volume RRTs outperform other methods in a set of constrained and unconstrained systems up to 134 dof, and
- a reachable volume local planner which generates constraint satisfying local paths and a reachable volume distance metric for use with this local planner.

This research supported in part by NSF awards CNS-0551685, CCF 0702765, CCF-0833199, CCF-1439145, CCF-1423111, CCF-0830753, IIS-0916053, IIS-0917266, EFRI-1240483, RI-1217991, by NIH NCI R25 CA090301-11, by DOE awards DE-AC02-06CH11357, DE-NA0002376, B575363, by Samsung, IBM, Intel, and by Award KUS-C1-016-04, made by King Abdullah University of Science and Technology (KAUST). This research used resources of the National Energy Research Scientific Computing Center, which is supported by the Office of Science of the U.S. Department of Energy under Contract No. DE-AC02-05CH11231. Parasol Lab., Dept. of Comp. Sci. and Eng., Texas A&M Univ., College Station, Texas, USA. {tmcmahon,sthomas,amato}@cse.tamu.edu

II. Related Work

Rapidly-Exploring Random Trees [6], or RRTs, are widely used to solve motion planning problems. RRTs (Algorithm 1) first creates a tree T containing only the start node. They then iteratively add nodes to T via an expansion method until a certain node count N has been reached, the problem has been solved, or a set of goal configurations have been connected, depending on the variant. RRTs create new nodes by generating a random sample, locating its nearest neighbor in T, and applying an expansion method to that neighbor. If the new node is valid, it is added to T along with an edge connecting it to the nearest neighbor. RRTs have been shown to be *expansive* and thus probabilistically complete.

Algorithm 1 RRT method

Input: An environment env, a root configuration q_{root}, the number of nodes N, a step size δq and a distance metric dm

Output: Tree T containing N nodes rooted at q_{root}

```
1: T = q_root
2: while NumberOfNodes(T) < N do
3:    q_ran=RandomCfg()
4:    q_near=NearestNeighbor(q_ran,T,dm)
5:    q_new=Expand(q_near, q_ran, δq)
6:    if IsValid(q_new) then
7:       T.AddNode(q_new)
8:       T.AddEdge(q_near,q_new)
9: return T
```

RRTs have been applied to a wide variety of problems including those with many dof. However, they are poorly suited to handle spatial constraints because the probability of randomly sampling a constraint-satisfying configuration can be very small and in some cases approaches zero [7].

Dynamic-domain RRTs (DDRRT) [15, 14] can be used to construct RRTs along regions of space that satisfy a problem's constraints. They have been shown to solve constrained problems but are not applicable to more than 18 dof.

The Atlas-RRTs [2] simultaneously builds an RRT and constructs an atlas, which is a set of charts which locally parametrize constraint manifolds. The atlas is used to generate samples along the constraint manifolds, which are added to the RRT, while the RRT is used to guide the direction which the atlas is expanded.

Tangent Bundle RRTs [13] construct an RRT along a set of tangent bundles which approximate a problem's constraint manifolds. They then project the solution path onto the manifold. They have been shown to solve problems with closed chains and linkages with end-effector constraints up to 14 dof.

While DDRRTs, Altas RRTs, and Tangent Bundle RRTs are able to solve motion problems with constraints, none of these methods have been shown to be applicable to problems with more than 18 dof. In contrast, we show that RVRRTs are capable of solving problems with as many as 134 dof.

III. Reachable Volumes

In [8, 9] we introduced *reachable volumes* (RV) and proposed a reachable volume sampler for PRMs. That work addresses the problem of motion planning with linkage robots. Formally, a linkage robot consists of a set of links L that are connected by a set of joints J. It allows for spherical, planar, and prismatic joints as well as combinations of them. In contrast, most previous work assumes a planar robot comprised of planar (and sometimes prismatic) joints. For constrained problems, we assume a set of constraints $S = (S_1, \ldots, S_{|J|})$, where S_j is a subset of the environment in which joint j must be located.

RV-space is a space of the same dimensionality as the workspace in which the origin is located at one of the robot's joints or end effectors (referred to as the root). RV-space has no obstacles and does not take validity into account. The reachable volume of a joint or end-effector j is the set of points in RV-space that j can reach while satisfying S. Formally, this is the set of points p for which there exists a constraint-satisfying configuration in which j is located at p in RV-space. The reachable volume of a chain is the reachable volume of its end effector.

IV. Reachable Volume Primitives

In this section we propose a reachable volume stepping function, a reachable volume local planner, and a reachable volume distance metric.

A. Stepping in Reachable Volume Space

We define a method for stepping reachable volume samples to produce samples that are similar to the original (Figure 1 and Algorithms 2, 3). This method starts with an initial configuration q, a specified joint j, and a target position v. It perturbs q by moving j by δ in the direction of v (Figures 1(b) and 1(c)). It then updates the position of j and its descendants to ensure that all joints are in the reachable volume of their parents which will ensure that the joint positions defined the new sample q' correspond to a constraint-satisfying configuration (Figures 1(d) and 1(e)).

