Reachable Volume RRT
Troy McMahon, Shawna Thomas, Nancy M. Amato
Technical Report TR14-007
Parasol Lab
Dept. of Computer Science and Engineering
Texas A&M University
College Station, Texas, 77843-3112, USA
\{tmcmahon,sthomas,amato\}@cse.tamu.edu
November 1, 2014

Abstract
Reachable volumes is 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 be used to 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 that can be used by methods such as PRMs to ensure that local paths satisfy constraints.

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

In [8, 9] we presented the concept of reachable volumes which denote the region 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.

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. To our knowledge, our work is unique in that it 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. Motion planning for constrained systems is a variation of the motion planning problem in which the motion of a robot is limited by constraints. These constraints could require that the robot remain in contact with a surface or that it maintain a specific clearance. They could also require 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 Motion planning for constrained systems has applications in parallel robotics [10], grasping and manipulation [5, 11], computational biology and molecular simulations [12], and animation [3].

Reachable volumes have not been applied to Rapidly-exploring Random Tree (RRT) [6] construction. The primary difficulty in applying reachable volumes to RRTs 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 in order to guarantee that the joint positions are feasible, as well as to ensure that the sample satisfies a problem’s constraints.

In this paper, we show how to apply reachable volumes to RRTs. As part of this work, we present a novel method for stepping reachable volume samples to generate samples that are close to the original while ensuring they satisfy the problem’s constraints. We then present an RRT expansion method called RV-Expand which uses reachable volume stepping to generate samples to be added to a RRT. We also present a reachable volume distance metric and a reachable volume local planner. We propose a reachable volume RRT, RVRRT, which uses
RV-expand and the reachable volume distance metric to construct a 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 a problem’s constraints.

We show experimentally that RV-RRTs are more efficient at solving high dof problems and problems with constraints. We present results for environments with a many as 134 dof, which demonstrate that RV-RRTs 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.

We also present a reachable volume local planner which can be applied to constrained problems in combination with constrained motion planning methods (including reachable volume sampling) in order to generate paths that are guaranteed to conform to a problem’s constraints.

The main contributions of this work include:

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

## II. RELATED WORK

Rapidly-Exploring Random Trees [6], or RRTs, are among the most widely used methods for solving motion planning problems. The RRT method (Algorithm 1) first creates a tree \( T \) which contains only the start node, then iteratively adds nodes to \( T \) using an expansion method until a certain node count \( N \) has been reached. Variations of the RRT method stop when a problem has been solved, or a set of goal configurations have been connected. New nodes are created by generating a random sample, locating the nearest neighbor to the sample in \( T \), then applying an expansion method to the nearest neighbor. If the new node is valid, then it is added to \( T \) along with an edge connecting it to the nearest neighbor. RRTs have been shown to be expansive and consequently probabilistically complete.

**Algorithm 1 RRT method**

**Input:** An environment \( env \), a root configuration \( q_{root} \), the number of nodes \( N \), a step size \( \delta q \) and a distance metric \( dm \)

**Output:** Tree \( T \) containing \( N \) nodes rooted at \( q_{root} \)

1: \( T = q_{root} \)
2: \textbf{while} NumberOfNodes(\( T \)) \(< N \) \textbf{do}
3: \( q_{ran} = \text{RandomCfg()} \)
4: \( q_{near} = \text{NearestNeighbor}(q_{ran}, G, dm) \)
5: \( q_{new} = \text{Expand}(q_{near}, q_{ran}, \delta q) \)
6: \textbf{if} IsValid(\( q_{new} \)) \textbf{then}
7: \( T.\text{AddNode}(q_{new}) \)
8: \( T.\text{AddEdge}(q_{near}, q_{new}) \)
9: \textbf{return} \( T \)

RRTs have been applied to a wide variety of motion problems including difficult problems with many dof, however they have been shown to be poorly suited for problems with spatial constraints [7]. The issue is that the probability of randomly sampling a configuration that satisfies the constraints could be very small and in some cases approaches zero.

