Biasing Medial Axis Rapidly-Exploring Random Trees with Safe Hyperspheres

David Qin
Biasing Medial Axis Rapidly-Exploring Random Trees with Safe Hyperspheres

David Qin
Computer Science Honors Paper\textsuperscript{1}
Department of Mathematics and Computer Science
University of Richmond
May 2, 2020

\textsuperscript{1}Under the direction of Jory Denny
This paper is part of the requirements for the honors program in computer science. The signatures below, by the advisor, a departmental reader, and the departmental honors representative, demonstrate that David Qin has met all the requirements needed to receive honors in computer science.

Advisor – Jory Denny

Reader – Douglas Szajda

Honors Committee Representative – Jory Denny
ABSTRACT

Motion planning is a challenging and widely researched problem in robotics. Motion planning algorithms aim to not only find unobstructed paths, but also to construct paths with certain qualities, such as maximally avoiding obstacles to improve path safety. One such solution is a Rapidly-Exploring Random Tree (RRT) variant called Medial Axis RRT that generates the safest possible paths, but does so slowly. This paper introduces a RRT variant called Medial Axis Ball RRT (MABallRRT) that uses the concept of clearance – a robot’s distance from its nearest obstacle – to efficiently construct a roadmap with safe paths. The safety of the paths generated by MABallRRT and the efficiency of the procedure in solving example queries were experimentally analyzed and compared to the original RRT and Medial Axis RRT algorithms, demonstrating MABallRRT’s potential effectiveness as a motion planner.
ACKNOWLEDGEMENTS

First and foremost, I thank my advisor, Dr. Jory Denny, for his immense role in shaping my college educational experience. Without his guidance, my past four years would have been far less fulfilling. Under his mentorship, I became a much better researcher, teammate, and student.

Thanks to my family and friends for their unwavering support and love.

Special thanks to my classmates whom I discussed ideas and performed research with, namely Tuan Manh Le, Tracy Nguyen, and Jojo Zhou. The conversations and camaraderie were a big part of what made my research experience enjoyable.

Thank you to Dr. Douglas Szajda for his valuable suggestions and edits, which helped me significantly improve the quality of this thesis.

Special shout out to Kathy Rothert for her guidance in ACM activities and her amazing compassion and work ethic that drives the Math and Computer Science department.

Lastly, I owe many thanks to the University of Richmond’s Computer Science department, especially Drs. Prateek Bhakta and Arthur Charlesworth, who also had major influences on my journey as a Computer Science student.
# TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ABSTRACT</td>
<td>ii</td>
</tr>
<tr>
<td>ACKNOWLEDGEMENTS</td>
<td>iii</td>
</tr>
<tr>
<td>TABLE OF CONTENTS</td>
<td>iv</td>
</tr>
<tr>
<td>LIST OF FIGURES</td>
<td>v</td>
</tr>
<tr>
<td>1. INTRODUCTION</td>
<td>1</td>
</tr>
<tr>
<td>1.1 Preliminaries</td>
<td>2</td>
</tr>
<tr>
<td>1.2 Research Contribution</td>
<td>4</td>
</tr>
<tr>
<td>1.3 Outline</td>
<td>5</td>
</tr>
<tr>
<td>2. RELATED WORK</td>
<td>6</td>
</tr>
<tr>
<td>2.1 Rapidly-Exploring Random Tree Algorithm</td>
<td>6</td>
</tr>
<tr>
<td>2.2 RRT Variants</td>
<td>6</td>
</tr>
<tr>
<td>2.3 Medial Axis Motion Planning</td>
<td>9</td>
</tr>
<tr>
<td>2.4 Conserving Collision Checks</td>
<td>12</td>
</tr>
<tr>
<td>3. METHODS</td>
<td>15</td>
</tr>
<tr>
<td>4. EXPERIMENTS</td>
<td>23</td>
</tr>
<tr>
<td>4.1 Experimental Setup</td>
<td>23</td>
</tr>
<tr>
<td>4.2 Experimental Results</td>
<td>24</td>
</tr>
<tr>
<td>5. CONCLUSION</td>
<td>29</td>
</tr>
<tr>
<td>REFERENCES</td>
<td>30</td>
</tr>
</tbody>
</table>
# LIST OF FIGURES

<table>
<thead>
<tr>
<th>FIGURE</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1 The medial axis (in red) of a workspace – the set of all points equidistant to two or more obstacle boundaries.</td>
<td>4</td>
</tr>
<tr>
<td>2.1 RRT expansion adds a new configuration $q_{new}$ to the roadmap by 1) sampling $q_{rand}$, 2) finding $q_{near}$ in the existing roadmap, and 3) iteratively extending from $q_{near}$ to $q_{rand}$ with step size $\Delta q$ (the dotted line segments) until a stopping condition is reached.</td>
<td>7</td>
</tr>
<tr>
<td>2.2 The PushToMedialAxis subroutine pushes $q$ to within $\epsilon$ of the medial axis of $C_{free}$.</td>
<td>10</td>
</tr>
<tr>
<td>2.3 Extension and retraction of the expansion step of Medial Axis RRT.</td>
<td>11</td>
</tr>
<tr>
<td>2.4 Ball RRT rejecting a sample already within a “safe” volume and accepting a sample outside of all volumes.</td>
<td>13</td>
</tr>
<tr>
<td>3.1 Computing direction and magnitude for MABallRRT efficient extension based on the orthogonal projection of $\vec{v}<em>{q</em>{curr}q_{rand}}$ onto $\vec{v}<em>{q</em>{curr}w_{curr}}$.</td>
<td>18</td>
</tr>
<tr>
<td>3.2 Handling turns in MABallRRT using ExtendDirectlyToRand.</td>
<td>20</td>
</tr>
<tr>
<td>3.3 Approximating obstacle stop condition in MABallRRT using ExtendDirectlyToRand.</td>
<td>21</td>
</tr>
<tr>
<td>4.1 Environments for experimentation.</td>
<td>24</td>
</tr>
<tr>
<td>4.2 Average Time (Normalized)</td>
<td>25</td>
</tr>
<tr>
<td>4.3 Average CD Calls (Normalized)</td>
<td>26</td>
</tr>
<tr>
<td>4.4 Average of Average Path Clearance (Normalized)</td>
<td>27</td>
</tr>
</tbody>
</table>
1. INTRODUCTION