We observe the following about perturbing a joint:

Observation 1: If we perturb a joint j in such a way that it is still in the intersection of the reachable volumes of its parents, then only j's descendants need to be repositioned.

Observation 2: If we reposition a joint j and one of j's children is still in the intersection of the reachable volumes of both its parents, then all of the descendants of this child must also be in the intersection of the reachable volumes of their parents, and we do not need to reposition the child or its descendants.

Observation 3: If the original sample satisfied all joint position constraints in the environment, then the final configuration must also satisfy all joint position constraints.

Proof: Every joint in the new sample is located in the reachable volume of that joint which is a subset of any constraints on the position of the joint. ■

Based on Observation 1, we know that, with the exception of the children of j, all of the joints must still be located in

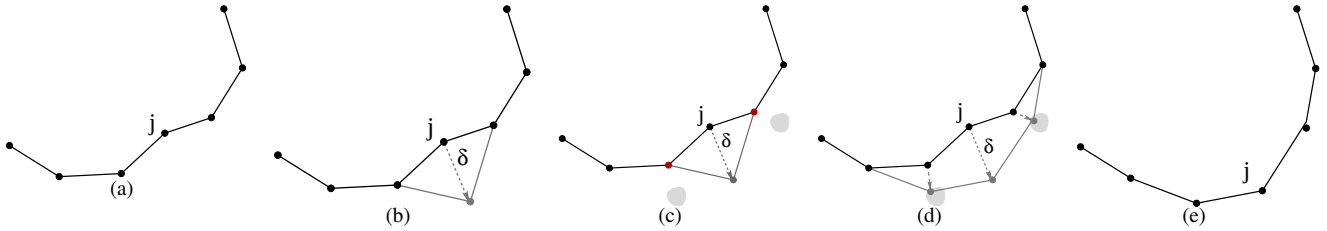


Fig. 1. Reachable Volume Stepping: We step one joint j by a distance of δ then update the j's descendants to be in their reachable volumes given j's new position. The gray regions are the reachable volumes of the third and fifth joints after j is stepped. These joints are repositioned to be in their reachable volumes (d) resulting in a configuration in which all joints are in their reachable volumes (e).

Algorithm 2 Reachable Volume Stepping

Function: RV-Step(q, j, p_{target}, δ)
Input: A cfg q, a joint j and a target position p_{target}, a stepping parameter δ
Output: A cfg in which the joint j has been perturbed by δ in the direction of p_{target}

```
let p_init = position of j in q
p_new=p_init+(p_target-p)*δ
if p_new ∈ RV(j.LeftParent) ∩ RV(j.RightParent)
    q_new=copy(q)
    Set position of joint j to be p_new in q_new
    Reposition(q_new,j.LeftChild)
    Reposition(q_new,j.RightChild)
    return q_new
return NULL
```

Algorithm 3 Method for repositioning descendants

Function: Reposition(q,j)
Input: A cfg q and a joint j
Output: A cfg with all $j' \in j \cap$ descendants(j) in the intersection of the reachable volume of their parents

```
if j ∈ RV(j.LeftParent) ∩ RV(j.RightParent)
    return q
Adjust position of j in q to be within RV(j.LeftParent)
  ∩ RV(j.RightParent)
if j.LeftChild ≠ NULL
    Reposition(q,j.LeftChild)
if j.RightChild ≠ NULL
    Reposition(q,j.RightChild)
    return q
```

the reachable volumes of their parents. We therefore only need to check if the descendants of j are still within the reachable volume of their parents. To do this we recursively test the descendants of j (Algorithm 3). If a joint is no longer in the reachable volume of its parents, we reposition it and recurse on its children. If we encounter a joint that is still in the intersection of the reachable volume of its parents, then we can stop by Observation 2.

To reposition a joint, we move it to a position that is in the intersection of the reachable volumes of the joint's parents and near the original position of the joint. When repositioning a joint we know that all previously repositioned joints were placed in their reachable volumes, so there must exist a sample for the positioning. The reachable volume of a joint being repositioned will therefore never be empty and there will always be a valid repositioning. The result of repositioning is a reachable volume configuration in which all of the joints are located in the intersection of the reachable volumes of their parents. Such a configuration must correspond to a feasible positioning of the joints in the linkage.

By applying reachable volume stepping to an initial RV-space sample, we can create a new sample that is near the original. These samples can be generated randomly by selecting a random target point or they can be generated in a specific direction by selecting a target in that direction. There are also a number of ways to select what joint to perturb.