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-RRT [2] which 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. It then projects the nodes along the solution path onto the manifold so that the solutions are confined to the manifold. This method is shown to be able to solve problems with closed chains and linkages with end-effector constraints, that have as many as 14 dof.

While DDRRTs, Atlas RRT 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 RVRRT is 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. This 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 $J$. This work allows for spherical, planar and prismatic joints as well as combinations of different joint types. 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 $S=(S_1,...,S_{|J|})$, where $S_j$ is a subset of the environment in which joint $j$ must be located.

This work defines RV-space to be a space of the same dimensionality as workspace in which the origin is located at one of the joints or end effectors of the robot (referred to as the root). RV-space has no obstacles and does not take validity into consideration. The reachable volume of a joint or end-effector $j$ is the set of points in RV-space that $j$ can reach while satisfying the constraints in $S$. 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. It also defines the reachable volume of a chain to be 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 $\delta$ 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 configuration (Figures 1(d) and 1(e)).

We make the following observations about the effects of 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 the only joints that will need to be repositioned are the decedents of $j$.

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. In this case 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 in the environment.

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 (see Section III).

Based on Observation 1 we know that with the exception of the children of $j$ all of the joints must still be located in the reachable volumes of their parents. We therefore only need to check if the decedents 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, then 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 because we know from Observation 2 that its descendants will be in the reachable volume of its parents.

To reposition a joint we move it to a position that is in the reachable volume 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, we know there must exist a sample for the positioning. The reachable volume
Algorithm 2 Reachable Volume Stepping

Function: RV-Step(q, j, p_{target}, \delta)

Input: A cfg q, a joint j and a target position p_{target}, a stepping parameter \delta

Output: A cfg in which the joint j has been perturbed by \delta in the direction of p_{target}

1: let p_{init} = position of j in q
2: p_{new} = p_{init} + (p_{target} - p) * \delta
3: if p' \in RV(j.LeftParent) \cap RV(j.RightParent)
4: q_{new} = copy(q)
5: Set position of joint j to be p_{new} in q_{new}
6: Reposition(q_{new}, j.LeftChild)
7: Reposition(q_{new}, j.RightChild)
8: return q_{new}
9: 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

1: if j \in RV(j.LeftParent) \cap RV(j.RightParent)
2: return q
3: Adjust position of j in q to be within RV(j.LeftParent) \cap RV(j.RightParent)
4: if j.LeftChild \neq NULL
5: Reposition(q, j.LeftChild)
6: if j.RightChild \neq NULL
7: Reposition(q, j.RightChild)
8: return q

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. You could for example select a joint at random or base your selection on a heuristic.

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:

Fig. 1. Reachable Volume Stepping: We step one joint j by a distance of \delta 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 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).
• **Linear, end effector first:** construct the reachable volumes linearly with the end effector as the top. This limits the effects of stepping to the nodes between the perturbed node and the root.

• **Linear, root first:** construct the reachable volumes linearly with the root at the top. This limits the effects to nodes between the perturbed node 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 node in a binary partitioning.

• **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, for a grasper robot you could partition the reachable volumes so that the fingers are in separate branches of the tree so that perturbing a joint in one of the fingers will only effect nodes in that finger.

### B. Reachable Volume Local Planner

We define a reachable volume local planner that is 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 $c_2$.

Figure 2 is an example of the reachable volume local planner (with a binary reachable volume structure) 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 $c_2$. Finally, it steps the fourth joint of the linkage to its position in $c_2$ resulting in the configuration $c_2$ (2(d)). Note that we are always stepping parents in the reachable volume data structure before their children. Because reachable volume stepping only effects a node’s children, we know that stepping a node will not change the position of any nodes that have already been stepped.

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 (e.g. straight line). If the robot is free-based, we can use reachable volume sampling to generate paths between 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 presented in [1].

