

ARRT/ARRT*: A Path Planning Variant that Combines Multiple Techniques to Achieve Fast Convergence to an Initial Solution

by

Al Mahir Ahmed
21315007

A thesis submitted to the Department of Mathematics and Physical Sciences
in partial fulfillment of the requirements for the degree of
B.Sc. in Applied Physics and Electronics

Department of Mathematics and Physical Sciences
Brac University
March 2026

© 2026. Brac University
All rights reserved.

Declaration

It is hereby declared that

1. The thesis submitted is my/our own original work while completing degree at Brac University.
2. The thesis does not contain material previously published or written by a third party, except where this is appropriately cited through full and accurate referencing.
3. The thesis does not contain material which has been accepted, or submitted, for any other degree or diploma at a university or other institution.
4. We have acknowledged all main sources of help.

Student's Full Name & Signature:

Al Mahir Ahmed

21315007

Approval

The thesis titled “ARRT/ARRT*: A Path Planning Variant that Combines Multiple Techniques to Achieve Fast Convergence to an Initial Solution” submitted by

1. Al Mahir Ahmed (21315007)

On Spring, 2026 has been accepted as satisfactory in partial fulfillment of the requirement for the degree of B.Sc. in Applied Physics and Electronics on March 1, 2026.

Examining Committee:

Supervisor:
(Member)

Tasmin Kamal Tulka

Senior Lecturer
Department of Mathematics and Physical Sciences
Brac University

Program Coordinator:
(Member)

Muhammad Lutfor Rahman

Assistant Professor
Department of Mathematics and Physical Sciences
Brac University

Head of Department:
(Chair)

Md. Firoze H. Haque, PhD

Associate Professor and Chairperson
Department of Mathematics and Physical Sciences
Brac University

Abstract

The path planning problem is a fundamental problem in robotics and has been studied extensively in the literature. In this thesis, we propose a novel sampling-based algorithm called ARRT (All Rapidly-exploring Random Tree) for solving the path planning problem in complex environments. ARRT combines multiple approaches from different variants of sampling based motion planners to provide an efficient path in the smallest amount of time possible. Its novelty lies in its use of grid based techniques to provide an initial estimate of the cost to reach the goal from a given configuration, and then using informed sampling to guide the search. Our claim is that ARRT has a fast convergence to a feasible path and also provides a path that is more optimal or efficient than other state of the art algorithms in terms of number of iterations. We evaluate the performance of ARRT against several state-of-the-art algorithms, including RRT, BIT*, and ABIT*, in various environments, including cluttered environments and narrow passage problems. Our results show that ARRT outperforms these algorithms in terms of both computational time and path cost, demonstrating its effectiveness in solving the path planning problem in complex environments. We also analyze the convergence properties of ARRT and show that it can be made to be asymptotically optimal, meaning that it will converge to the optimal solution as the number of iterations increases. Finally, we discuss the implications of our findings and suggest directions for future research in the field of path planning.

Keywords: Path Planning, Sampling-Based Algorithms, RRT, Asymptotic Optimality, Informed Sampling, Robotics, Complex Environments

Dedication

This is dedicated to my parents who have sacrificed very much in order to help me and my siblings gain an education. This is dedicated to the very special people in my life who have supported me through thick and thin. This is dedicated to my friends who just like my own blood and have always cheered for me along the way. This is dedicated to my team, BRACU Mongol Tori, who have been a huge factor in my journey in robotics and have been a source of inspiration for this thesis. Finally, this is dedicated to Spider-man, who inspired me to study the sciences and inspires me to be a hero everyday. Thank you all for being there for me and supporting me in this journey, I am forever grateful.

Acknowledgement

All praise is due to Allah, the Lord of all the worlds. (Quran 1:2)

I would like to thank my parents who have supported me. I would like to thank my supervisor, Tasmin Kamal Tulka, for her guidance and support throughout the course of this thesis and for her valuable insights. I would like to thank Dr. Khalilur Rhaman for his immense feedback and support throughout the creation of this thesis. I would like to acknowledge BRACU Mongol Tori for its support in my journey in robotics and its contribution to the inspiration for this thesis.

Table of Contents

Declaration	i
Approval	ii
Abstract	iii
Dedication	iv
Acknowledgment	v
Table of Contents	vi
List of Figures	viii
List of Tables	ix
Nomenclature	ix
1 Introduction	1
2 Literature Review	3
2.1 RRT	3
2.2 RRT*	4
2.3 Heuristic sampling	4
2.4 Lazy collision checking	5
2.5 Informed Sampling	6
2.6 Post-Processing	6
3 Proposed Algorithm	8
3.1 An Informed Subset	8
3.1.1 Using an A^* grid proxy to approximate L^*	10
3.1.2 Overestimation	10
3.2 The Simplicity of Goal Biasing	11
3.3 Bidirectionality	12
3.4 Optimality: Prune or Rewire	13
3.5 Final Algorithm	14
4 Simulation	18
4.1 Table 1: Per-Map Performance	19

4.2	Table 2: Overall Performance Across All Maps	20
4.3	Table 3: Asymptotic Optimality Convergence Comparisons	20
5	Discussion and Conclusion	21
	Bibliography	24

List of Figures

3.1	Comparison of the two RRT methods	12
3.2	Bidirectional tree with estimated ellipse	12
3.3	Path before and after optimization	15
3.4	Obtaining L^* estimate and path before pruning	16
3.5	Path after pruning and after optimization	16
4.1	Comparison of ARRT path before and after pruning	18
5.1	Iterations taken to first solution	21
5.2	Cost change as iterations goes up	22

List of Tables

3.1	Goal Biasing RRT on Cluttered Environment with 30 Obstacles . . .	11
4.1	Algorithm performance on narrow passage environment	19
4.2	Algorithm performance on bug trap environment	19
4.3	Algorithm performance on clutter environment with 50 obstacles . . .	20
4.4	Overall comparison of algorithms across all maps	20
4.5	Comparison of asymptotic optimality convergence between RRT*, In- formed RRT* and ARRT*	20

Chapter 1

Introduction

The motion planning problem is described as how a robot can get from point A to point B whilst avoiding obstacles in its path. This can further be divided into 2 parts, global planning and local planning [23]. Global planning involves generating a set of points or a path that will get the robot from A to B in a known or partially known environment while avoiding collisions. The local planning is responsible for following these points and avoiding any dynamic obstacles on its way. The reason for using both these planning methodologies and not any one is easy to understand when considering global planners only as global planners generate a path; however, this may be more difficult to grasp for local planners. By using a local planner only, to get from point A to B, optimal results will not always be obtained especially in complex environments which is usually the case in real world situations, for this a combination of the two is generally used [10].

In indoor environments, given a map, grid based global planners prove to be more effective due to the constrained and structured nature of the maps and their well defined boundaries. These spaces are small enough for discretized grids [3] at a reasonable resolution to be made and due to this it is computationally effective for grid based planners such as a A* or Dijkstra to find a complete path.

This is more difficult in outdoor environments. Due to their large scale and openness with uneven terrain and dynamic elements, it is much more difficult to discretize outdoor spaces and even after discretization, it would be heavily inefficient for grid based planners as they would have to process many cells at a time as well as replace due to the dynamic nature of outdoor spaces [8]. For this, the global planners usually used for outdoor path generation are sampling based planners. These algorithms generally work by randomly sampling what is called the configuration space and then building a tree from the start to the goal [11]. These algorithms end up being highly effective in high dimensional spaces as well as continuous spaces and the outdoor environment can be thought of as a nearly continuous space.

Examples of such sampling algorithms, which will further be delved into, include RRT [16], which is the base mother algorithm and from it comes different variations of it such as RRT-Connect [12], RRT* [13], Informed-RRT* [1] and more. RRT* and Informed-RRT* are optimized variations of the RRT algorithm. They are optimized such that the total cost between the starting configuration and the goal configuration

are minimized and this leads to the path becoming more smooth over time [20] compared to its unoptimized counterparts. However, in the outdoor setting, rather than smoothness, sometimes it is more preferable to have the total distance travelled be minimized and even the number of waypoints travelled be minimized to allow for simpler processing [5].