Motion planning is the problem of finding a path that navigates a robot through a complex 2D or 3D environment from a starting location to a goal location. We consider a path to be a sequence of constraint-satisfying placements of the robot. For example, consider a self-driving car navigating roads to a final destination.

Motion planning has been studied extensively in various contexts; besides robotics, motion planning algorithms have applications in fields such as bioinformatics [18], virtual reality [7], and computer-aided design [1]. Depending on these different contexts, additional constraints may be placed on the paths generated by a motion planner. For instance, it may be desirable for a robot’s path to be as far from obstacles as possible. This example implies optimizing on a robot’s and path’s clearance, i.e., its distance from obstacles.

In general, motion planning is a computationally intensive procedure [15]. Of the different motion planning methods that have been explored, sampling-based planners have been the most tractable in high dimensions [12]. These methods use random sampling to construct a graph-based approximation of an environment, called a roadmap (sampling-based planners are discussed fully in Section 1.1). Rapidly-exploring random trees (RRTs) are a class of premier sampling-based planning algorithms that, in general, randomly expand a tree-based roadmap outward from a start location towards a goal location [14]. RRTs are particularly useful for solving highly-constrained single-query problems, which are motion planning problems for which a path between a single pair of start and goal states is desired [14]. RRTs are discussed thoroughly in Section 2.1.

However, the original RRT algorithm (referred to as basic RRT) [14] is only
suitable for planning feasible (i.e., suboptimal) paths and the algorithm’s efficiency is dependent on the underlying problem context. To be specific, RRTs do not optimize path clearance, as they can expand their tree close to obstacles, and thus cannot plan safe paths.

Variants of the RRT seek to address these shortcomings of basic RRT by improving the algorithm’s efficiency and clearance [4, 17]. However, to our knowledge, there are no existing motion planners that focus on both efficiency and safety of paths. Specifically, this work combines central ideas from two such variants: 1) biasing growth of a tree to maximize the clearance of key placements of the robot, and 2) reasoning about clearance information to reduce redundant computations within the RRT algorithm.

1.1 Preliminaries

In motion planning, a robot is defined as a movable object whose position and orientation can be captured by \( n \) degrees of freedom (DOFs). Each DOF is a parameter corresponding to some axis of movement (positions, orientations, joint angles, etc.). Therefore, a robot can be represented as a unique point, or configuration, \( q = (x_1, x_2, \ldots, x_n) \), where \( x_i \) is the \( i^{\text{th}} \) DOF. The space consisting of all possible configurations is called configuration space (\( C_{\text{space}} \)) [19]. If the configurations in \( C_{\text{space}} \) have \( n \) DOFs, then the \( C_{\text{space}} \) is \( n \)-dimensional. A set of constraints may be placed on the motion of a robot, e.g., a robot might not be able to collide with obstacles in the environment or a robot may have to maintain an upright orientation. Constraint-satisfying configurations are considered feasible, or valid, while constraint-violating configurations are considered infeasible, or invalid. This naturally allows a division of \( C_{\text{space}} \) into two subsets: free space (\( C_{\text{free}} \)), the set of all valid configurations, and its complement, obstacle space (\( C_{\text{obst}} \)), which is the set of all invalid configurations.
$\partial C_{\text{obst}}$ denotes the boundary of $C_{\text{obst}}$. $C_{\text{obst}}$ can be valid or invalid depending on the context of the problem. A configuration’s clearance is its distance from its nearest configuration in $\partial C_{\text{obst}}$. Given these definitions, the formal motion planning problem is to find a contiguous sequence of configurations in $C_{\text{free}}$ (i.e., a path) between query points $q_s, q_g \in C_{\text{free}}$.

Exact and complete algorithms that solve the motion planning problem do not run in reasonable time [15]. However, determining whether a configuration satisfies the problem constraints, e.g., using a collision detection (CD) test in the robot’s physical space, is typically quite efficient. Also, CD tests can be used to compute a configuration’s clearance in a low dimensional $C_{\text{space}}$ ($n \leq 6$) [11]. For reference, a typical wheeled robot has 3-dimensional $C_{\text{space}}$ and a typical free flying rigid body robot (like an airplane) has 6 DOFs, but a robot arm manipulator may have a higher dimensional $C_{\text{space}}$.

As a result of these efficient constraint-satisfaction tests, sampling-based motion planning algorithms have had great success in overcoming the computational expense of motion planning [10], and have thus been the most tractable in high dimensions [12]. These strategies use random sampling to construct an approximate representation of $C_{\text{space}}$ in the form of a roadmap. Sampling-based motion planners are typically probabilistically complete, meaning that their probability of finding an existing solution converges to one as more points are sampled [13].