One of the advantages of reachable volumes is that they may be computed in any order. We observe that reachable volume stepping only effects the joint being perturbed and its children, meaning that the ordering will determine which nodes are effected by a stepping operation. In this work we explore the following orderings:

- **Linear, end effector first:** construct the reachable volumes linearly with the end effector as the top. This limits the effects of stepping to the joints between the perturbed joint and the root.
- **Linear, root first:** construct the reachable volumes linearly with the root at the top. This limits the effects to joints between the perturbed joint and the end effectors.
- **Binary:** compute the reachable volumes in a binary manner (as described in [8]). This localizes the effect of stepping to the children of the perturbed joint.
- **Based on structure of robot:** compute the reachable volumes so that related parts of the robot are in the same branch of the reachable volume tree. For example, a grasper robot could be partitioned so that the fingers are in separate branches of the tree. Consequently, perturbing a joint in one of the fingers will only effect joints in that finger.

B. Reachable Volume Local Planner

We define a reachable volume local planner based on reachable volume stepping. This planner can be used by PRMs to find local paths that satisfy a problem's constraints.

The reachable volume local planner (Algorithm 4) connects two configurations, c_1 and c_2, by using reachable

volume stepping to move each joint to its position in the second configuration. To accomplish this, it performs a traversal of the joints in the reachable volume data structure (see Section IV-A). During each iteration of the traversal, it uses reachable volume stepping to move the joint from its position in c_1 to its position in c_2.

Algorithm 4 Reachable Volume Local Planner

Input: Cfgs c_1 and c_2, a step size δ
Output: Boolean value indicating if a path was found

```
1:  queue.push_back(j_root)
2:  while j=queue.pop_front() do
3:     c'=c_1
4:     while position of j in c' ≠ position of j in c_2 do
5:        p_target=position of joint j in c_2
6:        c' = RV-Step(c_1, j, p_target, δ)
7:        if c'==NULL OR invalid(c') then
8:           return false
9:     queue.push_back(children(j))
10: success = RigidBodyLocalPlanner(c',c_2)
11: return success
```

Figure 2 is an example of the reachable volume local planner (with a binary reachable volume ordering) being applied to a 5 link chain. The local planner first steps the end effector of the robot from its position in c_1 to its position in c_2 (2(a)). It then steps the third joint (2(b)), then the fourth joint (2(c)) to their positions in c_2. This results in the configuration c_2 (2(d)). Note that we are always stepping parents in the reachable volume data structure before their children to ensure that stepping a node will not change the position of any nodes that have already been stepped.

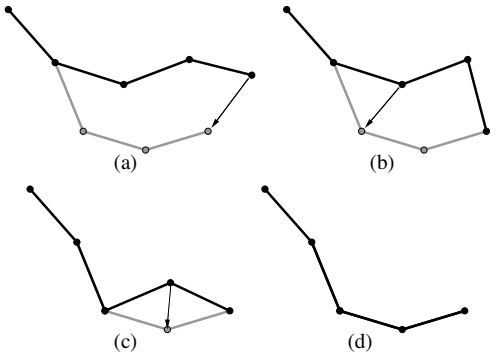


Fig. 2. The reachable volume local planner uses reachable volume stepping to move the joints of a linkage from their positions in one configuration (black) to their positions in a second (gray).

The sequence of steps covered by the local planner forms a path from c_1 to c_2. We can test the validity of this path by checking the validity at each step in the same manner as with other local planners. If the robot is free-based, we can use reachable volume sampling to generate paths between the internal configurations of c_1 and c_2 and apply a rigid body local planner to the translational and rotational coordinates. In most cases, we interleave the reachable volume local planner with the rigid body local planner so that we perform part of the rigid body transformation, apply the reachable volume sampler to the internal configuration, and perform the rest of the rigid body transformation. This is analogous to the rotate-at-S local planner [1].

C. Reachable Volume Distance

We define a reachable volume distance metric which measures the distance traversed during reachable volume stepping. The reachable volume expand function and local planner construct paths by moving each of the joints from its position in the first configuration to its position in the second configuration, so a good approximation would be the sum of the distances between the joints in reachable volume space (Figure 3). For free-base systems, a distance metric must also take into account the translational and rotational distance between configurations. This can be accomplished by adding the translational and rotational distance to the reachable volume distance (with a scaling factor, s):

$$D_{tran+rv}(c_1, c_2) = s * \text{Euclidean}(Base_{c1}, Base_{c2})) + (1-s) * \sum_{j \in J} \text{Euclidean}(j_{c1}, j_{c2})$$

where j_{c1} and j_{c2} are the position of j in c_1 and c_2 in RV-space, $Base_{c1}$ and $Base_{c2}$ are the position and orientations of the base in c_1 and c_2, and s is a scaling factor.

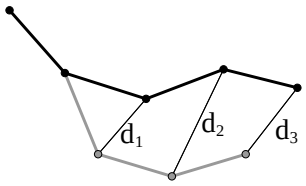


Fig. 3. The reachable volume distance between two samples (black and gray) is the sum of distances between the joints of the configurations in reachable volume space. Here the reachable volume distance is $d_1+d_2+d_3$.

