

Fast Marching Trees (FMT*) For Dynamic Motion Planning



By

Muhammad Salman Sadiq

(Registration No: 00000329897)

Department of Robotics and Artificial Intelligence

School of Mechanical and Manufacturing Engineering

National University of Sciences & Technology (NUST)

Islamabad, Pakistan

(2024)

Fast Marching Trees (FMT*) For Dynamic Motion Planning



By

Muhammad Salman Sadiq

(Registration No: 00000329897)

A thesis submitted to the National University of Sciences and Technology, Islamabad,

in partial fulfillment of the requirements for the degree of

Master of Science in

Robotics and Intelligent Machine Engineering

Supervisor: Dr. Khawaja Fahad Iqbal

Co Supervisor: Dr. Sara Ali

School of Mechanical and Manufacturing Engineering

National University of Sciences & Technology (NUST)

Islamabad, Pakistan

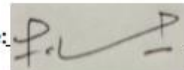
(2024)

THESIS ACCEPTANCE CERTIFICATE

THESIS ACCEPTANCE CERTIFICATE

Certified that final copy of MS/MPhil thesis written by Regn No. 00000329897 **Muhammad salman Sadiq** of **School of Mechanical & Manufacturing Engineering (SMME)** has been vetted by undersigned, found complete in all respects as per NUST Statues/Regulations, is free of plagiarism, errors, and mistakes and is accepted as partial fulfillment for award of MS/MPhil degree. It is further certified that necessary amendments as pointed out by GEC members of the scholar have also been incorporated in the said thesis titled. **Fast Marching Trees (FMT*) for Dynamic Motion Planning**

Signature:



Name (Supervisor): Khawaja Fahad Iqbal

Date: 11 - Oct - 2024

Signature (HOD):



Date: 11 - Oct - 2024

Signature (DEAN):





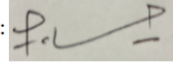
Date: 11 - Oct - 2024




National University of Sciences & Technology (NUST)
MASTER'S THESIS WORK

We hereby recommend that the dissertation prepared under our supervision by: Muhammad salman Sadiq (00000329897)
Titled: Fast Marching Trees (FMT*) for Dynamic Motion Planning be accepted in partial fulfillment of the requirements for the
award of MS in Robotics & Intelligent Machine Engineering degree.

Examination Committee Members

1. Name: Sara Baber Sial Signature: 
 2. Name: Yasar Ayaz Signature: 
- Supervisor: Khawaja Fahad Iqbal Signature: 
Date: 11 - Oct - 2024


Head of Department

11 - Oct - 2024
Date

COUNTERSIGNED

11 - Oct - 2024
Date


Dean/Principal

CERTIFICATE OF APPROVAL

This is to certify that the research work presented in this thesis, entitled “Fast Marching Trees (FMT*) for Dynamic Motion Planning” was conducted by Mr. Muhammad Salman Sadiq under the supervision of Dr Khawaja Fahad Iqbal. No part of this thesis has been submitted anywhere else for any other degree. This thesis is submitted to the Department of Robotics and Artificial Intelligence in partial fulfillment of the requirements for the degree of Master of Science in Field of Robotics and Intelligent Machine Engineering Department of Robotics and Artificial Intelligence National University of Sciences and Technology, Islamabad.

Student Name: Muhammad Salman Sadiq

Signature: 

Examination Committee:

a) External Examiner 1: NIL
(Designation & Office Address)

Signature: NA

.....

b) External Examiner 2: NIL
(Designation & Office Address)


Signature: NA

.....

Supervisor Name: Dr Khawaja Fahad Iqbal

Signature: 

Name of Dean/HOD: Dr Kunwar Faraz

Signature: 

AUTHOR'S DECLARATION

I, Muhammad Salman Sadiq hereby state that my MS thesis titled “Fast Marching Trees (FMT*) for Dynamic Motion Planning” is my own work and has not been submitted previously by me for taking any degree from National University of Sciences and Technology, Islamabad or anywhere else in the country/ world.

At any time if my statement is found to be incorrect even after I graduate, the university has the right to withdraw my MS degree.

Name of Student: Muhammad Salman Sadiq

Date: 11th, October, 2024

A handwritten signature in blue ink, appearing to read 'Muhammad Salman Sadiq', with a horizontal line extending to the right.

PLAGIARISM UNDERTAKING

I solemnly declare that research work presented in the thesis titled “Fast Marching Trees (FMT*) for Dynamic Motion Planning” is solely my research work with no significant contribution from any other person. Small contribution/ help wherever taken has been duly acknowledged and that complete thesis has been written by me.

I understand the zero tolerance policy of the HEC and National University of Sciences and Technology (NUST), Islamabad towards plagiarism. Therefore, I as an author of the above titled thesis declare that no portion of my thesis has been plagiarized and any material used as reference is properly referred/cited.

I undertake that if I am found guilty of any formal plagiarism in the above titled thesis even after award of MS degree, the University reserves the rights to withdraw/revoke my MS degree and that HEC and NUST, Islamabad has the right to publish my name on the HEC/University website on which names of students are placed who submitted plagiarized thesis.

Student Signature: _____



Name: Muhammad Salman Sadiq

Dr. Khawaja Fahad Iqbal
Assistant Professor
Deptt of Robotics & AI
SMME, NUST, Islamabad



DEDICATION

To my loving mother.

ACKNOWLEDGEMENTS

I consider myself fortunate that I gained the chance to benefit from the highly skilled and erudite people working at the cutting edge of technology of the National Center of Artificial Intelligence at National University of Science and Technology. An engineer must thank many people for it is truly on the back of giants that ordinary men can peer into the humongous vastness of science. Firstly, I would like to extend my deepest heartfelt gratitude to my supervisor, Dr Khawaja Fahad Iqbal who accepted me as an MS Student and stood by and guided me profusely while I was engaged in my thesis. I would also like to thank my GEC member and Chairman NCAI, Dr Yasar Ayaz, who inspired me towards motion planning algorithms and whose guidance at the formative phases of my research was invaluable. Moreover, I would also like to thank my mother, Dr Sadiqa Arshad for her support during the trying and testing times when all hope seemed to be lost

Salman Sadiq

TABLE OF CONTENTS

ACKNOWLEDGEMENTS	VIII
TABLE OF CONTENTS	IX
LIST OF TABLES	X
LIST OF FIGURES	XI
LIST OF SYMBOLS, ABBREVIATIONS AND ACRONYMS	XII
ABSTRACT	XIII
CHAPTER 1: INTRODUCTION	1
1.1 Background	1
1.2 Motivation	3
1.3 Research Contributions	4
1.4 Problem Definition	4
1.5 Thesis Overview	5
CHAPTER 2: BACKGROUND AND LITERATURE REVIEW	7
2.1 Sampling based Motion Planning Algorithm	8
2.1.1 Randomly exploring Random Tree (RRT) family of planners	9
2.1.2 Potential function guided RRT*	13
2.1.3 Fast Marching Tree Algorithm	14
2.3.1 Dynamic Motion Planning	29
CHAPTER 3: METHODOLOGY	31
3.1 Dual Tree Fast Marching Tree (DT-FMT*)	32
3.1.1 Working Principle	32
3.1.2 Tunnel Construction	34
3.1.2 Tunnel Construction	38
3.2: Reduced Sampling Re-planning Fast Marching Tree (RRFMT*)	40
3.2.1 Working Principle	40
CHAPTER 4: EXPERIMENTATION AND DISCUSSION	46
4.1 Dynamic Environments	52
4.1 Analysis	54
CHAPTER 5: CONCLUSION AND FUTURE WORK	55
REFERENCES	56