The concept of the medial axis is important to this work. The medial axis of $C_{\text{free}}$ is the set of all points equidistant to two or more obstacle boundaries (see Figure 1.1) and can be used to create paths with maximal clearance [20].
1.2 Research Contribution

This paper presents a RRT variant, named Medial Axis Ball RRT (MABallRRT), that combines key insights about clearance from two RRT variants, Medial Axis RRT and Ball RRT [4, 17]. First, we use clearance information to modify key configurations during planning towards the medial axis, similar to the approach from Medial Axis RRT. Second, we use clearance information to avoid the constraint-satisfaction tests by defining “safe” areas around configurations. MABallRRT applies the concept of “safe” volumes from Ball RRT to reduce the number of expensive checks that a configuration is in collision with an obstacle, called collision checks. The second extension of clearance is that roadmaps with high clearance can be used to generate safe paths. MABallRRT uses the idea of medial axis pushing from Medial Axis RRT to create high clearance paths. We present experimental results demonstrating an improvement in clearance while maintaining a comparable runtime when compared to RRT in most environments tested.
1.3 Outline

Chapter 2 compares RRT variants that seek to improve roadmap clearance and reduce collision checks. Chapter 3 describes the implementation of a new extension method for RRT. This method contains two main components: volume-based or direct expansion followed by configuration retraction to the medial axis. Chapter 4 explains the experimental procedure and displays experimental results. Efficiency and clearance results of RRT, Medial Axis RRT, and MABallRRT are compared. Chapter 5 provides a recap of this work and suggests avenues for future research.
2. RELATED WORK

In this section, the RRT algorithm is explained and related RRT variants are summarized.

2.1 Rapidly-Exploring Random Tree Algorithm

RRTs solve a query by iteratively growing, or expanding, a tree-based roadmap from a root configuration $q_{root}$. In each expansion attempt, a random configuration $q_{rand}$ is chosen and the nearest configuration in the current tree is determined. This nearest configuration is denoted $q_{near}$. The current tree is then extended by incrementally growing $q_{near}$ toward $q_{rand}$ in the EXPAND method (Algorithm 1, Line 5). Each iteration within EXPAND yields a new configuration, $q_{new}$, which is added to a list, $I$. Thus, $I$ represents a polygonal chain of intermediate configurations between $q_{near}$ and $q_{rand}$; this can be visualized in Figure 2.1, where each red circle represents a node in $I$. This incremental extension within EXPAND terminates when a stopping condition is reached. Afterwards, the configurations of $I$ are added to the current tree (Line 5 of Algorithm 1). These expansion attempts repeat until the query is solved – that is, a collision-free path has been found from the starting configuration to the goal configuration.

The desired performance of RRTs may depend on different problem constraints. Thus, various methods have been introduced to improve some aspect of the RRT.

2.2 RRT Variants

Obstacle-Based RRT is an RRT variant that uses the shape of obstacles to influence exploration [16]. This is accomplished by using a collection of heuristics that biases growth based on information from triangles within a polyhedral obstacle.
Algorithm 1 RRT

Input:
q\textsubscript{root}: A root configuration. ∆q: Step size.

Output: A tree T

1: T.AddNode(q\textsubscript{root})
2: while ¬ done do
3: q\textsubscript{rand} ← RANDOMCfg()
4: q\textsubscript{near} ← NEARESTNeighbor(T, q\textsubscript{rand})
5: I ← EXPAND(q\textsubscript{near}, q\textsubscript{rand}, ∆q)
6: UPDATE(T, q\textsubscript{near}, I)
7: return T

Figure 2.1: RRT expansion adds a new configuration q\textsubscript{new} to the roadmap by 1) sampling q\textsubscript{rand}, 2) finding q\textsubscript{near} in the existing roadmap, and 3) iteratively extending from q\textsubscript{near} to q\textsubscript{rand} with step size ∆q (the dotted line segments) until a stopping condition is reached.
surface; e.g., biasing based on a tangent vector to an obstacle surface. One of the proposed heuristics biases exploration towards the medial axis. Obstacle-Based RRT performs best when the heuristics are parameter-tuned specifically for an environment. Because Obstacle-Based RRT does not bias growth specifically away from obstacle boundaries, trees grow in close vicinity to obstacles. Thus, Obstacle-Based RRT is not intended for generating high-clearance paths.

RRT* is an RRT variant that ensures asymptotically optimal tree growth according to some cost function [9]. It does so by performing a few additional steps to the basic RRT’s extension. In the basic RRT algorithm, $q_{\text{new}}$ (the result of an RRT extension) is connected to $q_{\text{near}}$ ($q_{\text{new}}$’s nearest neighbor in the current tree, $T$). In RRT*, $Q_{\text{near}}$ ($q_{\text{new}}$’s nearest neighbors in $T$ within some threshold distance) are obtained and $q_{\text{new}}$ is connected to a neighbor $q \in Q_{\text{near}}$ such that the minimum accumulated cost is incurred. Afterwards, $T$ is rewired such that a new edge is added between $q_{\text{new}}$ and $q \in Q_{\text{near}}$ if $q$ can be reached with lower cost through $q_{\text{new}}$ than though $q$’s original parent. The edge from $q$ to its original parent would then be removed to maintain the tree structure. However, claims of the algorithm’s optimality assume a monotonic cost function in the sense that for any two paths $\sigma_1$ and $\sigma_2$, where each connects two arbitrary points, $\text{cost}(\sigma_1) \leq \text{cost}(\sigma_1|\sigma_2)$, where $\sigma_1|\sigma_2$ denotes the concatenation of $\sigma_2$ to $\sigma_1$. This assumption does not hold for path clearance.