V. Reachable Volume RRT (RVRRT)

We introduce an RRT expansion strategy called RV-Expand (Algorithm 5) that uses reachable volume stepping to generate RRT nodes (line 5 of Algorithm 1). This strategy takes as input a random sample, q_{ran}, and its nearest neighbor in the graph, q_{near}. It then steps one of the joints in q_{near} by a distance δ in the direction of the position of the joint in q_{ran} to form a candidate sample, q_{new}. Because RV-space encodes the relative joint positions of the robot, stepping a reachable volume sample will change the relative position of the joints and thus alter the internal coordinates of the robot. For free-base robots we also step the translational and rotational coordinates in the direction of q_{ran}.

We make no assumptions about how q_{ran} is generated (e.g., by uniform sampling as in [6] or by reachable volume sampling [8]). Here, we use reachable volume sampling [8].

RV-Expand may select a joint to perturb (line 4 of Algorithm 5) in a variety of ways:

- **Random:** select a joint at random. This is advantageous because it requires no overhead and it ensures that all joints have a chance of being selected.

Algorithm 5 RV-Expand

Function: RV-Expand(q_{ran}, q_{near}, δ, s)

Input: A cfg q_{ran}, its nearest neighbor q_{near}, a stepping parameter δ and a distance metric scaling factor s

Output: A new cfg to be added to the RRT

1: **if** free base **then**
2: δ_{tran}= $\delta * s*$ $\frac{Distance(q_{ran},q_{near})}{Distance_{tran+rv}(q_{ran},q_{near})}$
3: δ =δ - δ_{tran}
4: Select a joint j to perturb
5: q_{new} = RV-Step(q_{near},j,position of j in q_{ran}, δ)
6: **if** free base **then**
7: Step rotational and translational dofs by δ_{tran} in direction of q_{ran}
8: return q_{new}

- **Most Distant:** to select the joint that is furthest from its counterpart in the random sample, q_{ran}.
- **Probabilistic:** assign each joint a selection probability. A joint's selection probability could be proportional to the distance between it and its counterpart in q_{ran}.

VI. Experimental Results

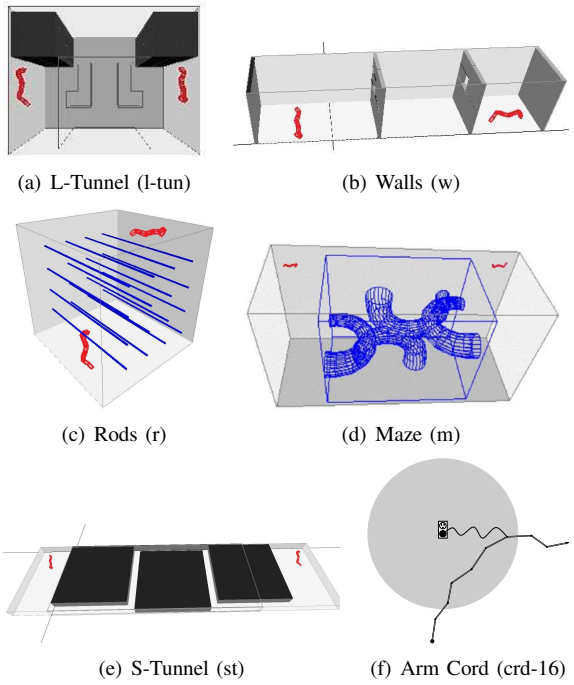
We first study how different parameter settings impact RVRRT performance in order to determine which settings work best. We then compare RVRRT performance to RRT and DDRRT in a set of constrained and unconstrained problems. We show that RVRRTs are capable of solving problems in less time and with fewer nodes than either RRTs or DDRRTs. Finally, we investigate reachable volume local planner and distance metric performance as used in PRMs.

A. Experimental Setup

We study two metrics: the **number of nodes** needed to solve the query and the **total running time** required. Smaller roadmaps are better because they are quicker to construct, require less memory, and are faster to query.

We use the following environments:

- The **L-Tunnel** (l-tun) environment (Figure 4(a)) is a commonly used benchmark that consists of 3 free regions that are connected by a pair of L-shaped tunnels. We run experiments using a 22 dof open chain. To solve this problem, a planner needs to generate highly deformed configurations that will fit in the passages.
- The **Walls** (w) environment (Figure 4(b)) is another commonly used benchmark. It is a 19x4x4 unit^3 environment consisting of 4 chambers separated by 5 walls. Both walls have 1x1 openings. We run experiments using free flying chains of varying dof (22–134), and a 70 dof closed chain. The high dof of the problem and multiple narrow passages make it difficult.
- The **Rods** (r) environment (Figure 4(c)) consists of a 70 dof open chain (r-70) or closed chain (r-cc) in an environment with 16 rods. The rods and the high dof make generating collision free samples very difficult.
- The **Maze** (m) environment (Figure 4(d)) consists of a 22 dof open (m-22) or closed chain (m-cc) which passes through a 3-dimensional maze. Mazes are notoriously difficult for RRTs to solve.
- The **S-Tunnel** (s-tun) environment (Figure 4(e)) is another commonly used benchmark which consists of 2 chambers connected by a long narrow tunnel. We run experiments using free flying chains of varying dof (22–134) and a 70 dof closed chain.
- The **Arm Cord** (crd-16) environment (Figure 4(f)) consists of a 16 dof fixed base linkage robot in a cluttered environment. The motion of the robot is constrained by a cord which is attached to one of its joints so that the distance between the joint and the base of the cord cannot exceed the length of the cord (light gray region). This scenario could be encountered when an industrial robot is operating a tool with an external power supply.