LIST OF TABLES

	Page No.
Table 1: Path Distance and Time Comparison	47
Table 2: Comparison with FMT*	48
Table 3: Comparison with ST-FMT* (preprocessing included)	50
Table 4: Path Distance and Time Comparison (Dynamic)	53
Table 5: T-test (Distance and Time)	53

LIST OF FIGURES

Figure 1: Scenarios in dynamic motion planning.....	2
Figure: 2: Re-planning a path in the presence of dynamic obstacles	3
Figure 3 Tree Growth Method of RRT Algorithm.....	10
Figure 4: A series of images illustrating FMT* algorithm steps	16
Figure 5: Distance Comparison of FMT*, RRT, P-RRT, RRT*, P-RRT*.....	19
Figure 6: Time Comparison of FMT*, RRT, P-RRT, RRT*, P-RRT*.....	20
Figure 7: Comparison of FMT*	22
Figure 8: A rendering of Generalized Voronoi Graph with multiple configurations.....	23
Figure 9: FMT* performance with an increasing number of samples.	24
Figure 10: Generalized Voronoi Graph (GVG)	28
Figure 11: An overview of the DT-FMT* algorithm..	34
Figure 12: Tunnel construction methodology for DT-FMT*	35
Figure 13:Rendering of the performance of DT-FMT*	37
Figure 14: Different type of sampling distributions.	38
Figure 15: Different type of sampling distributions.	39
Figure 16: Procedure of Reduced Sampling Re-planning FMT*	41
Figure 17:Complete process of Dynamic Path computation	42
Figure 18: Working procedure of RR-FMT*.....	45
Figure 19: List of Classical Motion Planning Environments used in this work.	46
Figure 20: Distance and Time Comparisons Map 1-3.....	51
Figure 21: Distance and Time Comparison for Maps 4-6	52
Figure 22: Status of the environment at different values of time.....	53

LIST OF SYMBOLS, ABBREVIATIONS AND ACRONYMS

$X \subset \mathbb{R}^n$	Configuration space
X_{obs}	Obstacle space
X_{free}	Obstacle-free space
X_{goal}	Goal Region
x_{goal}	Goal state
x_{start}	Initial state
τ^*	Optimum path
$Cost(\tau)$	Cost function of path
AO	Asymptotically Optimal
RRT	Randomly Exploring Random Tree
RRT*	Randomly Exploring Random Tree (optimal)
P-RRT*	Potential guided Randomly Exploring Random Tree
PRM*	Probabilistic Roadmap Methods GVG Generalized Voronoi Graph
FMT*	Fast Marching Tree
aFMT*	Anytime Fast Marching Tree
IAFMT*	Informed Anytime Fast Marching Tree
PRM	Probabilistic Roadmap Methods
PRM*	Probabilistic Roadmap Methods(optimal)
DT-FMT*	Dual Tree Fast Marching Tree
RR-FMT*	Reduced Sample Re-planning Fast Marching Tree
SBL	Single-query Bidirectional Lazy
ST-FMT*	Secure Tunnel Fast Marching Tree

ABSTRACT

In light of recent advances in autonomous mobile robots, the chance for the robot presence in human domains have increased. To avoid collisions and to compute the optimal path between two points, motion planning has come to the fore as an essential area of research. Sampling based motion planners offer an advantage with respect to computational cost as in contrast to conventional planners they avoid an explicit construction of cspace. However, two of the major problems of sampling based motion planners is the need to efficiently adapt in the presence of dynamic obstacles and the degradation of path quality with a reduced number of samples. Much work has been done in order to adapt existing sampling based motion planning algorithms, including Randomly exploring Random Trees (RRT,RRT*), Probabilistic Roadmap Methods (PRM,PRM*), for dynamic scenarios. In order to solve the above-mentioned problems, we introduce two different sampling algorithms in order to solve the above mentioned problems. Firstly, Dual Tree Fast Marching Tree (DT-FMT*) is an asymptotically optimal static motion planning algorithm that is used to improve the path quality with a limited number of initial samples. It does this by quickly computing an initial path and uses that information to draw a batch of new samples to generate an improved path. Secondly, we introduced Reduced samples Re-planning Fast Marching Tree (RR-FMT*) in order to modify an initial path in presence of dynamic obstacles. This is done by, first, computing an initial path using DT-FMT*, then during the course of robot motion along the path, we monitor the presence of obstacle at a certain clearance. In case of obstruction along the path, we grow a new tree to connect the current position of the robot to a way-point along the path. To validate our planner performance we have rigorously tested our both DT-FMT* and RR-FMT* performance against standard version of FMT*, as well as Secure tunnel FMT* (ST-FMT*). Similarly, in a dynamic environment, we compared planner performance against a dynamic version

of RRT* planner. The result show an overall improvement with respect to both path cost and time taken to compute the path.

Keywords: Sample based motion planning, dynamic environment, optimal path planning, Fast Marching Tree (FMT*), computational efficiency

CHAPTER 1: INTRODUCTION

1.1 Background

Thanks to a rise in demand for intelligent systems in various fields, from self-driving cars [1] and unmanned aerial [2] and underwater vehicles [3], to a wide array of task specific robots, motion planning has come to the forefront to fulfil one of the most challenging, and pertinent, research problem: to develop an efficient and robust algorithm for safely navigating in a dynamic environment with a wide range of unknown obstacles. Sampling based motion planners, introduced in the 1990s [4], leverage random sampling in order to construct a graph to find an obstacle free path. An advantage of this is that they do not require exploring the full configuration space; hence, ensuring faster and more efficient solution. However, motion planning in dynamic environments requires the ability of a robot to modify its plan on the fly.

Dynamic Environments introduce, a time constraint to the problem of motion planning [5]. Formally, dynamic environments are defined as environments where obstacle configuration is not known by the robot before starting motion, or the obstacle configuration can change with time or obstacles can move with time [6]. Fig 1 (a) to (h) gives the complete information about the types of dynamic environments a robot is likely to encounter. Case 1 includes environments that are static but whose presence and location are not known the robot when it starts moving. Case 2 consists of obstacle that appear when the robot is in a certain proximity. Case 3 consists of obstacles that appear at different intervals and also disappear when a certain time period is reached. Case 4 consists of moving obstacle that can follow a particular trajectory. In a typical implementation of the motion planning scenario, where the initial path is first computed in then motion begins, of motion planners, dynamic obstacles, can cause the invalidity of computed paths. Hence, the dynamic re-planner, is responsible for the modification, and optimization of a given trajectory, in response to the presence of unexpected obstacles which can come as one, or a combination of Case 1 to 4.