Transition-Based RRT grows trees using a cost map over $C_{\text{space}}$. In this RRT variant, a filtering function is used to filter configurations that do not meet a threshold for the generation of low cost paths [6]. Transition-Based RRT with a clearance-based cost function was shown to be effective in a molecular motion problem. However, the threshold used in that filtering function may be difficult to tune for other environments.
2.3 Medial Axis Motion Planning

Generating roadmaps on the medial axis of $C_{free}$ was introduced in the Medial Axis Probabilistic Roadmap (Medial Axis PRM) planner [20]. In this algorithm, configurations are retracted to the medial axis in a subroutine that will be referred to as **PushToMedialAxis** (Algorithm 2 and Figure 2.2). Given a configuration $q$, this process requires finding its closest point in $\partial C_{obst}$, $w_q$ (e.g., by an exact computation of clearance in the workspace). $w_q$ is also called the *witness configuration* of $q$. There are two cases for $q$’s validity: 1) if $q$ is in $C_{free}$, then $q$ is translated towards the medial axis in direction $\overrightarrow{w_qq}$, or 2) if $q$ is in $C_{obst}$, $q$ is first translated to $w_q$ and then pushed perpendicularly from $w_q$ towards $C_{free}$. In either case, the translation of $q$ continues until $w_q$ changes. This translation step is depicted in Figures 2.2a – 2.2c. Then, using binary search between the last two configurations of the translation step, $q$ is pushed to within an $\epsilon$ distance of the medial axis of $C_{free}$, where $\epsilon > 0$ is arbitrarily close to 0, thereby locally maximizing clearance (see Figures 2.2d – 2.2f).

---

**Algorithm 2 PushToMedialAxis**

**Input:** Configuration $q$ and tolerance $\epsilon$.

**Output:** A node $q$ within $\epsilon$ of the medial axis.

1: $w_q \leftarrow \text{NearestContactCfg}(q)$

2: **if** $q \in C_{free}$ **then**
3: $\vec{v} \leftarrow \overrightarrow{w_qq}$
4: **else**
5: $\vec{v} \leftarrow \overrightarrow{qw_q}$
6: Translate $q$ in direction $\vec{v}$ until $q = w_q$.
7: $\vec{v} \leftarrow -\vec{v}$
8: Translate $q$ in direction $\vec{v}$ until $w_q$ changes.
9: Perform binary search between the last two points of the translation step, stopping when two consecutive points of the binary search are within $\epsilon$ of each other.
10: **return** $q$
(a) $q$’s translation direction $\overrightarrow{w_qq}$ is computed.

(b) $q$ has translated in direction $\overrightarrow{w_qq}$. Since $w_q$ has not yet changed, $q$ will again move in direction $\overrightarrow{w_qq}$.

(c) After translating $q$, $w_q$ has changed. Thus, bisection with bounds $q_{prev}$ ($q_{low}$) and $q$ ($q_{up}$) begins.

(d) (Zoomed in) $q_{mid}$ and its witness are computed. Since $q_{mid}$’s witness equals $w_{q_{up}}$, $q_{up}$ is updated to $q_{mid}$.

(e) Again, $q_{mid}$ and its witness are computed. Since $q_{mid}$’s witness equals $w_{q_{low}}$, $q_{low}$ is updated to $q_{mid}$.

(f) Bisection repeats until $\|q_{low} - q_{up}\| < \epsilon$.

Figure 2.2: The PushToMedialAxis subroutine pushes $q$ to within $\epsilon$ of the medial axis of $C_{free}$. 
Computing exact distances to obstacle boundaries is feasible in low dimensional $C_{space}$. However, for high DOF robots, approximate methods for clearance computations were shown to be successful [8].

Medial Axis RRT incorporates the PushToMedialAxis subroutine (Algorithm 2) into RRT’s expansion step (i.e., by substituting Algorithm 3 for the EXPAND() call in Line 5 of Algorithm 1). Medial Axis RRT randomly picks configurations to iteratively extend towards, the same as in RRT (Figure 2.3a). Before adding the configuration resulting from the extension to the tree, the new configuration is pushed to the medial axis of $C_{free}$ (Line 7 of Algorithm 3 and Figures 2.3b and 2.3c). However, this action of pushing every configuration to the medial axis is computationally expensive. Additionally, checking that a valid (e.g., collision-free) edge, i.e., an edge that lies entirely in $C_{free}$, exists between each consecutive configuration is computationally expensive.
Algorithm 3 MARRTExpand

Input:
\- $q_{rand}$: The sampled configuration to expand towards.
\- $q_{near}$: The configuration in the current tree nearest to $q_{rand}$.
\- $l$: Maximum expansion length $l$.
\- $\Delta q$: Step size.
\- $\epsilon$: A tolerance threshold.

Output: The polygonal chain of intermediates on the medial axis.

1: Polygonal Chain $I \leftarrow \emptyset$
2: Configurations $q_{prev} \leftarrow q_{near}, q_{new}$
3: repeat
4: $q_{prev} \leftarrow q_{new}$
5: $I \leftarrow I \cup \{q_{prev}\}$
6: $q_{new} \leftarrow \text{EXTEND}(q_{prev}, q_{rand}, \Delta q)$
7: $\text{PUSHTO MEDIAL AXIS}(q_{new}, \epsilon)$
8: $\text{dist} \leftarrow \Delta(q_{prev}, q_{new}) + \sum_{q_i, q_{i+1} \in I} \Delta(q_i, q_{i+1})$
9: until $q_{prev} \equiv q_{new} \lor \neg \text{VALID}(q_{prev}, q_{new}) \lor \text{dist} > l$
10: return $I$

2.4 Conserving Collision Checks

Collision detection can be a computationally intensive component of the motion planning process [13]. Detecting collisions is required in RRT for expanding tree edges and is used for clearance computations when computing witness nodes. Thus, techniques have been developed for reducing the time spent on collision checks.