The path planning problem can be formally defined as follows, let C be the configuration space of a robot and C_{obs} be the obstacle space for that robot. Therefore, C_{free} , the free space, would be C/C_{obs} . There exists a goal configuration, x_{goal} where x_{goal} belongs to C_{free} that has to be traversed to from x_{start} , where x_{start} belongs to C_{free} . To solve this a feasible path, $\tau : [0, 1] \rightarrow C_{free}$ where at $\tau(0) = x_{start}$ and $\tau(1) = x_{goal}$, must be found if one exists. Sampling based algorithms such as RRT solve this by incrementally constructing a tree, $\mathcal{T} = (V, E)$ that starts from x_{start} and explores the C_{free} region.

In this paper, a variation of the original RRT called A-RRT (All-RRT) which combines methods from its predecessors as well as simple tweaks such as integrating a goal oriented nature to provide a path is presented. Finally, optimizations are carried out within the algorithm to provide a final path that minimizes distance as well as the number of waypoints that need to be travelled. Furthermore, direct comparisons with other sampling based algorithms will also be shown. It should be noted that for the remainder of the paper the cost of the paths found or to be found will fully be defined as the sum of the distances between each waypoint in the path.

Chapter 2

Literature Review

In the vast majority of real world robotics applications – ranging from high speed autonomous cars to disaster response systems in unstructured environments to even dynamic manipulation in the industrial setting – the primary operational constraint is time [14]. While theoretical guarantees are intellectually satisfying and are required for long horizon efficiency, the practical ‘time-to-first’ solution serves as a critical metric [14] to determine the viability of a robotic system. Especially in the context of outdoor navigation, a robot cannot wait for an asymptotically optimal path, it might immediately require a path.

Optimality is a multifaceted constraint that can often be defined as minimizing Euclidean distance, reducing time needed to get to goal or minimizing energy required or maximizing robot safety [7]. Typically and for the remainder of this paper, optimality will be referred to as the minimization of the Euclidean distance. Standard sampling based motion planners inherently produce suboptimal paths that require refining and because of this much research has been dedicated to the minimizing of path cost which gave rise to the rewiring operations introduced by Karaman and Frazolli [13]. Apart from this there also exist other methods of minimizing the path cost such as post processing [5] or usage of heuristics [22].

In this section, prior techniques that attempt to reduce time taken to first solution as well as minimize path cost, will be examined. Before doing this it is important to discuss the preceding it that gave rise to the popularity among sampling based methods.

2.1 RRT

The standard RRT algorithm, introduced by LaValle, is a single query planning algorithm found to be efficient in high dimensional search spaces compared to grid based planners such as A*. The algorithm plots a tree that is rooted at its start configuration, x_{start} , and at each iteration it samples a point x_{rand} in the C_{free} space. It then extends the tree from node nearest to the sampled point, x_{near} , by a defined step size to a new node, x_{new} which is steered towards the sampled point by an amount, sigma, the step size. This process continues where the sampling operation across the whole search space causes the tree to rapidly explore unvisited regions in

the space eventually reaching the goal configuration, x_{goal} . The primary advantage of RRT is its probabilistic completeness, i.e. if a solution exists, it will find one as the number of iterations tend to infinity.

$$\lim_{N \rightarrow \infty} P(Success) = 1 \quad (2.1)$$

RRT is not without its limitations, the paths produced by the algorithm are not optimal and are typically characterized by unnecessary detours and “jerky” motion. Uniform sampling can be inefficient due to its sampling over entire configuration spaces and hence it lacks the ability to generate useful samples. And finally, although it converges to a solution, it can be slow as it lacks a mechanism to improve its solution over time.

While RRT excels in exploration, it lacks a mechanism to converge to a solution at a fast rate especially in cluttered and narrow environments.

2.2 RRT*

RRT* by Karaman and Frazzoli built on top of RRT while addressing the problem of the quality of the path generated. It does this by adding an additional rewiring operation after a new node is added to the tree. This step considers the neighbors around the new node x_{new} and connects to the neighbor that would result in a lower cost from the root to x_{new} rather than connecting to the nearest euclidean neighbor. After this, the algorithm checks if x_{new} can provide a lower cost path to its neighbors, if so then the neighbors are “rewired” making x_{new} their new parent. The rewiring allows for continuous cost minimization from the root to every node leading to an asymptotically optimal solution which means that as the number of iterations, $N \rightarrow \infty$, the converged path will be an optimal one. This solves the “jerky” path problem of RRT by providing a smoother, more lower cost path upon convergence and well after convergence.

This was a theoretical breakthrough, practically it had the same exploration inefficiencies that RRT had, along with being more computationally slow due to the constant search for optimal parents and rewiring. The most significant drawback for this algorithm was its slow rate of convergence to an optimal solution. The algorithm focuses greatly on path quality and cost rather than initial path convergence.

2.3 Heuristic sampling

Heuristically guided RRT variants such as hRRT by Urmson and Simmons [4] employ goal biasing as a probability, i.e. there is a defined chance that at every iteration the goal point will be used as a sample. From their experiments it was seen that the quality of paths as well as the time taken to converge on a path was improved significantly with a biasing probability of only 5-10 percent.

Potentially guided Directional RRT* by Qureshi et al [17], treats obstacles as repulsive fields and the goal point as an attractive field. This idea for artificial potential fields was introduced by Khatib for the use of real time obstacle avoidance [2] for

local planning. The fields are then used to heuristically guide the sampled points away from obstacles and towards the goal. This addresses, the slow processing rate of RRT* by increasing the rate of convergence as well as improving path quality.

RRT-Connect introduced once again by LaValle [21] is a bidirectional planner, that is, it grows 2 trees allowing for faster convergence of the initial path along with a minimization of cost relative to standard RRT without the use of the rewiring operations [13]. It does this using the **Connect** heuristic, a greedy expansion strategy. The mechanism of the heuristic is such that one tree is greedily extended towards the newest point of the other tree until it hits an obstacle or reaches the point. If the point is reached, the path is found or else another random point is sampled and the other tree is now extended in the same way. While this speeds up the convergence to an initial solution, it still suffers from producing low quality paths which are jagged and irregular.

This method proved to be so effective, however, that the **Connect** heuristic is usually used in combination with other techniques. Tahir et al, proposed the use of potentially guided bidirectional RRT* to accelerate the convergence of optimal paths using artificial potential fields as well as dual trees [15], albeit both mentioned potentially guided algorithms suffer from the local minima problem often getting stuck in bug trap scenarios, the trade off to reaching a solution was greater. *Connect-RRT-SMART also attempts to address the slow initialization issue of RRT* by carrying out standard RRT-Connect and then later using SMART optimization to refine the path for the final solution [6].

2.4 Lazy collision checking

In high dimensional motion planning collision checking is the single most expensive operation, often taking up 90 percent of the planning time [18]. This is due to the geometry engine being every time the planner samples a point thereby creating a bottleneck [19]. In order to negate this, the lazy evaluation paradigm was introduced to accelerate the time to initial solution. The core idea was the optimistic nature of the algorithm where it assumed that the path between two points was always free and only after an initial path was found [9][23]. If collisions were found between edges of the planned path, it would be removed from the graph and search would be repeated until a collision free path was found. The most concerning properties of this technique was its performance in high-clutter environments. The optimistic assumption would fail and the planner would repeatedly check for collision, find one, replan. This would lead to a degradation in performance worse than the eager checking done initially.

Adaptive solutions were introduced where a prediction model would be used to estimate the probability of the region being collision free [20]. If the region was predicted to be free, the planner would do lazy evaluation and if it was predicted to be occupied, the planner would use eager sampling. This improved the performance of planners compared to pure lazy evaluation however there was the problem of choosing the right model, tuning that model and the model itself would also consume additional RAM.