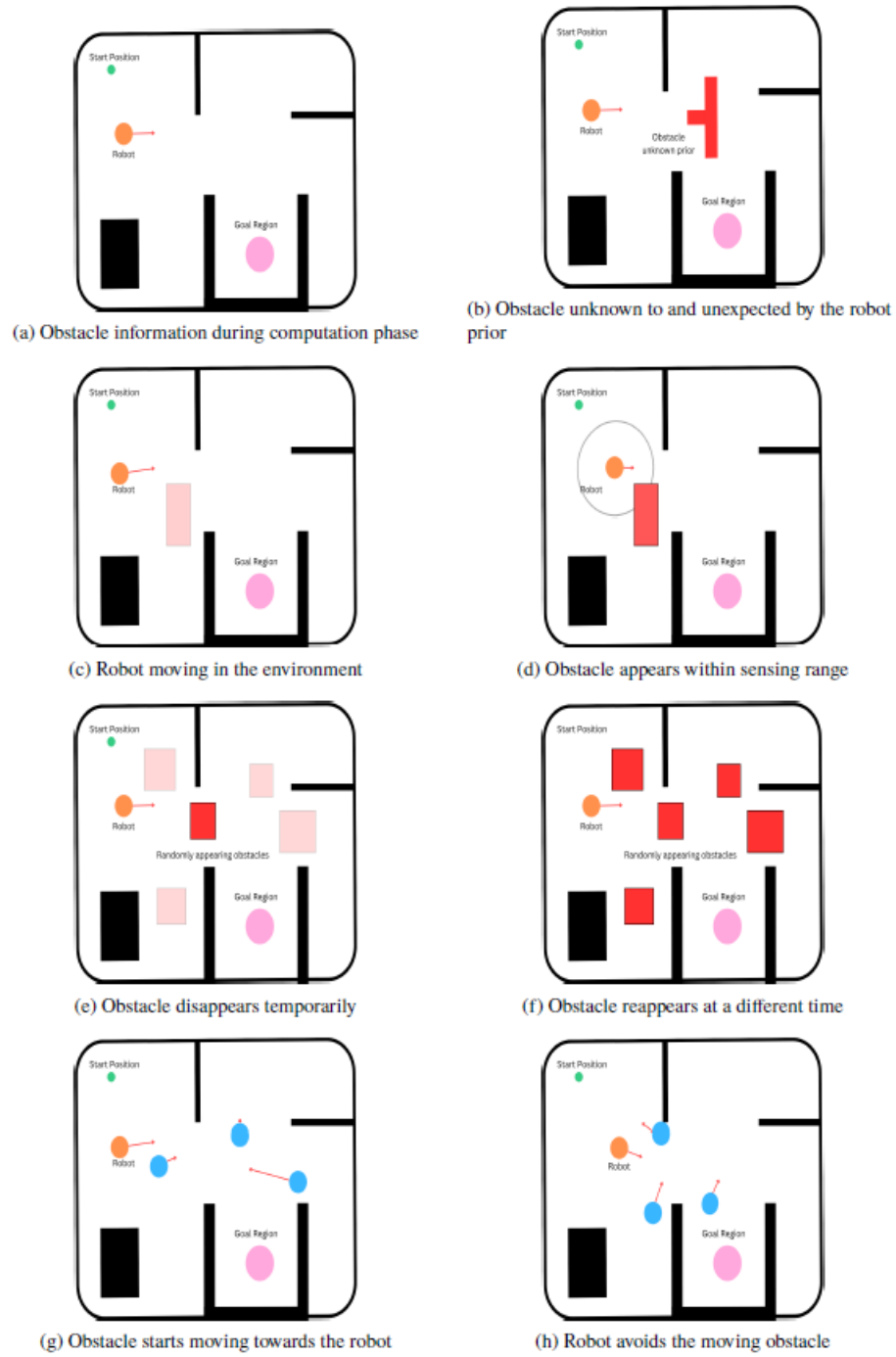


Figure 1: Scenarios in dynamic motion planning: (a-b) Case 1: Unknown obstacles; (cd) Case 2: Sense-able obstacles; (e-f) Case 3: Randomly appearing obstacles; (g-h) Case 4: Moving obstacles.

Fast Marching Tree (FMT*) algorithm is a sampling-based motion planning algorithm, based upon lazy forward recursion [7]. They were introduced in 2015 by Janson et al. [7]. It draws inspiration from the Fast-Marching Methods, used for a numerical solution of Eikonal equations [5]. The FMT* algorithm has a number of advantages over contemporary sampling-based planner. For one, it reduces the number of collision checks [8], whilst also offering a faster convergence [9]. However, during the course of this thesis we have highlighted two of the major problems of FMT* Algorithms. One, as a batch sampling algorithm, in which the total samples are taken at the beginning of the sampling process, as the number of samples increase the total time taken increases. Secondly, there have been no optimal dynamic version of the FMT* Algorithm [10].

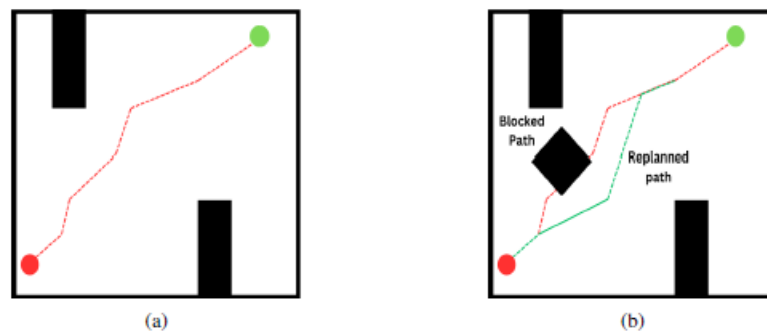


Figure: 2: Re-planning a path in the presence of dynamic obstacles

1.2 Motivation

As robots become increasingly prominent and plentiful in public and domestic environments, their ability to both plan its path quickly and locomote along the path rapidly becomes indispensable. Two of the key problems highlighted earlier, slow convergence speed and slow re-planning speed are key issues in today's robotics not just at a simulation level, but on roads and public spaces where a lack of there can have disastrous consequences. This project explores the use of sampling-based planners, a category of planners enjoying the advantages of both faster speed and better path over other categories of robot motion planner, in dynamic environments.

Personally, being able to contribute to this field of research is highly motivating, as motion planning for dynamic environments poses both theoretical and practical challenges. In addition, working on this field of study allows robots to be adaptive to their surroundings which can have a wide range of impact on the world from efficient and safe transportation system as well as improved manipulator path.

1.3 Research Contributions

The purpose of this thesis is to present a modified FMT* Algorithm, for planning in dynamic environment, better convergence, and reduced computation, as well as to insulate the FMT* Algorithm to the effects of moving obstacles, on its trajectory. Our contribution is as follows:

- Introduction of a Dual Tree Fast Marching Tree (DT-FMT*) for path improvement in a reduced sample set
- As a motion planner Double-FMT* planner carries forward the asymptotic optimality and probabilistic completeness of FMT*, but with a much faster path generation in static environment with a reduced sample set.
- Experimentation with 3 different sampling strategies based on obstacle distance for path improvement in dynamic environment
- A tunnel construction strategy in order to ensure high quality contiguous samples
- Introduction of a novel Waypoint based path replanning using RR-FMT* planner in dynamic environments
- Rigorous testing in various scenarios to prove the viability of our planners

1.4 Problem Definition

In this section we elucidate the commonly used terminology and notations pertaining to sampling-based motion planners in order to facilitate greater understanding in latter sections. Additionally, we also formalize the definition of feasible and optimal motion planning. Furthermore, we also review and discuss the standard FMT* algorithm alongside the ST-FMT* algorithm. For a configuration space $X \subset \mathbb{R}_n$, let X_{obs} be the obstacle region

such that the $X \setminus X_{\text{obs}}$ is an open set. Resultantly, the obstacle-free space is defined as $X_{\text{free}} = \text{cl}(X \setminus X_{\text{obs}})$. The initial x_{start} and goal state x_{goal} are elements of X_{free} . Alternately, the goal region X_{goal} is an open subset of X_{free} . $\tau : [0, 1] \rightarrow \mathbb{R}^d$ is defined as a continuous sequence with a bounded variation. The sequence τ can be defined as a path if it is continuous and collision-free.