Volume-based sampling uses clearance information to construct a “safe” volume around a configuration. This volume is a hypersphere in $C_{free}$ such that every configuration in the volume is free. This idea has been applied to create a variant of RRT called Ball RRT, which grows a tree of volumes. In Ball RRT, exploration is biased away from existing volumes, yielding an efficiency improvement over RRT if the distance to nearest obstacles are approximated rather than computed exactly [17]. However, Ball RRT does not make clearance guarantees.
In a different method, collision-checking is used to save a value $d_{\text{min}}$, the lower bound on a configuration’s minimum distance to an obstacle. The value $d_{\text{min}}$ corresponding to a checked point $p$ defines a safety certificate, which is a subset of space containing $p$ that is known to be collision-free [2]. Within this safety certificate, new samples do not have to be collision-checked. Ultimately, as certificates cover a larger proportion of the space as the road map size increases, the probability that a configuration needs to be collision-checked decreases. The certificate checking algorithm shows that collision-checking does not have to be the computational bottleneck in sampling-based motion planning. However, this algorithm does not apply safety certificates to create high-clearance paths.

Entropy-based sampling seeks to decrease the number of required collision checks by reducing the number of samples necessary for the construction of a probabilistic roadmap that solves a query. This method seeks to add configurations that maximize the expected information gain (minimize entropy) to the roadmap [3]. Samples that,
when added, connect the largest volumes of previously disjoint connected components maximize information gain. However, this motion planner does not intend to improve roadmap clearance.

Lazy edge checking reduces the number of collision checks by delaying the test for edge feasibility [5]. In this approach, a “lazy graph,” $L$, with unchecked edges is used in conjunction with the roadmap $R$. To add a sampled free configuration, $q$, to $R$, $q$ is first added to $L$. Then, the shortest path from the starting configuration to $q$ is considered in $L$; now, the edges in the potential shortest path must be collision-checked. When edges are found to be unsafe, they are removed from $L$, but if all edges in a shortest path are feasible then they are added to $R$. While significantly less time is spent on collision checks, this motion planner does not consider roadmap clearance.

Thus, previous work has introduced methods for improving the efficiency of RRTs while others have provided ways to maximize roadmap clearance. However, there has been no proposed motion planner that combines a reduction in collision checks with path safety improvement in high dimensions.
3. METHODS

In this section, we describe the MABallRRT algorithm, a new variant of the basic Medial Axis RRT algorithm that incorporates the idea of “safe” volumes around a configuration to grow a tree whose nodes are on the medial axis of $C_{\text{free}}$. MABallRRT is the same as RRT except for its subroutine for tree growth – that is, the EXPAND method from Algorithm 1 is replaced by a new extension algorithm called MABallRRTExpand. The three main steps of MABallRRTExpand are to iteratively 1) try volume-based extension, 2) push new configurations to the medial axis, and 3) repeat this process until the algorithm extends towards an obstacle or a maximum number of iterations has been reached. The following paragraphs discuss the details of this approach.

Algorithm 4 displays the procedure for MABallRRTExpand. The following notation is used:

- $q_{\text{rand}}$: Randomly sampled configuration in $C_{\text{space}}$
- $q_{\text{near}}$: Nearest neighbor of $q_{\text{rand}}$
- $q_{\text{curr}}$: Current configuration being expanded from
- $w_{\text{curr}}$: Witness to $q_{\text{curr}}$
- $q_{\text{prev}}$: Previous configuration that was extended from
- $q_{\text{new}}$: Resulting configuration of the extension
- $\vec{v}_{q_a q_b}$: Vector from configuration $q_a$ to $q_b$
- $\vec{t}_{w_{\text{curr}}}$: Vector tangent to $C_{\text{obst}}$ at $w_{\text{curr}}$ in the direction of $q_{\text{rand}}$
• $r_{\text{curr}}$: Radius of the safe hypersphere defined by $w_{\text{curr}}$; the distance from $q_{\text{curr}}$ to $w_{\text{curr}}$; the step size for extension

• $B(q)$: Clearance ball/safe hypersphere around configuration $q$ with radius $r_{\text{curr}}$

The algorithm for volume-based extension can be found in Algorithm 5, `ExtensivelyUsingVolume`. The idea with this extension method is to extend from $q_{\text{curr}}$ towards $q_{\text{rand}}$ as far as possible without having to perform any safety checks. This is done by always expanding within $B(q_{\text{curr}})$ with a step size of $r_{\text{curr}}$. The exact direction for this extension vector is found by computing the rejection vector of $\vec{v}_{q_{\text{curr}}q_{\text{rand}}}$ onto $\vec{v}_{q_{\text{curr}}w_{\text{curr}}}$, remembering that $\text{proj}_{\vec{b}}\vec{a} + \text{rej}_{\vec{b}}\vec{a} = \vec{a}$ (visualized by Figure 3.1). The magnitude of this vector is then set to the radius of the safe hypersphere and multiplying it by some fraction close to one (to ensure that the result of the extension, $q_{\text{new}}$, is within the safe hypersphere and the path from $q_{\text{curr}}$ to $q_{\text{new}}$ does not need to be collision-checked).