**Algorithm 4 Reachable Volume Local Planner**

**Input:** Cfgs $c_1$ and $c_2$, a step size $\delta$

**Output:** Boolean value indicating if a path was found

1: queue.push_back($j_{\text{root}}$)
2: while $j=$queue.pop_front() do
3:    $c' = c_1$
4:    while position of $j$ in $c' \neq$ position of $j$ in $c_2$ do
5:        $p_{\text{target}}$ = position of joint $j$ in $c_2$
6:        $c' = \text{RV-Step}(c_1, j, p_{\text{target}}, \delta)$
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

### 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 the reachable volume local planner constructs paths by

![Diagram](image-url)
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 between configurations to the reachable volume distance (with a scaling factor applied to control how much weight is given to the translational and rotational distance).

\[
D_{ran+rv}(c_1,c_2) = s*\text{Euclidean}(Base_{c_1}, Base_{c_2}) + (1-s)*\sum_{j \in J} \text{Euclidean}(j_{c_1}, j_{c_2})
\]

Where \(j_{c_1}\) and \(j_{c_2}\) are the positions of \(c_1\) and \(c_2\) in Reachable Volume space, \(Base_{c_1}\) and \(Base_{c_2}\) are the position and orientations of the base in \(c_1\) and \(c_2\), and \(s\) is a scaling factor.

V. REACHABLE VOLUME RRT (RVRRT)

In this section we introduce an RRT expansion strategy called RV-Expand (Algorithm 5) that uses reachable volume stepping to generate nodes to be added to a RRT (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 the nearest neighbor by a distance \(\delta\) 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 next discuss how the RV-Expand method selects a joint to be perturbed (line 4 of Algorithm 5).

- **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.
- **Most Distant**: to select the joint that is furthest from its counterpart in the random sample, \(q_{ran}\).
- **Probabilistic**: assign each joint a probability and select a joint using this probability. You could for example assign each joint a probability that is proportional to the distance between it and its counterpart in \(q_{ran}\).

VI. EXPERIMENTAL SETUP

We first study how different parameter settings impact the performance of the RVRRT in order to determine what settings yield the best performance. We then evaluate the performance of the RVRRT in comparison to the
Algorithm 5 RV-Expand

Function: RV-Expand(q\textsubscript{ran}, q\textsubscript{near}, \delta, s)

Input: A config q\textsubscript{ran}, its nearest neighbor q\textsubscript{near}, a stepping parameter \delta and a distance metric scaling factor s

Output: A new config to be added to the RRT

1: if free base then
2: \delta\textsubscript{tran} = \delta * \text{Distance}(q\textsubscript{ran}, q\textsubscript{near}) / \text{Distance}_{\text{tran-re}}(q\textsubscript{ran}, q\textsubscript{near})
3: \delta = \delta - \delta\textsubscript{tran}
4: Select a joint \(j\) to perturb
5: q\textsubscript{new} = RV-Step(q\textsubscript{near}, j, \text{position of } j \text{ in } q\textsubscript{ran}, \delta)
6: if free base then
7: Step rotational and translational dofs by \delta\textsubscript{tran} in direction of q\textsubscript{ran}
8: return q\textsubscript{new}

RRT and DDRRT methods in a set of constrained and unconstrained problems. We show that the RVRRT is capable of solving problems in less time and with fewer nodes than either RRT or DDRRT.

We study the following metrics:

- **Number of Nodes** is the number of nodes required to solve the query. Smaller roadmaps are better because they are quicker to construct, require less memory and are faster to query.

- **Total Running Time** is the total running time a method requires to solve a 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 bends of the narrow passages.

- The **Walls** (w) environment (Figure 4(b)) is another commonly used benchmark. It is a 19x4x4 unit\textsuperscript{3} environment consisting of 4 chambers separated by 5 walls. Both walls have 1x1 openings allowing the robot to travel between the chambers. In this environment we ran experiments using free flying chains of varying dof (22–134), and a 70 dof closed chain. The high dof of the problem an presence of multiple narrow passages make it difficult to solve.

- The **Rods** 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 of this problem make generating collision free samples very difficult.

- The **S-Tunnel** (s-tun) environment (Figure 4(d)) is another commonly used benchmark which consists of 2 chambers connected by a long narrow tunnel. We ran experiments using free flying chains of varying dof (22–134), and a 70 dof closed chain.