Feasible Path Planning: For a standard motion planning formulation $(x_{\text{start}}, X_{\text{free}}, x_{\text{goal}})$, find a feasible path $\tau : [0, 1] \rightarrow X_{\text{free}}$ if existing such that the $\tau [0] = x_{\text{start}}$ and $\tau [1] = x_{\text{goal}}$. If no extant path fulfilling this criteria, then return failure.

Optimal Path Planning: For the above-mentioned motion planning problem and a series of all feasible paths Σ . The cost function $Cost(\tau)$ denotes the cost of a particular path per the distance metrics used. The optimal path is such that it should satisfy the following criteria:

$$\tau^* = \underset{\tau \in \Sigma}{\text{argmin}} Cost(\tau) \mid \tau : [0,1] \rightarrow X_{\text{free}} \quad (1.1)$$

Fast Path Planning: For a given motion planning problem, find the optimal and feasible path τ^* in the least possible amount of time $t \subset \mathbb{R}$.

1.5 Thesis Overview

Chapter 2 is a review of motion planning algorithms consulted during the course of our thesis. It provides a comprehensive grounding in basic static motion planners, including RRT, RRT*, Potential-RRT, FMT* etc, as well as a select dynamic motion planners, including the Risk RRT-RRT* family. We also discuss recent research on these planners. These have been instrumental in developing our approach towards planning in dynamic environments. We also include our implementations of these motion planners with comparison in a comprehensive list of environments selected from literature.

Chapter 3 describes the scope and details of the modification that we have made to the FMT* algorithm. We include the details and complete pseudo-code of both the Dual Tree Fast Marching Tree as well as the Reduced Sampling FMT*. Additionally, we also outline the extension of FMT* algorithms to dynamic environments with a certain number of obstacles that are not known a priori. We include certain number of obstacles that appear

and disappear randomly with time. Moreover, the total system design for rapid path computation is also presented.

In Chapter 4 we present the result of the modification we have discussed above. We have made a wide range of sampling environments based on the literature. Separate experiments are conducted for DT-FMT* and RR-FMT*, and comparisons were made using the appropriate planner. DT-FMT* with standard FMT* and ST-FMT*, while RR-FMT* was compared with the Dynamic RRT* algorithm.

Chapter 5 provides a conclusion and discusses the potential future work, including the extension of this work to a physical system; as well as the future work.

CHAPTER 2: BACKGROUND AND LITERATURE REVIEW

Due to the advances in artificial intelligence, autonomous mobile robots are en-route to have an unprecedented level of access, to domains of human activity; from roads [11] and other public places [12] [13] [14], to the private lives of senescent individuals [15]. Concurrently, the usage of robots has also diversified to encompass a wide range of applications: service in restaurants [16], garbage removal [14], and working in space [17]. This has increased the chances of robots encroaching on human domains. Consequently, motion planning, which is responsible for ensuring collision-free robot motion from one point to another, has come to the fore as an indispensable part of robotics.

Thanks to the extensive work done, motion planning algorithms can be divided into four categories: combinatorial approaches [18], reactive planners [19], learning based planners [20], and sampling based planners [21]. Combinatorial approaches to motion planning compute an exact representation of the free space and return a complete solution; however, one disadvantage of these approaches is that they are computationally expensive. Reactive planners monitor the state of the environment and use stimulus from the environment to plot an obstacle free trajectory, but, while they have utility in real time operation, they give no guarantee of optimality with respect to path length [22]. Learning based planners work by utilizing either a human designed reward function to bias behaviour [23] or by learning viable solutions from prior iterations to compute an optimal path. While they have the advantage of being flexible and handle complex tasks; however, apart from being computationally expensive, they are dependent upon pre-computed data [24]. Sampling based planners work by generating a random configuration in order to find the path between two points [4]. Sampling based planners are easily scalable to higher dimensions; however, their performance is contingent upon the number of samples taken in the configuration space.

2.1 Sampling based Motion Planning Algorithm

Sampling based planners have become popular due to the distinct advantages they hold over conventional planners. Firstly, by avoiding the explicit representation of the c-space they ensure fast computation of feasible solutions especially in higher dimensions [7]. Secondly, as a result of the incremental nature of most sampling planner, they can reach a quick suboptimal solution first, before moving onto an optimal solution. Two of the most prolific sampling-based planners are Randomly exploring random trees (RRT) and probabilistic roadmap (PRM). PRM samples a roadmap from the environment and during the query stage uses a graph planner to find the minimum cost solution [25]. In contrast, the RRT algorithm incrementally grows a tree structure through generating random samples in the environment. Both of these planners guarantee probabilistic completeness, but are not asymptotically optimal (AO). Karaman [21] et al. demonstrated that the solution return by the RRT was not optimal and presented the RRT* and PRM* algorithms as the optimal versions of these planners. PRM and PRM* require the pre computation of the environment in order to generate the roadmap necessary. This is computationally expensive and not usable in cases where the environment is not known.

In addition, although it has been proven to be asymptotically optimal, however, one of the problems inherent within RRT* planner is the inherently slow rate of convergence to the optimal solution on account of the vast number of iterations needed. Qureshi et al. [26] gave the Potential guided directional-RRT* as a means of guiding random samples toward the direction of decreasing potential, hereby decreasing the number of iterations required to converge to an optimal solution. Similarly, the obstacle RRT algorithm [27] uses the obstacle location to create a vector in order to modify samples for use in generating solutions in narrow regions.

Ayaz et al. [28] introduced RRT*-Smart which optimizes an initial path and identifies beacon nodes in whose direction to bias sampling for a better quality path. Moreover, the Informed-RRT* algorithm [29] also increases the rate by convergence

by, firstly, quickly computing an initial solution and then concentrating the sampling on states admissible by a pre-defined heuristic.