In some cases, while attempting to extend from $q_{\text{curr}}$ towards $q_{\text{rand}}$, `MABall-RRTExpand` would get “stuck” between two configurations. More specifically, this occurred when $q_{\text{new}}$ was computed to be closer to $q_{\text{prev}}$ than to $q_{\text{curr}}$. This would usually happen when the expansion algorithm would need to make a drastic change in direction, like a turn. An example of this scenario can be seen in Figure 3.2. If $q_{\text{rand}}$ were sampled in $C_{\text{obst}}$, the turn could have been towards an obstacle, which would stop the expansion (described in the following paragraph). However, a turn could also be towards a configuration sampled in a narrow passage. Thus, when the expansion becomes stuck between two configurations, a direct extension from the configuration at the midpoint, $q_{\text{mid}}$, of $q_{\text{curr}}$ and $q_{\text{new}}$ towards $q_{\text{rand}}$ would be performed by Algorithm 6, `ExtendDirectlyToRand`.

This algorithm has another purpose: determining whether an extension from $q_{\text{curr}}$
Algorithm 4 MABallRRTExpand

**Input:**
- $q_{\text{rand}}$: The sampled configuration to expand towards.
- $q_{\text{near}}$: The configuration in the current tree nearest to $q_{\text{rand}}$.
- $maxN$: Number of extension attempts.
- $l$: Minimum expansion length $l$.
- $\epsilon$: A tolerance threshold.

**Output:** The polygonal chain of intermediates on the medial axis.

1: Polygonal Chain $I \leftarrow \emptyset$
2: $q_{\text{curr}} \leftarrow q_{\text{near}}$
3: $q_{\text{prev}} \leftarrow \text{NULL}$
4: $n \leftarrow 0$
5: **repeat**
6:   **{Attempt volume-based extension}**
7:   $q_{\text{new}} \leftarrow \text{EXTENDEFFICIENTLYUSINGVOLUMES}(q_{\text{curr}}, q_{\text{rand}})$
8:   **{Expansion stuck; needs to turn if $q_{\text{new}}$ is closer to $q_{\text{prev}}$ than to $q_{\text{curr}}}**
9:   **if** $\|\vec{v}_{q_{\text{new}}q_{\text{prev}}}\| < \|\vec{v}_{q_{\text{new}}q_{\text{curr}}}\|$ **then**
10:      $q_{\text{mid}} \leftarrow \text{MIDPOINT}(q_{\text{curr}}, q_{\text{prev}})$
11:      $q_{\text{new}} \leftarrow \text{EXTENDDIRECTLYTORAND}(q_{\text{mid}}, q_{\text{rand}})$
12:      **if** $q_{\text{new}} = \text{NULL}$ **then**
13:         **break**
14:   **end if**
15: **end if**
16: **{Push to medial axis}**
17: $I$.Add($q_{\text{new}}$)
18: **if** $q_{\text{rand}} \in B(q_{\text{curr}})$ **then**
19:      $I$.Add($q_{\text{rand}}$)
20: **end if**
21: **break**
22: $q_{\text{prev}} \leftarrow q_{\text{curr}}$
23: $q_{\text{curr}} \leftarrow q_{\text{new}}$
24: $n \leftarrow n + 1$
25: **until** $n \geq maxN \lor \|q_{\text{new}} - q_{\text{prev}}\| < l$
26: **return** $I$
Figure 3.1: Computing direction and magnitude for MABallRRT efficient extension based on the orthogonal projection of $\vec{v}_{q_{\text{rand}}}q_{\text{curr}}$ onto $\vec{v}_{q_{\text{curr}}}w_{\text{curr}}$.

**Algorithm 5** \textsc{ExtendEfficientlyUsingVolumes}

\begin{itemize}
\item \textbf{Input:}\n\begin{itemize}
\item $q_{\text{curr}}$: The configuration from which to extend.
\item $q_{\text{rand}}$: The sampled configuration to extend towards.
\end{itemize}
\item \textbf{Output:} Resulting configuration of the extension.
\end{itemize}

1: $\hat{t}_{w_{\text{curr}}} \leftarrow \text{proj}_{\vec{v}_{q_{\text{curr}}}w_{\text{curr}}} \vec{v}_{q_{\text{curr}}}q_{\text{rand}}$
2: $q_{\text{new}} \leftarrow q_{\text{curr}} + r_{\text{curr}}(\hat{t}_{w_{\text{curr}}})$
3: \textbf{return} $q_{\text{new}}$
Algorithm 6  

**ExtendDirectlyToRand**

**Input:**
- $q_{curr}$: The configuration from which to extend.
- $q_{rand}$: The sampled configuration to extend towards.

**Output:** Resulting configuration of the extension.

1: {Check if extension is towards obstacle using Eq. 3.1}
2: $\vec{v}_{toCheck} \leftarrow \|\vec{v}_{q_{curr}q_{rand}}\| \left(\frac{\|\vec{v}_{q_{curr}w_{curr}}\|}{\|\text{proj}_{\vec{v}_{q_{curr}w_{curr}}}\vec{v}_{q_{curr}q_{rand}}\|}\right) \hat{v}_{q_{curr}q_{rand}}$
3: $q_{toCheck} \leftarrow q_{curr} + \vec{v}_{toCheck}$
4: if $\neg \text{IsValid}(q_{toCheck})$ then
5:     return NULL
6: {Otherwise extend within the safe volume towards $q_{rand}$}
7: $q_{new} \leftarrow q_{curr} + r_{curr}(\vec{v}_{q_{curr}q_{rand}})$
8: return $q_{new}$

Towards an obstacle has occurred. We sought to prevent extensions towards obstacles, believing the resulting $q_{new}$ would take longer to be pushed to the medial axis and would not improve the roadmap. Thus, an approximation scheme was developed to determine whether or not *ExtendDirectlyToRand* was attempting to extend towards an obstacle.