2.5 Informed Sampling

A critical critique of RRT* is its slow rate of convergence to an optimal path, it samples the entire configuration space and wastes computational resources on areas that will not help in getting a better path. Informed RRT* by Gammell et al [1] was introduced to solve this by shifting the strategy from pure exploration to focused exploitation of the region. It was shown that a subset of states can improve a current solution and they belong to the prolate hyperellipsoid (n-dimensional ellipse). Within this geometric shape, the x_{start} and x_{goal} would be the focal points and the current cost of the path would be the length of the major axis. It was mathematically shown that points sampled outside this shape would yield a path with larger cost than the initial solution [1].

$$\mathcal{E}_c = \{x \in \mathcal{C} \mid ||x_{start} - x|| + ||x - x_{goal}|| \leq c_{best}\} \quad (2.2)$$

Rejection sampling was initially used whereby points sampled outside the calculated ellipse would be rejected and points sampled within the ellipse would be used. This was computationally unfeasible as it was similar to the problem posed by the obstacle querying on a much larger scale.

This gave rise to direct sampling whereby points were sampled directly on a unit ball, which was then transformed into the aligned prolate hyperellipsoid [1]. This would ensure that every point sampled would be within the ellipsoidal subset maintaining high convergence rates even at the ellipse shrinks. This was a novel innovation used in converging to an optimal path at an accelerated rate.

2.6 Post-Processing

Regardless of the planner used, in sampling based motion planning due to the stochastic nature of the path generated, the path is almost always guaranteed to be jagged. For this, post processing techniques are often used to refine paths and minimize path costs. Visibility pruning iterates over the set of waypoints [5], taking the current waypoint as an anchor and then looking ahead to further waypoints until the waypoint is out of the line of sight. Once this happens, the previous waypoint is taken and all waypoints between the anchor and this new current waypoint are pruned. The path that is later obtained is a path whose redundant points have been removed. This method exploits the triangle inequality where a straight line is always shorter than a multi-segment polygonal chain which is essentially what RRTs produce.

Another way to minimize path costs through path refinement are by smoothening out the ragged jaggy paths produced from the tree. This can be done by introducing cubic B-splines which compute curvature continuous trajectories. In the Improved RRT-Connect algorithm by Li et al, they introduced the procedure of removing unnecessary points and smoothening the path later on. The smoothening was done by employing a linear function with parabolic transitions to output smooth motion trajectories.

Trajectory optimization techniques such as Covariant Hamiltonian Optimization or

Algorithm 1 Line of Sight Prune

Require: Original path $W = \{w_1, w_2, \dots, w_n\}$ where $n > 1$

Ensure: Refined path W_{ref}

```
1:  $W_{ref} \leftarrow \{w_1\}$  ▷ Initialize refined path
2:  $curr \leftarrow 1$  ▷ Set anchor index to start
3: while  $curr < n$  do
4:   for  $target \leftarrow n$  down to  $curr + 1$  do ▷ Look backwards from end
5:     if  $\text{CHECKLOS}(w_{curr}, w_{target})$  then ▷ Check collision-free LOS
6:        $W_{ref} \leftarrow W_{ref} \cup \{w_{target}\}$  ▷ Add visible node
7:        $curr \leftarrow target$  ▷ Update anchor to current target
8:       break ▷ Stop inner loop, repeat from new anchor
9:     end if
10:  end for
11: end while
12: return  $W_{ref}$ 
```

CHOMP are used to minimize a functional using functional gradient descent [9]. The probabilistic alternative, Stochastic Trajectory Optimization or STOMP [11] aims to solve the same problem without gradient information, making it suitable for non-differentiable costs.

Chapter 3

Proposed Algorithm

From all this, it can be seen that the key limiting factor is speed of convergence whether it is that of the initial path from x_{start} to x_{goal} or the convergence of an optimal path. The proposed algorithm, All-RRT, aims to solve this problem by combining the techniques used in previous variants in different ways while also adding small tweaks that add significant impact. It should be noted moving forward that path optimality will refer to optimality of distance and number of waypoints as discussed previously rather than the smoothness of the path itself, however, smoothness will remain a metric for understanding the quality of the path in order to maintain objectivity. It should also be taken into account that for the remainder of this paper, everything will be done with respect to 2 dimensions, however, this can very easily be extended to n-dimensions. The goal for the algorithm presented is for fast convergence to an initial solution and then implementing optimization strategies to minimize the distance travelled primarily in the context of outdoor robotic systems.

3.1 An Informed Subset

As discussed previously, Informed RRT* sought to converge to an optimal solution faster than its predecessor, RRT*. It accomplished this by narrowing the amount of points sampled in the search space, thereby focusing the sample space. This was done by creating an ellipse with the start and goal acting as foci points, the current cost of the path, c_{best} , as the length of the major axis of the ellipse and the length of the minor axis, c_{min} is defined as the distance from the start and goal which is a fixed non-changing value. From there, the points were then sampled from within the formed ellipse. This idea came from the fact that there exists a point x in C_{free} for which the optimal path from x_{start} to x_{goal} is constrained to pass through x , such that the net cost, $f(x)$ is equal to the sum of the costs of x_{start} to x , $g(x)$, and x to x_{goal} , $h(x)$. This paper will use the idea of the informed subset, presented by Gammell et al, to converge to an initial solution at a fast rate. One of the issues with using this idea is determining the length of the major axis. This section will first show why it is possible and then present a method to find this length.

Lemma 1: for any two costs c_1 and c_2 such that $c_{min} < c_1 < c_2$, the sets are nested such that $\mathcal{E}_{c_1}$ is a subset of $\mathcal{E}_{c_2}$.

Proof: This comes from the definition of an ellipse. Given the same center and foci points if a point x is within an ellipse of major axis c_1 then it is automatically within c_2 as $c_1 < c_2$. Therefore, the Lebesgue measure of free space inside the ellipse, $\mu(\mathcal{E}_c \cap X_{free})$, is a non-decreasing function of c .

We, therefore, propose that if there is some optimal collision free feasible path with cost, L^* , between x_{start} and x_{goal} , then the probability of finding a feasible path with the length of the major axis, c , being less than L^* is 0, i.e. it is not possible to find a feasible path by sampling points within the ellipse created by a major axis of length smaller than L^* . This will be true even if the number of samples N goes to infinity.

$$\forall c < L^* \lim_{N \rightarrow \infty} P(\text{Success} \mid N, c) = 0 \quad (3.1)$$

This implies then that if c is greater than or equal to L^* and N tends to infinity (probabilistic completeness guarantee), then the probability that there will be a feasible path, provided that one exists, is 1. However, for both practical and theoretical cases, any path for a robot requires clearance to sample effectively such that a non-zero volume element can be formed around the path. Let the clearance be σ_Δ which will create a tube of free space. Let this free space be called τ_{path} .

$$\lim_{N \rightarrow \infty} P(\text{Success} \mid N, c \geq L^*) = 1 \quad (3.2)$$

Theorem 1: The solution probability monotonically increases with increase in length of major axis, c , i.e. as the ellipse grows, a feasible solution is more probable. Let μ_{valid} be defined as the volume of a valid solution tube captured by the ellipse with major axis c .

$$\mu_{valid} = \mu(\mathcal{E}_c \cap \tau_{path}) \quad (3.3)$$

Proof: From lemma 1, it is known that $\mathcal{E}_c$ subset of $\mathcal{E}_{c+\Delta}$. Therefore the volume of a valid solution captured by that ellipse is smaller or equal to the volume captured by an ellipse whose major axis is incremented by a small value Δ .

$$\mu(\mathcal{E}_c \cap \tau_{path}) \leq \mu(\mathcal{E}_{c+\Delta} \cap \tau_{path}) \quad (3.4)$$

Thus as the c increases, the volume captured increases and free space connectivity also increases. If multiple distinct paths exist within this volume then it implies that a larger c encompasses all possible paths and a smaller c may encompass less. This is true for if c is equal to the minimum value, L^* for which the captured volume will encompass only one solution path. Hence, for a sufficiently large number of samples, N :

$$P(\text{Success} \mid N, c + \Delta) \geq P(\text{Success} \mid N, c) \quad (3.5)$$

Theorem 2: As the sample space area increases, hence, as the length of the major axis of the ellipse, c , increases, there is a monotonic increase in the number of distinct paths.