2.1.1 Randomly exploring Random Tree (RRT) family of planners

In this section we introduce the Randomly exploring Random Tree (RRT) Algorithm along with the optimal version of this planner (RRT*). It is necessary to discuss these planners as they will be in use later on in this work. Lavelle et al. gave the basic version of RRT in 1998 [25]. It was designed to handle a wide variety of motion planning problems especially those involving high dimensions. Algorithm 1 gives the complete pseudocode of the algorithm. As a single query motion planning algorithm, the RRT algorithm works by incrementally building a tree structure in order to explore a given space. The 'root' of the tree is at the start position and it gradually grows by randomly sampling points in the environment. The process begins by initializing the tree with a single vertex at the initial state, and no edges. At each iteration, a random point $x_{rand} \in X_{free}$ is sampled from the environment by *SampleFree* in line 3 in Algorithm 1. The algorithm next tries to find the nearest vertex of the tree $v \in V$ to x_{rand} (line 4) and returns x_{near} . Then the Steer function steers a new sample x_{new} towards x_{near} . If the edge between x_{near} and x_{new} is collision free (line 6) then it is added to the tree structure G . The algorithm runs for a fixed number of iterations and afterwards returns the graph structure. Then using the x_{goal} node, we backtrack and get a series of path leading from the goal to x_{start} . The algorithm then chooses the minimum cost path:

Algorithm 1 RRT Algorithm

1: $V \leftarrow \{x_{\text{init}}\}; E \leftarrow \emptyset$	▷ Initialize tree with initial vertex
2: for $i = 1, \dots, n$ do	
3: $x_{\text{rand}} \leftarrow \text{SampleFree}()$	▷ Sample a random point in free space
4: $x_{\text{near}} \leftarrow \text{Nearest}(G = (V, E), x_{\text{rand}})$	▷ Find nearest node in the tree
5: $x_{\text{new}} \leftarrow \text{Steer}(x_{\text{near}}, x_{\text{rand}})$	▷ Steer towards the random sample
6: if $\text{CollisionFree}(x_{\text{near}}, x_{\text{new}})$ then	▷ Check if the path is collision-free
7: $V \leftarrow V \cup \{x_{\text{new}}\}$	▷ Add new node to the tree
8: $E \leftarrow E \cup \{(x_{\text{nearest}}, x_{\text{new}})\}$	▷ Add edge to the tree
9: end if	
10: end for	
11: return $G = (V, E)$	▷ Return the constructed tree

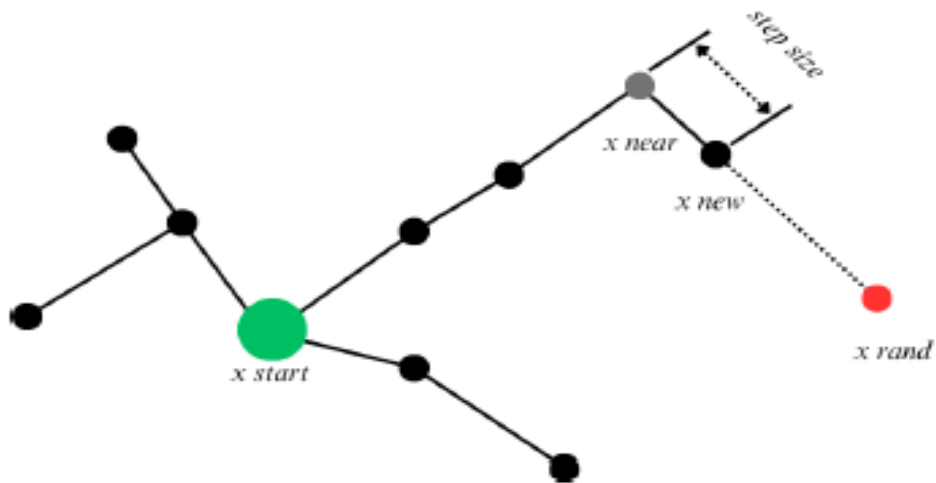


Figure 3 Tree Growth Method of RRT Algorithm

There has been extensive work done on RRT algorithm. Some of the most famous variants include RRT-Connect [30] which maintains two trees: one from x_{start} and the other of x_{goal} . The algorithm completes when the two trees meet, significantly enhancing search efficiency compared to single-tree methods and greatly increasing the speed required to reach a solution. The Single-query Bidirectional Lazy (SBL) planner [31] is also maintains two different trees; however, it maintains a lazy collision checking strategy. Similarly to

SBL and RRT-Connect, Triple RRT [32] generates three trees; apart from the trees from the goal region and start region, a separate tree is generated within a narrow corridor region. As a result of this, path performance is improved in narrow corridor environments. Anytime-RRT [33] works by first computing an initial sub-optimal path. It then continues to improve the tree deals with lack of computational time for path improvement by generating an initial sub-optimal solution. The tree is then stored and the rest of the time is used to attempt to improve the solution by running iterations of the RRT. A solution is returned if the initial path is improved by a pre-determined method. However, the RRT algorithm has a set of limitations. Firstly, Karaman *et al.* proved that the planner does not guarantee an optimal solution. This means that, practically, the path turned may be unnecessarily long, containing detours or redundant segments. Additionally, since RRT uses random sampling to explore the space, it focuses on rapid exploration rather than optimizing for the shortest or smoothest path. Furthermore, RRT struggles to efficiently explore narrow passages in the configuration space, as there is a reduced probability of sampling in narrow obstacle environments [34].

Algorithm 2 RRT* Algorithm

```

1: Input: Start node  $x_{start}$ , Goal node  $x_{goal}$ , Max iterations  $N$ , Step size  $\delta$ , Search radius  $r$ 
2: Output: Optimal path from  $x_{start}$  to  $x_{goal}$ 
3: Initialize tree  $T$  with root node  $x_{start}$ 
4: for  $i = 1$  to  $N$  do
5:    $x_{rand} \leftarrow \text{SampleRandomPoint}()$ 
6:    $x_{nearest} \leftarrow \text{FindNearest}(T, x_{rand})$ 
7:    $x_{new} \leftarrow \text{Steer}(x_{nearest}, x_{rand}, \delta)$ 
8:   if  $\text{CollisionFree}(x_{nearest}, x_{new})$  then
9:      $X_{near} \leftarrow \text{FindNearbyNodes}(T, x_{new}, r)$ 
10:     $x_{best} \leftarrow \text{ChooseBestParent}(X_{near}, x_{new})$ 
11:     $\text{AddNode}(T, x_{new}, x_{best})$ 
12:     $\text{Rewire}(T, X_{near}, x_{new})$ 
13:   end if
14:   if  $\|x_{new} - x_{goal}\| < \delta$  and  $\text{CollisionFree}(x_{new}, x_{goal})$  then
15:     return  $\text{ConstructPath}(T, x_{start}, x_{goal})$ 
16:   end if
17: end for
18: return  $\text{BestPathFound}(T, x_{start}, x_{goal})$ 

```

Algorithm 3 Rewire

```
1: procedure REWIRE( $T, X_{\text{near}}, x_{\text{new}}$ )
2:   for each  $x_{\text{near}} \in X_{\text{near}}$  do
3:     if CollisionFree( $x_{\text{new}}, x_{\text{near}}$ ) then
4:        $\text{new\_cost} \leftarrow \text{Cost}(x_{\text{new}}) + \text{Distance}(x_{\text{new}}, x_{\text{near}})$ 
5:       if  $\text{new\_cost} < \text{Cost}(x_{\text{near}})$  then
6:         ChangeParent( $T, x_{\text{near}}, x_{\text{new}}$ )
7:       end if
8:     end if
9:   end for
10: end procedure
```