As displayed by Figure 3.3, $\vec{v}_{q_{curr}q_{rand}}$ and $\text{proj}_{\vec{v}_{q_{curr}w_{curr}}}\vec{v}_{q_{curr}q_{rand}}$ form the hypotenuse and leg of a right triangle. A similar right triangle can be formed with $\vec{v}_{toCheck}$ and $\vec{v}_{q_{curr}w_{curr}}$ as the hypotenuse and leg, yielding the the equivalent ratios in Equation 3.1 and solving for the unknown $\|\vec{v}_{toCheck}\|$ (line 2 of Algorithm 6). Then, $q_{curr} + \gamma \vec{v}_{toCheck}$, where $\gamma$ is some value slightly larger than one, could be collision-checked to approximate whether extending directly from $q_{curr}$ to $q_{rand}$ would result in an extension towards an obstacle. If $q_{curr} + \gamma \vec{v}_{toCheck}$ is in collision, then MA-BallRRTExpand ends. Other stopping conditions for MA-BallRRTExpand are listed below.
(a) At iteration $i$ of an expansion, volume-based extension was used from $q_{\text{curr}}$ to get $q_{\text{new}}$.

(b) At iteration $i+1$, observe that $q_{\text{curr}}$ and $q_{\text{prev}}$ have been updated to $q_{\text{new}}$ and $q_{\text{curr}}$, respectively, from iteration $i$.

(c) At iteration $i+1$, $q_{\text{new}}$ is very close to $q_{\text{prev}}$, a configuration that has been visited. This signals the algorithm to switch from volume-based extension to direct extension.

(d) At iteration $i+1$, $q_{\text{mid}}$, the midpoint between $q_{\text{new}}$ and $q_{\text{curr}}$ is computed. $q_{\text{curr}}$ is subsequently updated to be $q_{\text{mid}}$.

(e) At iteration $i+1$, we directly extend from $q_{\text{curr}}$ towards $q_{\text{rand}}$. We compute $w_{\text{curr}}$ to get the extension step size.

Figure 3.2: Handling turns in MABallRRT using `ExtendDirectlyToRand`.
(a) Corresponds to Figure 3.2c, where the expansion algorithm has been signalled to use direct extension.

(b) Corresponds to Figure 3.2d, where \( q_{mid} \) has been computed and \( q_{curr} \) is subsequently updated to be \( q_{mid} \).

(c) Corresponds to Figure 3.2e. Using similar triangles, adding \( \vec{v}_{toCheck} \) to \( q_{curr} \) would yield \( q_{new} \) in \( C_{obst} \), indicating an obstacle was hit.

Figure 3.3: Approximating obstacle stop condition in MABallRRT using \textsc{ExtendDirectlyToRand}.

\[
\frac{\|\vec{v}_{toCheck}\|}{\|\vec{v}_{q_{curr}q_{rand}}\|} = \frac{\|\vec{v}_{q_{curr}w_{curr}}\|}{\|\text{proj}_{\vec{v}_{q_{curr}w_{curr}}} \vec{v}_{q_{curr}q_{rand}}\|}
\]  \quad (3.1)

The resulting configuration \( q_{new} \) of either the volume-based or direct extension is pushed to the medial axis using the same procedure as in Algorithm 2. At the end of an iteration of the expansion, \( q_{curr} \) gets updated to the value of \( q_{new} \) and the extension steps repeat.

This iterative expansion towards \( q_{rand} \) continues until a stopping condition is reached:

1. \( q_{rand} \) is reached by the expansion, i.e. \( q_{rand} \) is within the hypersphere of \( q_{curr} \) during some iteration of the expansion,

2. the algorithm detects that an extension is attempting to grow towards an obstacle (one of the possible outcomes of \textsc{ExtendDirectlyToRand}),
3. the distance of an expansion iteration is below some threshold $l$, or

4. some max number of expansion iterations is reached.

Thus, we see that $\text{MABALLRRTEXPAND}$ incorporates safe hyperspheres to determine step sizes for both direct extension (for changing directions) and volume-based extension with the intention of reducing collision checks. Additionally, new configurations are retracted to the medial axis with the intention of improving roadmap clearance.
4. EXPERIMENTS

We experimentally analyzed MABallRRT by comparing its growth cost and average clearance to those of RRT and Medial Axis RRT in 2D and 3D environments.

4.1 Experimental Setup

RRT, Medial Axis RRT, and MABallRRT were implemented in a C++ motion planning library developed in the SpiRoL Lab at the University of Richmond. In all methods, Proximity Query Package (PQP) [11] was used for collision detection. PQP is an open source collision detection library that can not only determine whether a configuration is in collision, but can also compute minimum distances between two robots. All methods use Euclidean distance for distance computations, a brute force neighborhood-finding strategy for determining the nearest neighbor of a configuration, and $\epsilon = 0.01$ for PushToMedialAxis.

The RRT variants were analyzed experimentally in four environments: Basic 2D with Boxes (Figure 4.1a), ZigZag environment (Figure 4.1b), MazeTunnel (Figure 4.1c), and ZTunnel (Figure 4.1d). The Basic 2D and ZigZag environments provide examples that allow us to clearly visualize the methods’ effectiveness, especially with regards to different turning angles. The MazeTunnel and ZTunnel environments have more degrees of freedom and use robots with different constraints.