(a) L-Tunnel (l-tun) (b) Walls (w) (c) Rods (r) (d) Maze (m) (e) S-Tunnel (st) (f) Arm Cord (crd-16)

Fig. 4. Environments studied.

All computation was performed on Brazos, a major computing cluster at Texas A&M University. Processing nodes consist of quad-core Intel Xeon processors running at 2.5 Ghz with 15 GB of RAM. All experiments had a maximum allocation of 8 hours and are averaged over 10 runs.

B. RV-Expand Parameter Study

We first evaluate the range of parameter settings for RV-Expand to determine which produce good results. RV-Expand takes the following parameters:

- **Order of Reachable Volume Computation:** The possible orderings are **End Effector First**, **Root First**, and **Binary**. The effects of the ordering are discussed in Section IV-A.
- **Repositioning Policy:** This policy determines how joints that are no longer in their reachable volumes are repositioned (line 5 of Algorithm 3). We consider two

policies: select a **Random** point in the new reachable volume and select the point in the new reachable volume that is **Closest** to the original position of the joint.

- **Joint Selection Policy:** This policy determines how to select the perturbed joint (line 4 of Algorithm 5). We study two policies: select a **Random** joint and select the joint that is **Most Distant** from its q_{ran} counterpart.
- δ**:** The step size used when generating q_{new}. To facilitate comparison across different environments, δ is normalized by the environment diameter.
- **Scaling Factors:** s indicates the relative weighting of reachable volume distance and rigid body distance while s_{rot} indicates the relative weighting of the translational and rotational coordinates (see Section IV-C).

We first evaluate the different combinations of reachable volume computation order, joint selection policies, and repositioning policies, see Table I. We run each combination across a variety of δ, s and s_{rot} values and select the settings which solve a problem with the fewest number of nodes. We will study these parameters in detail next.

Method	Computation Order	Repositioning	Joint Selection
RVRRT-1	EndEffectorFirst	Closest	Closest
RVRRT-2	EndEffectorFirst	Closest	Random
RVRRT-3	EndEffectorFirst	Closest	MostDistant
RVRRT-4	EndEffectorFirst	Random	Closest
RVRRT-5	EndEffectorFirst	Random	Random
RVRRT-6	EndEffectorFirst	Random	MostDistant
RVRRT-7	RootFirst	Closest	Closest
RVRRT-8	RootFirst	Closest	Random
RVRRT-9	RootFirst	Closest	MostDistant
RVRRT-10	RootFirst	Random	Closest
RVRRT-11	RootFirst	Random	Random
RVRRT-12	RootFirst	Random	MostDistant
RVRRT-13	Binary	Closest	Closest
RVRRT-14	Binary	Closest	Random
RVRRT-15	Binary	Closest	MostDistant
RVRRT-16	Binary	Random	Closest
RVRRT-17	Binary	Random	Random
RVRRT-18	Binary	Random	MostDistant

TABLE I
RVRRT VARIATIONS FROM DIFFERENT POLICY COMBINATIONS.

Figure 5 shows the number of nodes and running time required for each combination to solve l-tunnel (l-tun), 70-dof walls (w-70), and rods. Overall, methods with root-first or binary reachable volume computation orders outperformed methods with end effector first. Methods with random joint selection tended to do better than methods with closest or most distant joint selection. Methods with random repositioning also did better than closest repositioning.

We next observe that methods 11, 12, and 14 (RVRRT-11, RVRRT-12 and RVRRT-14) gave the best results. RVRRT-11 solves all environments, requires the least running time to solve rods, and performs well in other environments. RVRRT-14 also solves all environments, gives the best performance for l-tun, and was one of the most efficient methods in walls. RVRRT-12 gives the best performance in walls and the second best performance in l-tun, although it does not solve rods.