Proof: We will show that a path contained in one ellipse will also be the path contained in another ellipse whose major axis is greater. A set, $\Pi(c)$, is defined as the set of all feasible paths with a cost of $L(\sigma) \leq c$ where σ is a path.

$$\Pi(c) = \{\sigma : [0, 1] \rightarrow X_{free} \mid \sigma(0) = x_{start}, \sigma(1) = x_{goal}, L(\sigma) \leq c\} \quad (3.6)$$

Let $c_1 < c_2$. From lemma 1, it is known that $\mathcal{E}_{c_1}$ subset $\mathcal{E}_{c_2}$. If σ_1 is any feasible path in $\Pi(c_1)$ then by definition it will belong to X_{free} and it will have a cost of $L(\sigma_1) \leq c_1$. Since, $c_1 < c_2$, it follows that $L(\sigma_1) \leq c_1 < c_2$ which implies that $L(c_1) < c_2$. Thus any path that satisfies the cost and feasibility constraints in c_1 will also satisfy the same constraints for c_2 . Therefore:

$$\sigma \in \Pi(c_1) \implies \sigma \in \Pi(c_2) \quad (3.7)$$

This proves that the number of feasible solutions monotonically increases with increase in c .

This can be thought of very easily when considering that a path that resides in a certain area also resides in the entire earth and so on. The importance of this conclusion then begs the question that can the optimal cost, L^* , be found such that an optimal feasible solution, σ^* , can be sampled and obtained without starting first from an imperfect cost, c' , and then optimizing to the perfect cost, L^* . This paper presents a way of estimating this cost, L^* rather than directly obtaining it as it may be heavily computationally inefficient.

3.1.1 Using an A^* grid proxy to approximate L^*

Finding the estimated L^* cost is imperative as it leads to finding the ellipse required to be sampled upon to obtain the feasible path. We propose to find this estimated L^* cost by first overlaying a low resolution proxy grid on the search space such that any complex geometry of obstacles can be ignored. The A^* algorithm will then be run on this grid to obtain some path with cost, L_{grid} , and since this is being run on a very small grid, it will take only milliseconds to process. The cost, L_{grid} will now be the length of the major axis, c , for which an ellipse will be formed for onto which points will be sampled on.

3.1.2 Overestimation

Due to the nature of A^* and grid movement being strictly restricted in 90 degree or 45 degree directions, the length of L_{grid} will almost always be longer than L^* . This makes the ellipse start as larger than the theoretical minimum. This is not only optimal but necessary, because underestimating the L^* value will result in a c value smaller than L^* which will in turn result in 0 probability of a feasible solution as shown previously. A value of c perfectly equal to L^* will result in exactly 1 feasible solution which may be very difficult for a random sampling algorithm such as RRT to figure out. Therefore, with a value of c larger but still close to L^* , the algorithm allows for possibly sampling the perfect solution but also allows for the emergence

of other distinct solutions with similar costs while also not sampling unnecessary points.

3.2 The Simplicity of Goal Biasing

Traditional implementations of vanilla RRT include some sort of goal biasing whereby there is a certain probability that the sampled point will be sampled towards the goal. From experimentation on simulation compared to RRT, it was seen that directly sampling the goal rather than uniform random sampling across the search space with a small probability of goal sampling, was more effective. Random sampling would only be toggled if the direct goal sample would create a node on an obstacle, meaning if x_{new} was in the obstacle space, C_{obs} , the algorithm would default to uniform random sampling in an attempt to avoid the obstacle. By implementing this feature, it provides the direct mechanism that was mentioned previously to allow for the generations of more useful sample while also converging to a lower cost solution. Adding this feature does not affect the probabilistic completeness nature of RRT, rather it drastically improves the speed of convergence to a solution.

Algorithm	Success Rate	Avg Iterations	Avg Time (s)	Avg Path Length
RRT	100.0%	183.9	0.027	16.30
Toggle				
hRRT (goal bias = 5%)	100.0%	215.2	0.038	17.95
hRRT (goal bias = 15%)	100.0%	228.4	0.037	18.35
hRRT (goal bias = 10%)	100.0%	238.9	0.040	18.55

Table 3.1: Goal Biasing RRT on Cluttered Environment with 30 Obstacles

The above table shows data obtained from simulations done to compare the toggling implementation of RRT with hRRT at different goal bias probabilities. The data was obtained after running the simulation 100 times in an environment cluttered with 30 obstacles. At each iteration, the obstacles were randomized. The maximum number of iterations for each planner was set to 3000 to simulate the probabilistic completeness condition. It can be observed that the toggle implementation was more effective in terms of obtaining an initial solution than the other algorithms. It can also be observed that the average path length was also much lower, this is due to the algorithm being biased towards the goal, thereby being prevented from travelling to unnecessary locations.

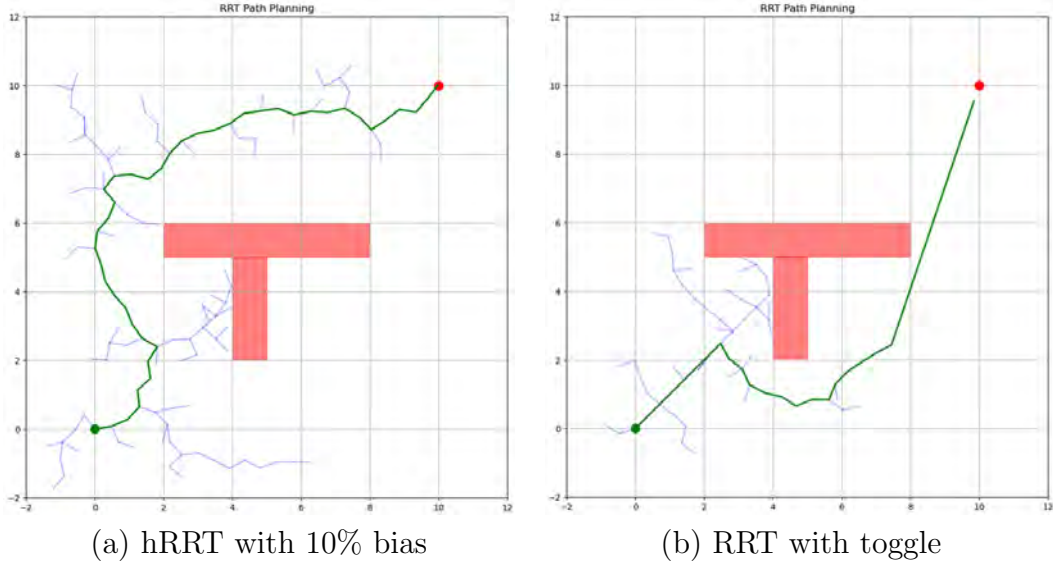


Figure 3.1: Comparison of the two RRT methods

3.3 Bidirectionality

Recall that RRT-Connect would essentially launch two RRT graphs from the start and the node, one tree would attempt to extend itself towards the latest node on the other tree and join together with that tree until it connected or until it hit an obstacle. If an obstacle was encountered, the trees would switch roles and now the other tree would attempt to extend itself to the latest node of the initial tree. This process would continue until both trees joined each other.