- The **Maze** (m-22) environment (Figure 4(e)) consists of a 22 dof open chain (m-22) or closed chain (m-cc) which pass through a 3 dimensional maze environment. Maze environments are notoriously difficult for RRTs to solve.

- The **Arm Cord** (cd-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.

- The **Wheeled Gresper** (wg-19) environment (Figure 4(g)) consists of a 19 dof wheeled robot with 2 graspers attached to it. The graspers have spherical joints and need to transport an object under a low hanging environment, thus forming a closed chain.

All computation was performed on Brazos, a major computing cluster at Texas A&M University. The processing nodes consisted of quad-core Intel Xeon processors running at 2.5 Ghz, with 15 GB of RAM. All experiments had a maximum time allocation of 80 hours and all results are averaged over 10 runs.
Fig. 4. Environments studied.

Fig. 5. (a) Number of nodes and (b) execution time required for variations of the RVRRT to solve the l-tun, walls and rods environments. *s indicate that a method was not able to find a solution.

A. RV-Expand Parameter Study

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

- **Order of Reachable Volume Computation**: The effects of the order in which we compute the reachable volumes are discussed in Section IV-A. The possible orderings are **End Effector First**, **Root First**, and **Binary**.

- **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 the joint that is perturbed is selected (line 4 of Algorithm 5). We study two policies: select a **Random** joint and select the joint that is **Most Distant** from its counterpart in \( q_{ran} \).

- **\( \delta \)**: The step size used when generating \( q_{new} \). To facilitate comparison across different environments, \( \delta \) is normalized by the diameter of the environment.
<table>
<thead>
<tr>
<th>Method</th>
<th>Computation Order</th>
<th>Repositioning</th>
<th>Joint Selection</th>
</tr>
</thead>
<tbody>
<tr>
<td>RVRRRT-1</td>
<td>EndEffectorFirst</td>
<td>Closest</td>
<td>Closest</td>
</tr>
<tr>
<td>RVRRRT-2</td>
<td>EndEffectorFirst</td>
<td>Closest</td>
<td>Random</td>
</tr>
<tr>
<td>RVRRRT-3</td>
<td>EndEffectorFirst</td>
<td>Closest</td>
<td>MostDistant</td>
</tr>
<tr>
<td>RVRRRT-4</td>
<td>EndEffectorFirst</td>
<td>Random</td>
<td>Closest</td>
</tr>
<tr>
<td>RVRRRT-5</td>
<td>EndEffectorFirst</td>
<td>Random</td>
<td>Random</td>
</tr>
<tr>
<td>RVRRRT-6</td>
<td>EndEffectorFirst</td>
<td>Random</td>
<td>MostDistant</td>
</tr>
<tr>
<td>RVRRRT-7</td>
<td>RootFirst</td>
<td>Closest</td>
<td>Closest</td>
</tr>
<tr>
<td>RVRRRT-8</td>
<td>RootFirst</td>
<td>Closest</td>
<td>Random</td>
</tr>
<tr>
<td>RVRRRT-9</td>
<td>RootFirst</td>
<td>Closest</td>
<td>MostDistant</td>
</tr>
<tr>
<td>RVRRRT-10</td>
<td>RootFirst</td>
<td>Random</td>
<td>Closest</td>
</tr>
<tr>
<td>RVRRRT-11</td>
<td>RootFirst</td>
<td>Random</td>
<td>Random</td>
</tr>
<tr>
<td>RVRRRT-12</td>
<td>RootFirst</td>
<td>Random</td>
<td>MostDistant</td>
</tr>
<tr>
<td>RVRRRT-13</td>
<td>Binary</td>
<td>Closest</td>
<td>Closest</td>
</tr>
<tr>
<td>RVRRRT-14</td>
<td>Binary</td>
<td>Closest</td>
<td>Random</td>
</tr>
<tr>
<td>RVRRRT-15</td>
<td>Binary</td>
<td>Closest</td>
<td>MostDistant</td>
</tr>
<tr>
<td>RVRRRT-16</td>
<td>Binary</td>
<td>Random</td>
<td>Closest</td>
</tr>
<tr>
<td>RVRRRT-17</td>
<td>Binary</td>
<td>Random</td>
<td>Random</td>
</tr>
<tr>
<td>RVRRRT-18</td>
<td>Binary</td>
<td>Random</td>
<td>MostDistant</td>
</tr>
</tbody>
</table>