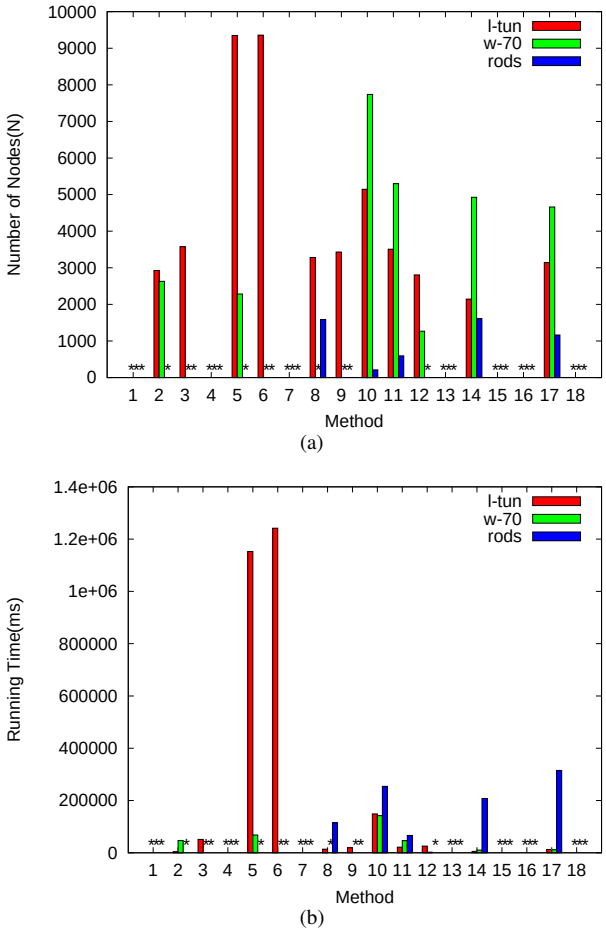


Fig. 5. (a) Number of nodes and (b) running time required for RVRRT variants (see Table I) in the l-tun, walls, and rods environments. *s indicate that a method was unable to find a solution.

We also ran experiments using δ values ranging from .001 to 10, s values ranging from .025 to .075, and s_{rot} values ranging from .025 to .975. Table II shows the best δ, s, and s_{rot} values for the selected methods in each environment. The best δ values were similar when the methods were applied to the same environment but varied greatly across environments. The best s value was consistently around .9, and the best s_{rot} value was always between .075 and .25. In our remaining experiments we use a $s = .9$ and a $s_{rot} = .1$, and tune δ to the environment.

C. RVRRT in Practice

Here we compare the RVRRT variations selected in Section VI-B to the RRT [6] and DDRRT [15, 14] methods. As in the previous section, we study the number of nodes and running time required to solve a problem. Our results (Figures 6 and 7) demonstrate that RVRRTs are capable of solving problems more efficiently then existing methods and of solving problems existing methods cannot in both unconstrained and constrained systems.

We first observe that RVRRT variations are able to solve all of the problems that RRTs and DDRRTs could solve. Furthermore, they all consistently required fewer nodes to

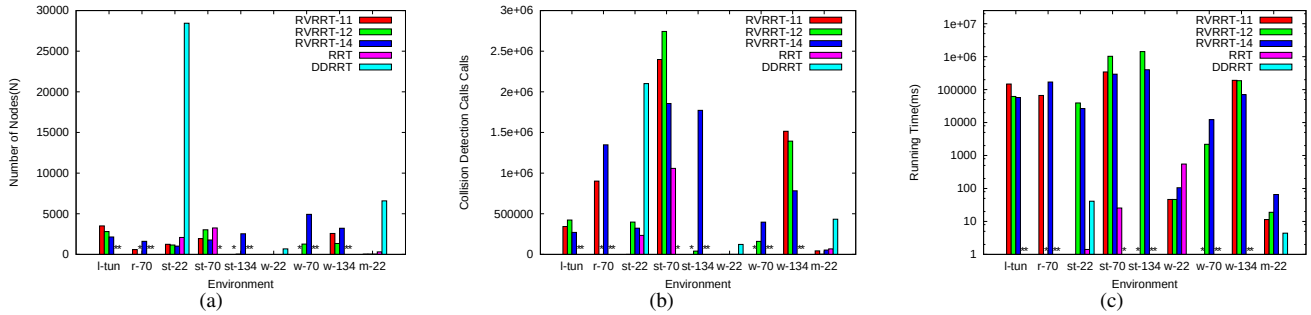


Fig. 6. (a) Number of nodes, (b) collision detection calls and (c) running time required for RRT, DDRRT and 3 RVRRT variations in environments without constraints. *s indicate that a method was not able to find a solution.

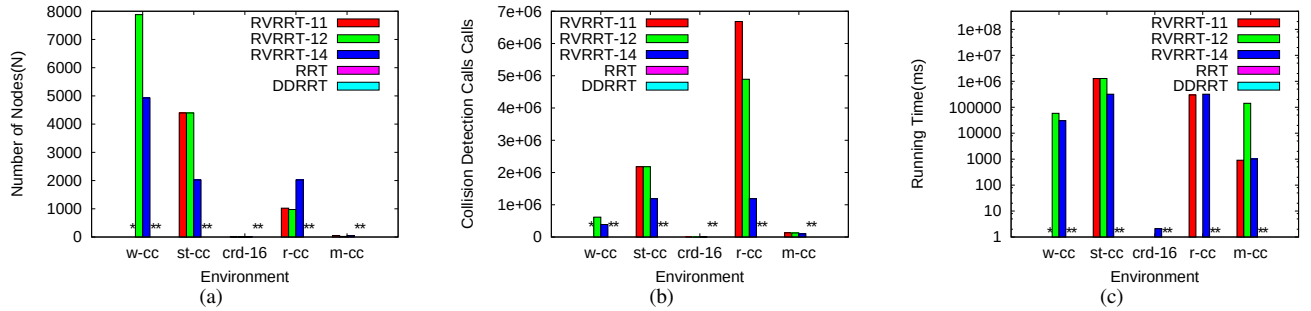


Fig. 7. (a) Number of nodes, (b) collision detection calls and (c) running time required for RRT, DDRRT and 3 RVRRT variations in environments with constraints. *s indicate that a method was not able to find a solution or could not be applied to the problem.