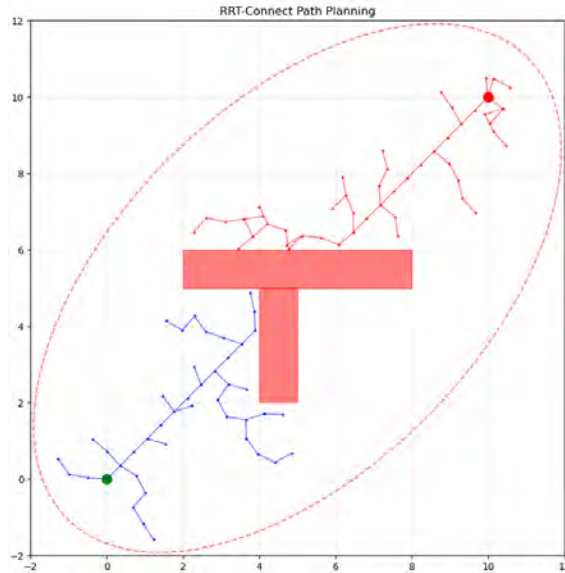


Figure 3.2: Bidirectional tree with estimated ellipse

For the ARRT algorithm, due to the goal oriented nature of the algorithm, bidirectionality is implemented quite differently. A primary and secondary tree are defined at the start of the algorithm. The secondary tree samples the leaf nodes of the primary tree from the newest node of the secondary tree. The primary tree is

set to sample the goal, however, if any of the leaf nodes of the secondary tree are within a certain distance, defined as `minimum connection distance`, then the leaf node closest to the newest node of the primary tree is sampled. Both these trees are allowed to grow for an equal number of iterations, defined by a parameter called `sub_iterations`. However, if either of the trees are within the `minimum connection distance` threshold of their sampled point, then the trees do not switch roles, the current tree continues to greedily sample the chosen point.

Algorithm 2 InformedDualTree

```

1:  $T_p \leftarrow \text{INITTREE}$ ;
2:  $T_s \leftarrow \text{INITTREE}$ 
3:  $\text{INSERTNODE}(T_p, x_{start})$ 
4:  $\text{INSERTNODE}(T_p, x_{goal})$ 
5:  $L_{T_p} \leftarrow \text{INITLIST}$ ;
6:  $l_{T_s} \leftarrow \text{INITLIST}$ 
7:  $T_p.\text{isActive} \leftarrow \text{True}$ 
8:  $n \leftarrow 0$ 
9: while  $n < N_{max}$  do
10:   if  $T_p.\text{isActive}$  then
11:      $T \leftarrow \text{GROWTREE}(x_{goal}, x_{prev\_a}, T_p, T_s, L_{T_p}, L_{T_s})$ 
12:   else
13:      $T \leftarrow \text{GROWTREE}(x_{new\_a}, x_{prev\_b}, T_s, T_p, L_{T_s}, L_{T_p})$ 
14:   end if
15:    $n \leftarrow n + 1$ 
16:   if  $T$  is found then return  $T$ 
17:   end if
18: end while

```

Algorithm 3 GrowTree

```

1:  $i \leftarrow \text{sub\_iterations}$ 
2:  $d \leftarrow \text{min\_conn\_dist}$ 
3:  $\sigma \leftarrow \text{step\_size}$ 
4:  $x_{sample} \leftarrow \text{CHOOSESAMPLINGNODE}(L_2, x_{target})$ 
5:  $x_{new} \leftarrow \text{GETNEWPOINT}(x_{prev}, x_{sample}, T_1)$ 
6:  $x_{prev} \leftarrow x_{new}$ 
7:  $\text{INSERTNODE}(T_1, x_{new})$ 
8: if  $(n \bmod i) == 0$  and  $(\text{Dist}(x_{new}, x_{sample}) > d)$  then
9:    $T_1.\text{isActive} \leftarrow \text{False}$ 
10:   $L_1 \leftarrow \text{GETLEAFNODES}(T_1)$ 
11: end if
12: if  $\text{Distance}(x_{new}, x_{sample}) < \sigma$  then
13:    $T \leftarrow \text{BUILDTREE}(T_1, T_2)$ 
14:   return  $T$ 
15: end if

```

3.4 Optimality: Prune or Rewire

In the context of outdoor robotics or exploratory missions it is often more desirable to travel shorter distances. The paths made from the initial solutions of most RRT variations are often very jagged and uneven for which the collective sum of the distance between each waypoint becomes very large. Oftentimes the number of waypoints can be reduced for these initial solutions as some travel to unnecessary

Algorithm 4 ChooseSamplingNode

```
1:  $x_{sample} \leftarrow x_{target}$ 
2: for  $leaf \in L$  do
3:   if  $\text{Dist}(leaf, x_{prev}) < \text{min\_conn\_dist}$  then
4:      $x_{sample} \leftarrow leaf$ 
5:   end if
6: end for
7: return  $x_{sample}$ 
```

Algorithm 5 GetNewPoint

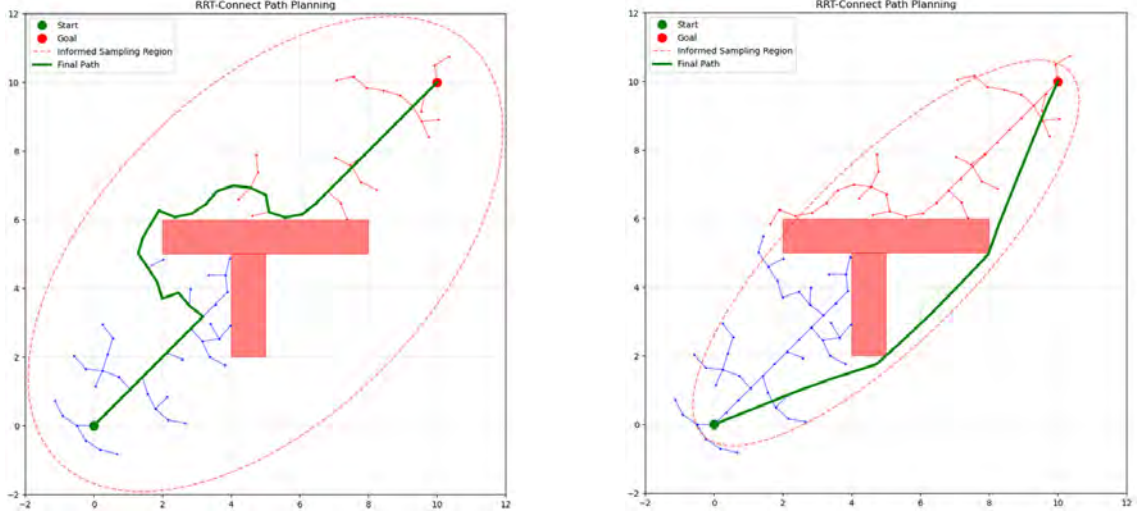
```
1:  $x_{near} \leftarrow \text{NEARESTNEIGHBOR}(x_{sample}, T_{half})$ 
2:  $x_{new} \leftarrow \text{STEER}(x_{near}, x_{sample})$ 
3: if  $\text{ISCOLLIDING}(x_{prev}, x_{sample})$  then
4:    $x_{sample} \leftarrow \text{INFORMEDSAMPLING}(x_{start}, x_{goal}, C_{est})$ 
5:    $x_{near} \leftarrow \text{NEARESTNEIGHBOR}(x_{sample}, T_{half})$ 
6:    $x_{new} \leftarrow \text{STEER}(x_{near}, x_{sample})$ 
7: end if
8: return  $x_{new}$ 
```

locations and by reducing the number of unnecessary waypoints ends up reducing the collective distance by large amounts. This pruning of waypoints operation is carried out by considering the current waypoint and check each point in the path to see if a line can be drawn without colliding with an obstacle, if it can, then the next point is considered and if it cannot then the previous points are pruned and the point that cannot be pruned is now considered as the current waypoint. This continues until the goal point is reached. Essentially it is a minimization of waypoints and hence a minimization of distance, this optimization is known as line of sight pruning. The pruning operation is always carried out after an initial solution is found to reduce the distance travelled and thereby reduce the cost of the path.