In order to address ensure asymptotic optimality in the RRT, Karaman *et al.* introduced the RRT* version of the planner. Algorithm 2 gives the complete pseudocode of the algorithm. It is similar to the RRT; however, it introduces a rewiring of new nodes (line 12). After inserting x_{new} into the tree in the same manner as that of RRT, the algorithm identifies a set of nearby nodes within a radius $r(n)$ of x_{new} . The value of $r(n)$ is written as such:

$$r(n) = \gamma_R \left(\frac{\log(n)}{n} \right)^{\frac{1}{d}} \quad (2.1)$$

where γ_R is the constant dependent upon the problem and d is the dimension of the space. For a set vertices $x_{\text{neighbour}}$ within $r(n)$, the algorithm calculates the cost between x_{new} and $x_{\text{neighbour}}$

$$c_{\text{new}}(x_{\text{new}}) = c(x_{\text{neighbour}}) + c_{\text{edge}}(x_{\text{neighbour}}, x_{\text{new}}) \quad (2.2)$$

If any of the neighbours offers a lower cost alternative then the parent of x_{new} is set to $x_{\text{neighbour}}$:

$$c_{\text{new}}(x_{\text{new}}) < c(x_{\text{neighbour}}) \quad (2.3)$$

However, this is not to say that RRT* planner is impeccable. The following gives the drawback of RRT*:

- While RRT* eventually converges to the optimal path, its rate of convergence is often quite slow. This is because the algorithm refines its tree structure, rewiring it to improve path quality. In high-dimensional or expansive spaces, a large number of iterations and samples are needed before the path approaches optimality. Therefore, achieving an optimal or near-optimal solution can be time-consuming, making RRT* unsuitable for applications requiring quick planning.
- Additionally, another problem is that RRT* is difficult planning through narrow spaces and mazes. The reason for this is evident: the random sampling approach makes the probability of generating samples within these constrained regions. As a result, finding and exploring narrow passages can be inefficient, with the algorithm either taking a long time to discover these areas or failing to do so altogether, leading to sub-optimal or incomplete solutions in environments with tight constraints.

In conclusion, while RRT* provides the benefit of converging to an optimal path over time, its limitations—especially its slow exploration in large or complex spaces, delayed convergence, and difficulty handling narrow passages—highlight scenarios where it may not be the most effective choice.

2.1.2 Potential function guided RRT*

One solution to the above mentioned problem in RRT* is the Potential function guided directional RRT* devised by Quershi, which integrates the Artificial Potential Field algorithm [35] to improve convergence speed and achieve a faster optimal path. The *potentialized random sample*, denoted as $z_{prand} \in Z$, and the *step size*, represented by $\alpha \in R$, apply an adjustment to the random state $z_{rand} \in Z$ in the direction of the decreasing potential field gradient. This direction is indicated by $f = \nabla$. The terms d_{obs} and d_{goal} , previously introduced, indicate the respective distances to obstacles and the goal.

Furthermore, the scaling factors k_a and k_r represent the strengths of the attractive and repulsive potentials, respectively.

2.1.3 Fast Marching Tree Algorithm

In this section we present the Fast Marching Tree algorithm [7]. Algorithm 6 provides a detailed overview of the algorithm. The set $V_{unvisited}$ consists of nodes not yet added to the tree.

Algorithm 4 P-RRT* Path Planning

```

1: Initialize:  $T = (V, E)$  with  $V = \{z_{init}\}$ ,  $E = \emptyset$ 
2: Initialize: tree  $T'$  with  $z_{init}$  and empty  $E'$ 
3: for  $i = 0$  to  $N$  do
4:    $z_{rand} \leftarrow \text{SampleFree}$  ▷ Sample from the sample space
5:    $z_{prand} \leftarrow \text{RGD}(z_{rand})$ 
6:    $z_{near} \leftarrow \text{Nearest}()$ 
7:    $(x_{new}, y_{new}, z_{new}) \leftarrow \text{Steer}(z_{prand}, z_{rand})$ 
8:   if  $\text{CollisionFree}(z_{prand}, z_{new})$  then
9:      $Z_{near} \leftarrow \text{Near}(T, z_{new}, |V|)$ 
10:     $z_{min} \leftarrow \arg \min_{z \in Z_{near}} \{\text{Cost}(z_{prand}, z_{new}) + \text{Cost}(z_{new}, z)\}$ 
11:    Insert  $z_{new}, z_{min}$  into  $T$ 
12:    Rewire $(T, z_{min}, z_{new}, Z_{near})$ 
13:   end if
14: end for

```

Algorithm 5 RGD

```

1: Initialize:  $Watt, Wrep \leftarrow$  Potential gradient field of obstacles and goal
2:  $\alpha \leftarrow \text{ComputeStepSize}(Watt, Wrep, rand)$ 
3:  $W \leftarrow Watt + Wrep$ 
4:  $f \leftarrow \frac{W}{|rand|}$ 
5:  $rand \leftarrow rand + \alpha \cdot f$ 
6: Return  $rand$ 

```

Initially, the entire sample set is added to $V_{unvisited}$. Similarly, the V_{open} consists of nodes considered 'active' for tree growth. One key feature of FMT* is that it generates a readily growing set of paths and performs graph construction and graph search synchronously. Two samples are considered neighbours if their distance is within a certain bound. As FMT* is a batch sampling algorithm, $\text{SampleFree}(n)$ generates the samples required to

compute the solutions. Normally, as in this case, uniform distribution is used; however, it has been shown [7] to work with non-uniform sampling distributions. After an initial amount of samples are generated across the environment, FMT* works by performing a forward dynamic recursion on the set of samples. The function $Near(V_{unvisited}, z, r_n)$ returns a subset of samples within the radius r_n . The cost function $Cost(y, x)$ is used to denote the cost of the straight line between configuration y and x in cost to go space. Furthermore, after the minimum cost node is within the goal region, the planner terminates and $Path(z, T = (V_{open} \cup V_{closed}, E))$ returns the optimal path from the tree. To determine the validity of the $Collision(y, x)$ which is a boolean operation that returns True in case of an intersection of path with obstacles.

2.1.3.1 Working Principle

After n samples have been taken across X_{free} , they are initially placed within $V_{unvisited}$ with x_{start} being placed within V_{open} . At least one sample must be within the goal region X_{goal} . Afterward, a tree is initialized with its root nodes at x_{start} . Then the lowest cost node z is selected. $Near$ returns X_{near} the subset of z neighbours in $V_{unvisited}$. Then, for each x in X_{near} , the algorithm tries to find the neighbouring nodes y within V_{open} . It then considers the most locally optimal of these node for incorporation into the tree and performs a collision check in order to determine if the edge is obstacle-free. If so then it adds y to the tree. Such a collisionchecking strategy is known as 'lazy' [36]. Node x is then shifted to V_{open} and z moves to V_{closed} , the set ineligible for further expansion. This process repeats itself until the tree reaches X_{goal} . In case V_{open} is empty, then the planner returns *Failure*.