Method	Environment	δ	s	s_{rot}
RVRRT-11	l-tun	12.5	0.9	0.1
	walls-70	5	0.9	0.1
	rods	0.788	0.925	0.075
RVRRT-12	l-tun	20	0.9	0.2333
	walls-70	7	0.9	0.75
	rods	-	-	-
RVRRT-14	l-tun	15.875	0.9	0.2
	walls-70	6.4	0.9	0.75
	rods	0.6295	0.925	0.1625

TABLE II

BEST δ, s, AND s_{rot} VALUES FOR SELECTED METHODS.

solve these problems than RRTs or DDRRTs (Figures 6(a) and 7(a)). In some cases, such as st-22 and m-22, the RVRRT variations require substantially fewer nodes. This is significant because roadmaps with fewer nodes require less memory and are cheaper to query. We next observe that the RVRRT variations are more efficient than RRTs and DDRRTs in that they require fewer collision detection calls (Figures 6(b) and 7(b)). The running time of RVRRTs is generally higher than RRTs in low dof problems but comparable to RRTs and DDRRTs in higher dof problems (Figures 6(c) and 7(c)). Finally, we observe that RVRRTs are able to solve many difficult problems, such as the l-tun, r-70, and r-cc, that RRTs and DDRRTs are not able to. RVRRTs are also the only methods that were able to solve the high dof w-134 and st-134 environments.

There are a number of explanations for why RVRRTs outperform RRTs and DDRRTs. One reason is that reachable volume stepping tends to move the joints of the robot in the same direction in workspace. This allows RVRRTs to expand into more distant regions of the workspace quicker. Another reason is that RVRRTs tend to produce nodes where the orientation of the joints is more uniform. This trend was observed in [9] and was shown to produce samples that were less likely to be invalid due to self-collision. A third reason is that a problem's constraints are incorporated into reachable volumes, so new nodes will always conform to them. A fourth reason comes from using reachable volume sampling, which generates q_{ran} samples over the constraint satisfying subset of c-space. The growth of the RVRRT will therefore be biased towards the unexplored regions of the constraint-satisfying subset of c-space.

D. Reachable Volume Local Planner and Distance Metric in Practice

The primary advantage of the reachable volume local planner is that it generates paths that satisfy the problem's constraints while other methods such as straight line do not. We compare the reachable volume local planner and distance metric to straight line local planning and scaled Euclidean distance in the context of PRM construction. We use the walls environment as a case study and look at a 22 dof linkage (no constraints) and a 70 dof closed chain (has constraints). We attempt k-closest connection for 2000 samples with $k = 8$ using the following combinations: reachable volume local planning with scaled Euclidean distance (rv-se), reachable volume local planning with reachable volume distance (rv-rv), and straight line local planning with scaled

Euclidean distance (sl-se).

Figure 8 summarize the results. For the unconstrained problem (w-22), rv-se and sl-se produce a similar number of edges using a similar amount of time. The reachable volume distance metric (rv-rv) does not perform as well here. For the constrained problem (w-cc), only rv-se and rv-rv are applicable as straight line local planning does not enforce the closure constraints. As in w-22, rv-se requires less time and produces more edges than rv-rv.

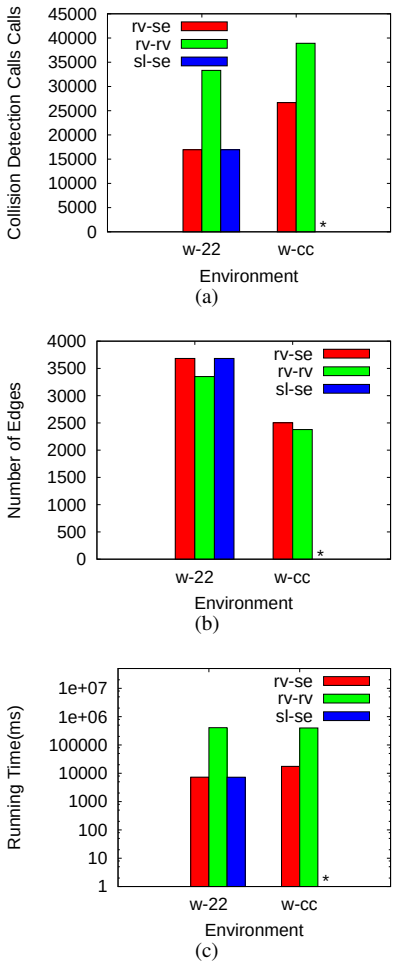


Fig. 8. (a) Collision detection calls, (b) number of edges, and (c) connection time for roadmaps constructed using reachable volume local planning with scaled Euclidean (rv-se), reachable volume local planning with reachable volume distance (rv-rv), and straight line local planning with scaled Euclidean distance (sl-se) when applicable.