The rewiring operation introduced by Karaman and Frazolli also helps in optimizing the path cost while also making the path smoother, allowing for a robot to take less sharp turns. From experimentation, it was seen that while effective in smoothening the path, the operation did not improve the path found in terms of efficiency or quality for initial convergence. Rather it increased the time taken to converge to an initial solution and provided similar quality of path as without the rewiring operations (i.e. normal RRT) for initial solutions. It was however found to be effective to run the rewiring operations after carrying out the pruning operation on the initial solution which led to a smoother, more lower cost path.

3.5 Final Algorithm

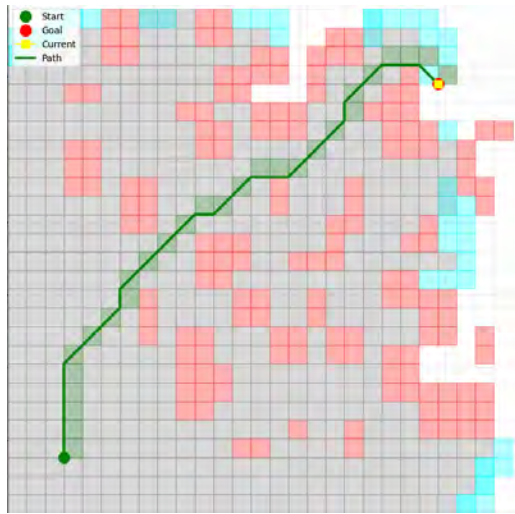
The overall algorithm can be described as a dual tree growing from the start and the goal. The primary tree (from the start) targets the goal but will also target leaf nodes of the secondary tree if they are sufficiently close. The secondary tree only targets the latest nodes of the primary tree or the leaf nodes that are sufficiently close to the tree. Both trees attempt to do this greedy sampling (i.e. goal biasing), however, upon collision with an obstacle they toggle random sampling. Each tree grows for a fixed number of iterations before allowing the other tree to grow. Random sampling takes



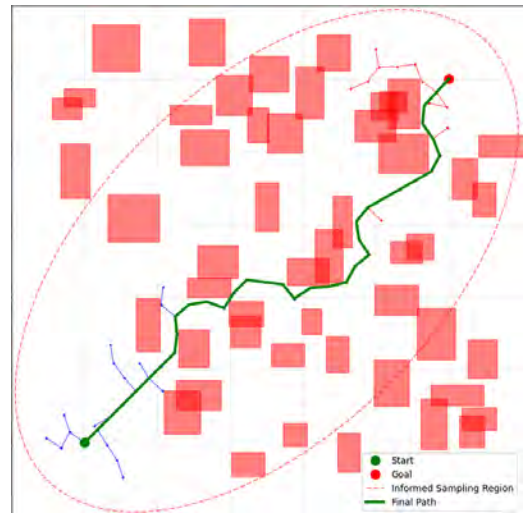
(a) Path found by ARRT, no post processing (b) Path optimization for 1700 iterations

Figure 3.3: Path before and after optimization

place on an informed subset rather than the whole search space thereby narrowing the sampling space, the informed subset is defined as an ellipse in 2 dimensions. To define the ellipse, the start and goal are taken as focus points and the distance between them is taken to be the minor axis. The major axis is found by carrying out an A^* search on a coarse grid proxy and taking the cost of the path found from the search. This value acts a rough estimate from the actual perfect L^* cost. The rough estimate allows for convergence of a path with a possible cost of L^* but also allows for the convergence of other distinct paths with similar costs. After the path is found, the unnecessary waypoints are pruned to obtain a path which travels a much smaller distance and finally if further optimization is allows, more points are sampled and a rewiring operation is carried out to lead to an even smoother and more low cost path, this also leads the final path to being asymptotically optimal. It should be noted that after pruning, if further optimization is allowed, a new tree is formed from the pruned path where new branches are grown from.

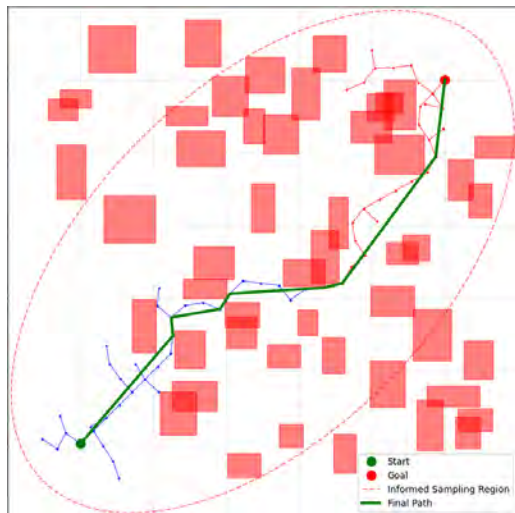


(a) A* search for initial L^* cost

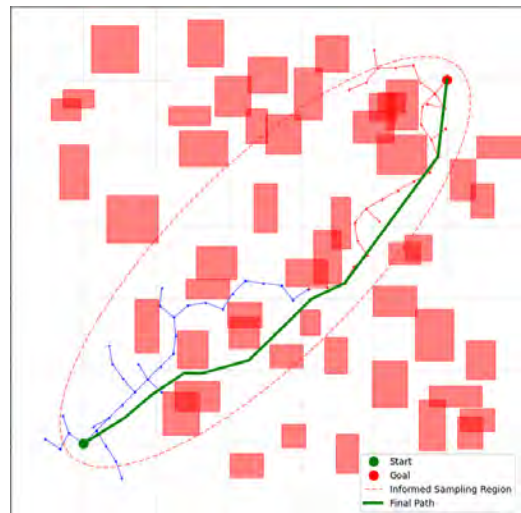


(b) Path from ARRT before pruning (cost = 17.54)

Figure 3.4: Obtaining L^* estimate and path before pruning



(a) Pruned Path (cost = 15.82)



(b) Path after RRT* optimization (cost = 15.08)

Figure 3.5: Path after pruning and after optimization

Algorithm 6 RunARRT

```
1:  $C_{est} \leftarrow \text{Distance}(x_{start}, x_{goal}) \times 1.2$ 
2:  $res \leftarrow \min(1.0, \text{SmallestWidth}(\text{obstacles}))$ 
3: success,  $C_{astar} \leftarrow \text{RUNASTAR}(x_{start}, x_{goal}, res)$ 
4: if success then
5:    $C_{est} \leftarrow C_{astar}$ 
6: end if
7:  $T \leftarrow \text{INFORMEDDUALTREE}(x_{start}, x_{goal}, C_{est})$ 
8:  $\sigma_{path} \leftarrow \text{BUILDPATH}(T)$ 
9:  $\sigma_{path} \leftarrow \text{PRUNE}(\sigma_{path})$ 
10: if optimize is True then
11:    $T \leftarrow \text{BUILDTREE}(\sigma_{path})$ 
12:   for  $k \leftarrow 1$  to  $K$  do
13:      $x_{new} \leftarrow \text{getNewPoint}(x_{start}, x_{goal})$ 
14:      $x_{parent} \leftarrow \text{chooseParent}(x_{new}, T)$ 
15:      $T \leftarrow \text{InsertNode}(x_{new}, x_{parent})$ 
16:      $T \leftarrow \text{Rewire}(T, x_{new}, x_{parent})$ 
17:   end for
18: end if
```

Chapter 4

Simulation

The ARRT algorithm was tested against state of the art algorithms such as Batch Informed Trees* (BIT) and *Advanced Batch Informed Trees* (ABIT), as well as *RRT*. Since ABIT and BIT* are asymptotically optimal planners, it was set such that the algorithms would end as soon as the first solution was found. This would allow the data to be taken based on initial path convergence which is what we claim ARRT to excel at. Both BIT* and ABIT* are equipped with KDTrees to allow for faster, more efficient search of nearest neighbors while constructing the trees. Aside from this, no additional computational efficiency optimizations were given to ARRT or RRT. Furthermore, all algorithms were run on the same hardware, with all software written in Python (Numpy) in similar styles and common random seed for random environments. ARRT and RRT both have a step size of 0.5 while BIT* and ABIT* have a batch sample size of 100 points. All of these together, allows for comparison of algorithms in terms of relative computational time.