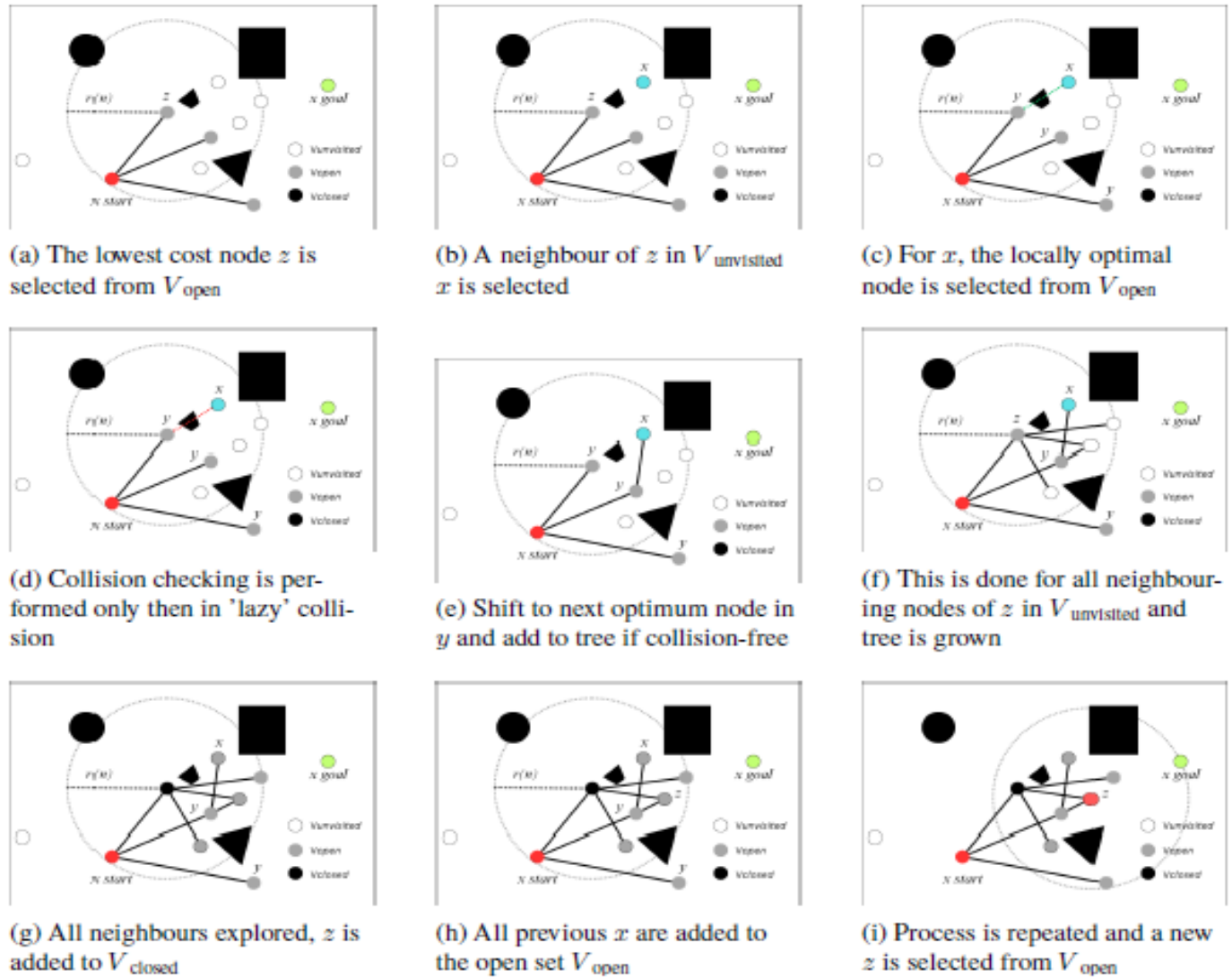


Figure 4: A series of images illustrating FMT* algorithm steps

Algorithm 6 Fast Marching Tree Algorithm (FMT*)

```
1: Input:  $x_{\text{start}}, x_{\text{goal}}, n, \text{environment}$ 
2: Output: Path( $z, T = (V_{\text{open}} \cup V_{\text{closed}}, E)$ )
3: SampleSet  $\leftarrow$  SampleFree( $n$ )
4:  $V \leftarrow \{x_{\text{start}}, x_{\text{goal}}\} \cup \text{SampleSet}$ 
5:  $E \leftarrow \emptyset$ 
6:  $V_{\text{unvisited}} \leftarrow V \setminus \{x_{\text{start}}\}$ 
7:  $V_{\text{open}} \leftarrow \{x_{\text{start}}\}$ 
8:  $V_{\text{closed}} \leftarrow \emptyset$ 
9:  $z \leftarrow x_{\text{start}}$ 
10: while  $z \neq x_{\text{goal}}$  do
11:    $X_{\text{near}} \leftarrow \text{Near}(V_{\text{unvisited}}, z, r_n)$ 
12:    $V_{\text{open,new}} \leftarrow \emptyset$ 
13:   for  $x \in X_{\text{near}}$  do
14:      $Y_{\text{near}} \leftarrow \text{Near}(V_{\text{open}}, x, r_n)$ 
15:      $y_{\text{min}} \leftarrow \arg \min_{y \in Y_{\text{near}}} \{c(y) + \text{Cost}(y, x)\}$ 
16:     if Collision( $y_{\text{min}}, x$ ) == False then
17:        $G(x) \leftarrow G(y_{\text{min}}) + \text{Cost}(y_{\text{min}}, x)$ 
18:        $E \leftarrow E \cup \{(y_{\text{min}}, x)\}$ 
19:        $V_{\text{open,new}} \leftarrow V_{\text{open,new}} \cup \{x\}$ 
20:        $V_{\text{unvisited}} \leftarrow V_{\text{unvisited}} \setminus \{x\}$ 
21:     end if
22:   end for
23:    $V_{\text{open}} \leftarrow (V_{\text{open}} \cup V_{\text{open,new}}) \setminus \{z\}$ 
24:    $V_{\text{closed}} \leftarrow V_{\text{closed}} \cup \{z\}$ 
25:   if  $V_{\text{open}} = \emptyset$  then
26:     return Failure
27:   end if
28:    $z \leftarrow \arg \min_{x \in V_{\text{open}}} \{G(x)\}$ 
29: end while
30: Output: Path( $z, T = (V_{\text{open}} \cup V_{\text{closed}}, E)$ )
```

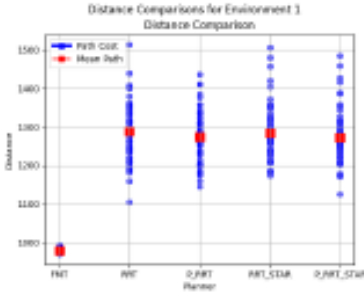
2.1.3.2 Comparison of FMT* with other planners

In this section we present our work on the comparison of FMT* in contrast to various environments. Our comparison planners are the RRT, RRT*, p-RRT* and the PRM* planner. Our results show the improvement with respect to path cost and time of FMT*. Fig 5 (a) to (i) show the outcome of our results. Our results show that the FMT* planner out-performs the RRT by 75 % , 70 % RRT* , 69 % P-RRT, 65 % P-RRT* with respect to time and distance.