VII. Conclusion

We present a *reachable volume RRT* (RVRRT) that generates constraint-satisfying solutions to high degree of freedom motion planning problems. As part of this work, we present a reachable volume stepping function, a reachable volume expand function, a reachable volume local planner, and a reachable volume distance metric. We show experimentally that RVRRTs are more efficient than RRTs and DDRRTs for high degree of freedom problems and problems with constraints. We present results for environments with a many as 134 degrees of freedom and show that RVRRTs are capable of solving problems using fewer nodes and fewer collision detection calls than RRTs or DDRRTs. We also show that they are capable of solving a series of difficult problems that neither RRTs or DDRRTs can solve.

References

[1] N. M. Amato, O. B. Bayazit, L. K. Dale, C. V. Jones, and D. Vallejo. Choosing good distance metrics and local planners for probabilistic roadmap methods. *IEEE Trans. Robot. Automat.*, 16(4):442–447, August 2000.

[2] L. Jaillet and J.M. Porta. Path planning with loop closure constraints using an atlas-based RRT. In *15th International Symposium on Robotics Research*, 2011.

[3] Marcelo Kallmann, Amaury Aubel, Tolga Abaci, and Daniel Thalmann. Planning collision-free reaching motion for interactive object manipulation and grasping. *Computer Graphics Forum*, 22(3):313–322, September 2003.

[4] L. E. Kavraki, P. Švestka, J. C. Latombe, and M. H. Overmars. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE Trans. Robot. Automat.*, 12(4):566–580, August 1996.

[5] Keith Kotay, Daniela Rus, Marsette Vona, and Craig McGray. The self-reconfiguring robotic molecule: Design and control algorithms. In Pankaj K. Agarwal, Lydia E. Kavraki, and Matthew T. Mason, editors, *Robotics: The Algorithmic Perspective*, pages 375–386. A. K. Peters, Boston, MA, 1998. book contains the proceedings of the International Workshop on the Algorithmic Foundations of Robotics (WAFR), Houston, TX, 1998.

[6] S. M. LaValle and J. J. Kuffner. Rapidly-exploring random trees: Progress and prospects. In *New Directions in Algorithmic and Computational Robotics*, pages 293–308. A. K. Peters, 2001. book contains the proceedings of the International Workshop on the Algorithmic Foundations of Robotics (WAFR), Hanover, NH, 2000.

[7] S.M. LaValle, J.H. Yakey, and L.E. Kavraki. A probabilistic roadmap approach for systems with closed kinematic chains. In *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, pages 1671–1676, Detroit, MI, 1999.

[8] T. McMahon, S. Thomas, and N. M. Amato. Sampling based motion planning with reachable volumes: Theoretical foundations. In *IEEE Int. Conf. Robot. Autom. (ICRA)*, pages 6514–6521, Hong Kong, China, May 2014.

[9] T. McMahon, S. Thomas, and N. M. Amato. Sampling based motion planning with reachable volumes: Application to manipulators and closed chain systems. In *Intelligent Robots and Systems (IROS)*, pages 3705–3712, Chicago, Il., Sept. 2014.

[10] Jean-Pierre Merlet. Still a long way to go on the road for parallel mechanisms. In *ASME Bienneal Mech. Rob. Conf.*, Montreal, Canada, 2002.

[11] A. Nguyen, L. J. Guibas, and M. Yim. Controlled module density helps reconfiguration planning. In *New Directions in Algorithmic and Computational Robotics*, pages 23–36. A. K. Peters, Boston, MA, 2001. book contains the proceedings of the International Workshop on the Algorithmic Foundations of Robotics (WAFR), Hanover, NH, 2000.

[12] Amit P. Singh, Jean-Claude Latombe, and Douglas L. Brutlag. A motion planning approach to flexible ligand binding. In *Int. Conf. on Intelligent Systems for Molecular Biology (ISMB)*, pages 252–261, 1999.

[13] Chansu Suh, Beobkyoon Kim, and Frank C. Park. The tangent bundle RRT algorithms for constrained motion planning. In *13th World Congress in Mechanism and Machine Science*, 2011.

[14] A. Yershova and S. M. LaValle. Motion planning for highly constrained spaces. In *Robot Motion and Control*, pages 297–306, 2009.

[15] A. Yershova, L. Jaillet, T. Simeon, and S. M. LaValle. Dynamic-domain RRTs: Efficient exploration by controlling the sampling domaine. In *ICRA*, pages 3867–3872, 2005.