The metrics for comparison were: average roadmap clearance, average time to completion, and average number of collision detection (CD) calls. The total number of CD calls made during RRT execution is commonly used to measure algorithmic efficiency. Average path clearance is used to measure the quality of the paths generated and is computed by averaging the clearances of all edges in the roadmap. All experimental metrics were averaged over 30 random seeds for each combination of
RRT variant and environment.

4.2 Experimental Results

For each experiment, the RRT variants were tasked with solving an example query. In this section, we compare the quality of the roadmaps generated by the various methods.

Figure 4.2 displays results of the average time to complete a query. In the Basic 2D, ZigZag, and MazeTunnel environments, MABallRRT was clearly more time efficient than Medial Axis RRT in solving the example queries, but less efficient than RRT. However, in these three environments, MABallRRT’s average time to solve a query was within a factor of five of RRT’s average time for all environments. In general, the longer runtimes of MABallRRT and Medial Axis RRT may be attributed to the extra \texttt{PushToMedialAxis()} operations called on each configuration before adding it to the roadmap. Medial Axis RRT’s runtime difference in the Basic 2D and
ZigZag environments was attributed to the \texttt{PushToMedialAxis()} procedure. The roadmap generated by Medial Axis RRT in the Basic2D environment contained many more nodes on average than in the ZigZag environment, each of which was pushed to the medial axis. The Basic 2D environment may have yielded more roadmap nodes due to there being more free space than in the ZigZag environment.

Figure 4.2 also shows that MABallRRT had the worst time efficiency of the three algorithms in the ZTunnel environment. We conjecture that this is because the ZTunnel robot has a tighter fit in its environment. Additionally, the large step sizes per expansion iteration may have resulted in more difficulty entering narrow passages. So even when a configuration was sampled in the narrow passage, direct expansions toward it would be recorded as an obstacle being detected. Verification of this conjecture requires further experimentation using initial configurations intentionally
placed around a narrow passage.

Figure 4.3 shows that for three of the four environments, MABallRRT had the fewest collision detection calls. In the last environment, ZTunnel, MABallRRT still had around three times fewer CD calls than Medial Axis RRT. This result demonstrates the successful application of the “safe” volumes for generating edges between successive configurations. The requirement of less collision detection calls is a major reason for the faster runtime of MABallRRT versus Medial Axis RRT to complete a query.

Figure 4.4 depicts the averages of the average roadmap clearance for each method in the various environments. Medial Axis RRT, which grows an RRT that is \(\epsilon\)-close to the medial axis of \(C_{free}\), had the highest clearance values, and therefore generates the “safest” paths [4]. However, in the Basic 2D, MazeTunnel, and ZigZag
environments, the roadmaps constructed by MABallRRT had clearance values within 75% of those of Medial Axis RRT. Additionally, the clearance results from these three environments show that the paths generated by MABallRRT were safer than those of RRT.

In Basic 2D, MazeTunnel, and ZigZag, the lower clearance values of MABallRRT relative to Medial Axis RRT may be attributed to the lengths of roadmap edges. Since the longer expansion step sizes of MABallRRT yield longer edges, the solution for a query obtained by MABallRRT would have fewer configurations on the medial axis than Medial Axis RRT. As a result, the solution to the single query would have a path that strays further from the medial axis than the solution given by Medial Axis RRT.

However, in the ZTunnel environment, MABallRRT had worse average path clear-
ance values than RRT and Medial Axis RRT. Again, this may have been due to how constrained the ZTunnel robot is with its rotational **dofs** leading to difficulty entering narrow passages. The MABallRRT algorithm also generates roadmaps with longer edges. These longer edges connecting configurations near the entrance of narrow passages were likely very close to obstacles. On the other hand, Medial Axis RRT only had slightly better clearance than RRT in the ZTunnel (whereas Medial Axis RRT’s roadmaps had noticeably higher clearance than the basic RRT’s roadmaps in the other environments), which may imply that the ZTunnel is not a good environment for studying clearance. Nonetheless, a scenario in which a similarly constrained robot in an environment with sharp turns may appear in a motion planning problem; therefore, this is a weakness of the proposed method that will need to be addressed in future work.

Thus, experimental results demonstrate the ability of MABallRRT to efficiently generating high-clearance paths in less constrained environments. MABallRRT has the potential to have good efficiency and clearance performance in constrained environments with improvements in the subroutine for changing direction.
5. CONCLUSION

This research introduces MABallRRT, a variant of RRT that can often generate safer motion plans compared to basic RRT with runtime improvements over Medial Axis RRT. Specifically, the results of experiments in Basic 2D (2 dofs), ZigZag (2 dofs) and MazeTunnel (6 dofs) demonstrate that MABallRRT’s clearance was within 75% of Medial Axis RRT’s while its time to solve a query was within a factor of five of RRT. MABallRRT performed worse in terms of clearance and runtime in the ZTunnel environment (6 dofs), which could be due to the robot’s rotational constraints making it more difficult to explore narrow passages.

There are various avenues for continuation of this study. MABallRRT should be tested in more environments to determine if the presence of other constraints affects the algorithm’s effectiveness (like in ZTunnel). Different approaches should be explored for changing directions to maintain high clearance while making turns. More components of MABallRRT could be timed to experimentally determine reasons for inefficiencies. Hyperspheres could also be used for sampling, like in BallRRT, to prevent adding samples to the roadmap that do not provide additional information about the $C_{space}$. 
REFERENCES