**TABLE I**

Variations of the RVRRRT method formed by using different combinations of the policies discussed in Sections VI-A. We will denote these variations as RVRRRT-1 through RVRRRT-14.

- **Scaling Factors**: The scaling factor $S$ indicates the relative weighting of the 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 methods, joint selection policies and repositioning policies, see Table I. To do this we run each combination across a variety of $\delta$, $S$ and $S_{rot}$ values and select the settings which solve a problem with the fewest number of nodes. Figure 5 shows the number of nodes and execution time required for each combination of methods to solve the l-tunnel (l-tun), 70-dof walls (w-70) and rods environments.

Overall, methods with root-first or binary reachable volume structures outperformed methods with end effector first structures. 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 in most cases.

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

We also ran experiments using $\delta$ 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 $\delta$, $S$ and $S_{rot}$ values for the selected methods in each environment. The best $\delta$ 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 $S$ value of .9 and a $S_{rot}$ of .1, while tuning $\delta$ based on the environment.

**B. RVRRRT in Practice**

Here we compare the RVRRRT variations we selected in Section VI-A to the RRT [6] and DDRRT [15, 14] methods. As in the previous section, we study the number of nodes and computation time required to solve a problem. Our results (Figure 7) demonstrate that the RVRRRT is capable of solving problems more efficiently than existing methods and that it is capable of solving problems that other methods cannot.
<table>
<thead>
<tr>
<th>Method</th>
<th>Environment</th>
<th>δ</th>
<th>S</th>
<th>S rot</th>
</tr>
</thead>
<tbody>
<tr>
<td>RVRRT-11</td>
<td>l-tun</td>
<td>12.5</td>
<td>0.9</td>
<td>0.1</td>
</tr>
<tr>
<td></td>
<td>walls-70</td>
<td>5</td>
<td>0.9</td>
<td>0.1</td>
</tr>
<tr>
<td></td>
<td>rods</td>
<td>0.788</td>
<td>0.925</td>
<td>0.075</td>
</tr>
<tr>
<td>RVRRT-12</td>
<td>l-tun</td>
<td>20</td>
<td>0.9</td>
<td>0.2333</td>
</tr>
<tr>
<td></td>
<td>walls-70</td>
<td>7</td>
<td>0.9</td>
<td>0.75</td>
</tr>
<tr>
<td></td>
<td>rods</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>RVRRT-14</td>
<td>l-tun</td>
<td>15.875</td>
<td>0.9</td>
<td>0.2</td>
</tr>
<tr>
<td></td>
<td>walls-70</td>
<td>6.4</td>
<td>0.9</td>
<td>0.75</td>
</tr>
<tr>
<td></td>
<td>rods</td>
<td>0.6295</td>
<td>0.925</td>
<td>0.1625</td>
</tr>
</tbody>
</table>

TABLE II

BEST δ, S AND S rot VALUES FOR SELECTED METHODS IN EACH ENVIRONMENT

![Graphs showing data](image)

Fig. 6. (a) Number of nodes, (b) collision detection calls and (c) execution time required for RRT, DDRRT and 3 RVRRT variations. *s indicate that a method was not able to find a solution.

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

We present a reachable volume RRT, RVRRT, that can generate 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 and a reachable volume distance metric. We also present a reachable volume local planner that can be used to generate constraint satisfying local paths.

We show experimentally that RVRRTs are more efficient than RRTs and DDRRTs at solving high degree of freedom problems and problems with constraints. We present results for environments with a many as 134 degrees of freedom, which demonstrate that RVRRTs are capable of solving problems with 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