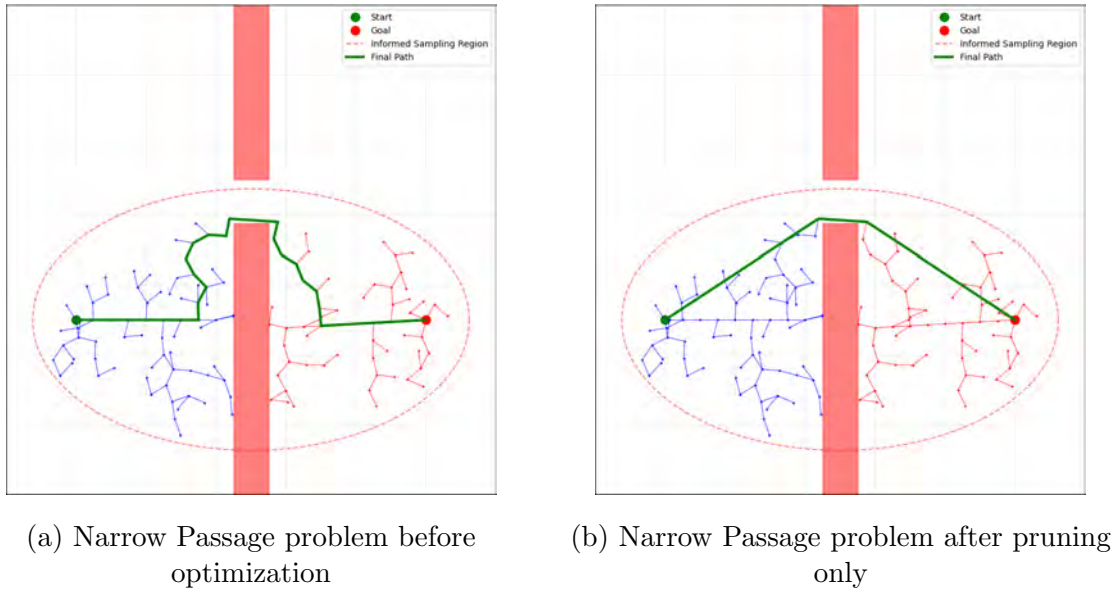


Figure 4.1: Comparison of ARRT path before and after pruning

All algorithms were run on 3 different environments and each algorithm was run on each environment 100 times. The environments are as follows: bug trap, clutter and narrow passage. Bug trap allows for assessment on an algorithms ability to

handle local minima, if it is able to get past the trap and reach the goal or does it get stuck in a loop, i.e. the local minima. In our testing, the clutter environment is set up such that there will be 50 obstacles randomly placed over the space. These obstacles will be able to overlap each other. Since it will be run 100 times, no single clutter environment is the same as its previous. This allows for testing of raw efficiency, scalability and how optimal the path is upon initial convergence. For sampling based algorithms, the narrow passage problem allows for testing how well the sampling strategy is in terms of precision and resolution.

In the table presented below, the mean time represents the average time taken from all 100 simulation runs of the environment to find the first solution from start to goal. The mean initial path cost represents the average path cost (L^2 norm) from all 100 simulation runs of the environment of the path generated from the initial solution from start to goal.

4.1 Table 1: Per-Map Performance

Narrow Passage

Algorithm	Success Rate	Avg Iterations ($\pm$ SD)	Avg Path Cost ($\pm$ SD)	Avg Time
RRT	100.0%	379.4 $\pm$ 279.4	16.650 $\pm$ 1.721	0.027s
BIT*	100.0%	8752.6 $\pm$ 9127.5	17.514 $\pm$ 3.348	0.120s
ABIT*	100.0%	3983.0 $\pm$ 5321.3	15.042 $\pm$ 1.926	0.052s
ARRT	100.0%	492.7 $\pm$ 195.0	12.125 $\pm$ 0.194	0.026s

Table 4.1: Algorithm performance on narrow passage environment

Bug Trap

Algorithm	Success Rate	Avg Iterations ($\pm$ SD)	Avg Path Cost ($\pm$ SD)	Avg Time
RRT	100.0%	310.1 $\pm$ 224.5	10.000 $\pm$ 1.219	0.019s
BIT*	100.0%	1641.3 $\pm$ 1634.3	11.074 $\pm$ 4.168	0.029s
ABIT*	100.0%	1959.1 $\pm$ 2483.3	9.035 $\pm$ 1.447	0.031s
ARRT	100.0%	117.3 $\pm$ 43.0	7.032 $\pm$ 0.691	0.006s

Table 4.2: Algorithm performance on bug trap environment

Clutter (50 obstacles)

Algorithm	Success Rate	Avg Iterations ($\pm$ SD)	Avg Path Cost ($\pm$ SD)	Avg Time
RRT	100.0%	572.3 $\pm$ 311.2	20.955 $\pm$ 2.512	0.155s
BIT*	100.0%	8871.1 $\pm$ 3522.2	24.213 $\pm$ 3.327	0.272s
ABIT*	100.0%	2949.7 $\pm$ 3034.9	22.044 $\pm$ 3.160	0.146s
ARRT	100.0%	260.4 $\pm$ 153.9	16.032 $\pm$ 1.376	0.083s

Table 4.3: Algorithm performance on clutter environment with 50 obstacles

4.2 Table 2: Overall Performance Across All Maps

Algorithm	Avg Success Rate	Avg Iterations	Avg Path Cost	Avg Time	Total Success Rate
RRT	100.0%	420.6	15.868	0.067s	100.0%
BIT*	100.0%	6421.7	17.600	0.140s	100.0%
ABIT*	100.0%	2963.9	15.374	0.076s	100.0%
ARRT	100.0%	290.1	11.729	0.039s	100.0%

Table 4.4: Overall comparison of algorithms across all maps

4.3 Table 3: Asymptotic Optimality Convergence Comparisons

Metric	RRT*	Informed RRT*	ARRT*
Success rate	100.0%	100.0%	100.0%
Mean final cost	15.293	15.252	20.537
Std dev final cost	0.031	0.014	1.363
Median final cost	15.276	15.259	20.695
Min final cost	15.266	15.233	18.794
Max final cost	15.337	15.264	22.122
Mean time to converge	0.120s	0.110s	0.150s

Table 4.5: Comparison of asymptotic optimality convergence between RRT*, Informed RRT* and ARRT*

Chapter 5

Discussion and Conclusion

From the table it can be seen that ARRT consistently outperforms all of the algorithms it is tested against. The time difference in standard RRT and ARRT indicate that goal biasing plays a major role in finding the initial path. ARRT also growing 1 node at time as opposed to processing multiple large batches at once also indicates the impact on speed of the algorithm. Performing very well in the cluttered environments shows how versatile and scalable ARRT can be for very complex situations. Strictly in terms of path, ARRT also performs very well returning the lowest average path cost in all environments for all the algorithms tested. This however, does not necessarily imply that the path will be smooth as the lack of waypoints in ARRT after the pruning process imply that sharp turns may need to be taken however, these can be mitigated by running ARRT* or by adding quadratic bezier curves at the vertices of the path. It should be noted however that ARRT may not perform as well in obtaining an optimal solution at a faster rate than BIT* and ABIT*.

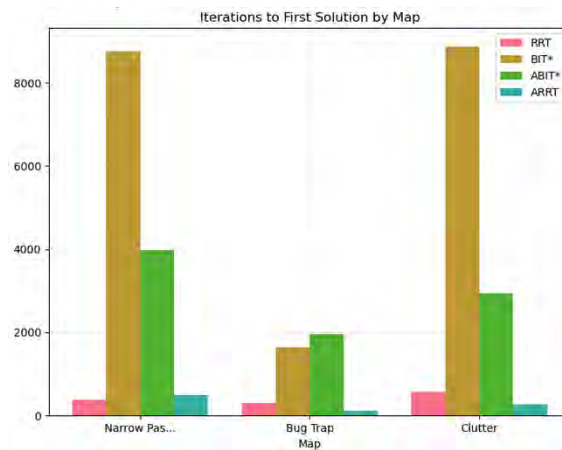


Figure 5.1: Iterations taken to first solution

In the cluttered environment ARRT obtained a path in more than 50% the time taken by BIT* and almost 20% more than that of ABIT. *However, there is a vast difference in initial path cost comparing ABIT and BIT* compared to ARRT where ARRT provides the most minimal path costs.*

For the narrow passage problem, the gap between the passage was taken to be ...

This is the problem where ARRT performs the worst in as it takes multiple tries before finding the proper resolution for the A* search and then finally finding a cost estimate for the major axis length or else it defaults to 1.2 times the L2 norm of the start and goal. If the gap between the 2 obstacles was smaller, the planner may not have been able to perform as well due to the narrowness of the gap hindering the performance of the A* search and hence the performance of the overall planner.

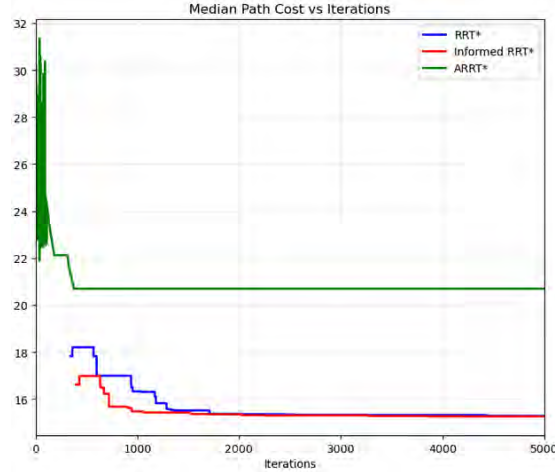


Figure 5.2: Cost change as iterations goes up

This leads into the curse of dimensionality, as the number of dimensions increases, the A* search will perform less well and hence it will be more difficult to estimate the L* cost. Hence, if ARRT is to perform well in $n > 3$ it will absolutely require coarse grids such that it can perform the searches properly.

In real time robotics and outdoor navigation, it is imperative that an initial path is found as fast as possible and it is also a great benefit if that path is not jagged and does not explore unnecessary locations. These objectives are perfectly fulfilled by ARRT surpassing other state of the art planners, making it the planner best fit as of now for outdoor navigation.

Bibliography

- [1] O. Khatib, “Real-time obstacle avoidance for manipulators and mobile robots,” *The International Journal of Robotics Research*, vol. 5, no. 1, pp. 90–98, 1986.
- [2] S. H. Suh and K. G. Shin, “A variational dynamic programming approach to robot-path planning with a distance-safety criterion,” *IEEE Journal on Robotics and Automation*, vol. 4, no. 3, pp. 334–349, 1988.
- [3] D. Fox, W. Burgard, and S. Thrun, “The dynamic window approach to collision avoidance,” *IEEE Robotics & Automation Magazine*, vol. 4, no. 1, pp. 23–33, 1997.
- [4] S. M. LaValle, “Rapidly-exploring random trees: A new tool for path planning,” Computer Science Department, Iowa State University, Tech. Rep. TR 98-11, 1998.
- [5] J. J. Kuffner and S. M. LaValle, “RRT-Connect: An efficient approach to single-query path planning,” in *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, San Francisco, CA, USA, 2000, pp. 995–1001.
- [6] C. Urmson and R. Simmons, “Approaches for heuristically biasing RRT growth,” in *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Las Vegas, NV, USA, 2003, pp. 1178–1183.
- [7] S. M. LaValle, *Planning Algorithms*. Cambridge, U.K.: Cambridge University Press, 2006.
- [8] R. Geraerts and M. H. Overmars, “Creating high-quality paths for motion planning,” *The International Journal of Robotics Research*, vol. 26, no. 8, pp. 845–863, 2007.
- [9] N. Ratliff, M. Zucker, J. A. Bagnell, and S. Srinivasa, “CHOMP: Gradient optimization techniques for efficient motion planning,” in *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, Kobe, Japan, 2009, pp. 489–494.
- [10] M. P. Akgun, A. M. Z. E. Kavraki, and T. Bretl, “Sampling heuristics for optimal motion planning in high dimensions,” in *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, San Francisco, CA, USA, 2011, pp. 2640–2645.
- [11] M. Kalakrishnan, S. Chitta, E. Theodorou, P. Pastor, and S. Schaal, “STOMP: Stochastic trajectory optimization for motion planning,” in *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, Shanghai, China, 2011, pp. 4569–4574.
- [12] S. Karaman and E. Frazzoli, “Sampling-based algorithms for optimal motion planning,” *The International Journal of Robotics Research*, vol. 30, no. 7, pp. 846–894, 2011.

- [13] S. Karaman, M. R. Walter, A. Perez, E. Frazzoli, and S. Teller, “Anytime motion planning using the RRT*,” in *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, Shanghai, China, 2011, pp. 1478–1483.
- [14] R. Siegwart, I. R. Nourbakhsh, and D. Scaramuzza, *Introduction to Autonomous Mobile Robots*, 2nd. Cambridge, MA, USA: MIT Press, 2011.
- [15] J. Pan, S. Chitta, and D. Manocha, “FCL: A general purpose library for collision and proximity queries,” in *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, Saint Paul, MN, USA, 2012, pp. 3859–3866.
- [16] K. Hauser, “Lazy collision checking in asymptotically-optimal motion planning,” in *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, Seattle, WA, USA, 2015, pp. 2951–2957.
- [17] M. Moll, I. A. Sucan, and L. E. Kavraki, “Benchmarking motion planning algorithms,” *IEEE Robotics & Automation Magazine*, vol. 22, no. 3, pp. 96–102, 2015.
- [18] A. H. Qureshi and Y. Ayaz, “Potential functions based sampling heuristic for optimal path planning,” *Autonomous Robots*, vol. 40, no. 6, pp. 1079–1093, 2016.
- [19] A. H. Qureshi, S. Mumtaz, Y. Ayaz, and O. Hasan, “Potential guided directional-RRT* for accelerated motion planning in cluttered environments,” in *Proceedings of the IEEE International Conference on Robotics and Biomimetics (ROBIO)*, Qingdao, China, 2016.
- [20] J. Kim, J. Lee, A. H. Qureshi, and J. J. Kim, “Adaptive lazy collision checking for optimal sampling-based motion planning,” in *Proceedings of the 15th International Conference on Ubiquitous Robots (UR)*, Honolulu, HI, USA, 2018.
- [21] Z. Tahir, A. H. Qureshi, Y. Ayaz, and R. Nawaz, “Potentially guided bidirectionalized RRT* for fast optimal path planning in cluttered environments,” *Robotics and Autonomous Systems*, vol. 108, pp. 13–27, 2018.
- [22] R. P. Padhy, S. Verma, S. Ahmad, S. K. Choudhury, and P. K. Sa, “Path planning for the autonomous mobile robot: A comprehensive review,” in *Paradigms of Computing, Communication and Data Sciences*, Springer, 2020, pp. 1–102.
- [23] L. Zhang et al., “Motion planning for robotics: A review for sampling-based planners,” *Biomimetic Intelligence and Robotics*, vol. 5, no. 1, p. 100 207, 2025.