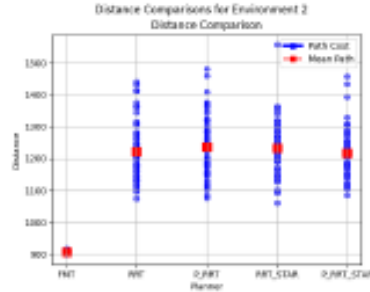
2.1.3.3 Recent work on FMT*

The Anytime FMT* (aFMT*) [37] introduces a hybrid sampling, and a region identification in order to present a solution to the narrow corridor problem. It uses a three-sampling distribution uniform sampling, Gaussian sampling, and bridge sampling over the configuration space, the classification of different regions, and then concentration of sampling over the 'difficult' regions, where it is most likely to achieve a breakthrough. The bridge test is a method of evaluating the feasibility of a sample. A sample passes the bridge test if it is on a line [38] segment such that the two end of the line segment, lie on obstacles. The aFMT* also adds another modification to the lazy optimal local one step connection strategy. If a node has been added once into the tree as an optimal node, then it cannot be added again. The aFMT* first of all takes a hybrid sample over the configuration space. It then establishes buRatio between bridge samples and uniform samples, and a guRatio between the gaussian and the uniform samples. The region construction is done on the basis of a sample type (Gaussian Samples are given priority) and a certain radius. Depending on the buRatio it will also adjust the number and type of samples. New samples are then added to the difficult regions and then passed on to the FMT*. The Informed Anytime FMT* Algorithm [9], introduces a hyperelliptic subregion for directed sampling, reminiscent of the BIT* [8] or Informed RRT* [29] algorithm. In addition to this it also introduces a Rewiring procedure to extend the tree based on non-lazy evaluation of the state space.

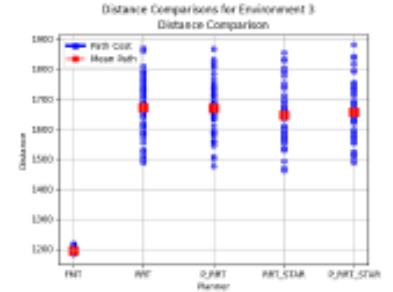
The Group Marching Tree (GMT) [39] , uses the parallel computing power of multiple GPUs in order to expand multiple nodes at



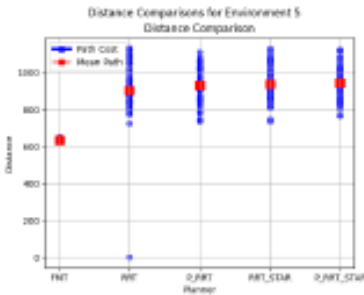
(a) $FMT^*=978.18$,
 $RRT=1288.20$, $RRT^*=1284.56$,
 $P-RRT=1274.72$, $P-RRT^*=1272.82$



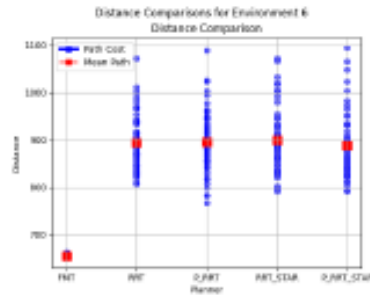
(b) $FMT^*=906.64$,
 $RRT=1221.79$, $RRT^*=1236.09$,
 $P-RRT=1232.84$, $P-RRT^*=1217.34$



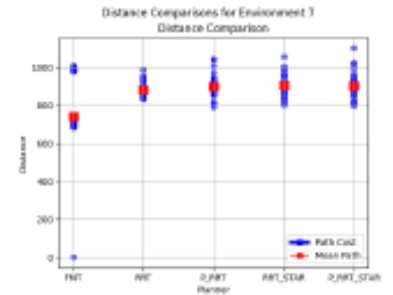
(c) $FMT^*=1194.05$,
 $RRT=1670.41$, $RRT^*=1669.82$,
 $P-RRT=1646.97$, $P-RRT^*=1656.63$



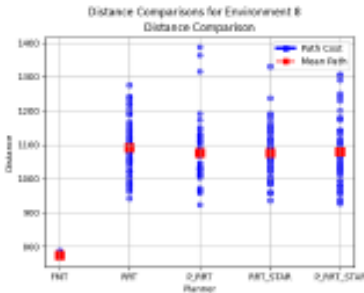
(d) $FMT^*=636.43$, $RRT=906.35$,
 $RRT^*=933.12$, $P-RRT=938.73$,
 $P-RRT^*=946.37$



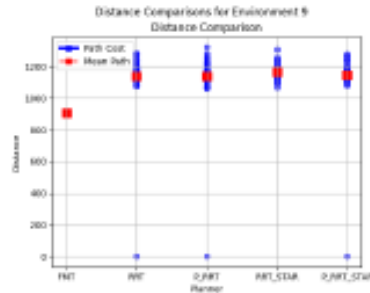
(e) $FMT^*=655.40$, $RRT=894.71$,
 $RRT^*=895.05$, $P-RRT=899.88$,
 $P-RRT^*=887.69$



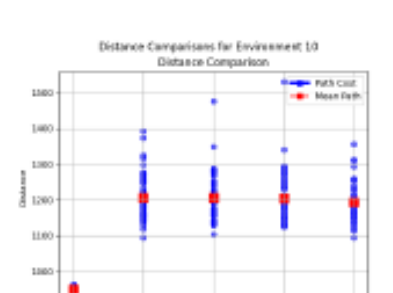
(f) $FMT^*=740.47$, $RRT=881.76$,
 $RRT^*=900.07$, $P-RRT=906.37$,
 $P-RRT^*=902.65$



(g) $FMT^*=774.26$,
 $RRT=1091.00$, $RRT^*=1074.88$,
 $P-RRT=1074.70$, $P-RRT^*=1078.87$

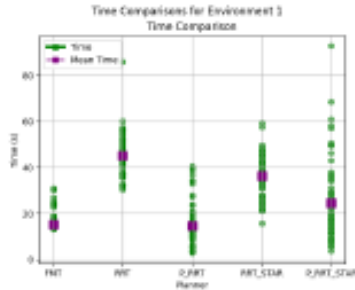


(h) $FMT^*=905.14$,
 $RRT=1139.58$, $RRT^*=1140.73$,
 $P-RRT=1166.36$, $P-RRT^*=1146.87$

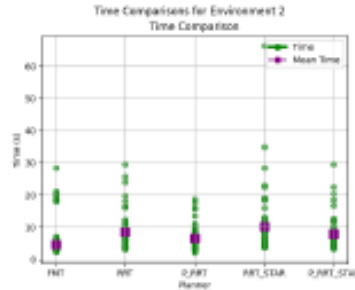


(i) $FMT^*=949.06$, $RRT=1205.98$,
 $RRT^*=1205.26$, $P-RRT=1204.83$,
 $P-RRT^*=1193.00$

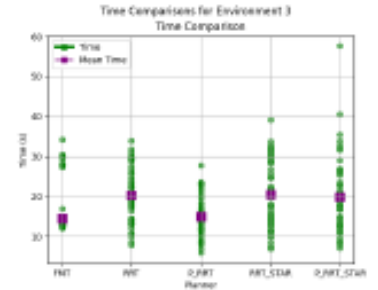
Figure 5: Distance Comparison of FMT^* , RRT , $P-RRT$, RRT^* , $P-RRT^*$.



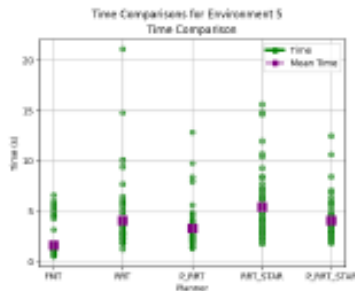
(a) $FMT^*=15.0$, $RRT=45.09$, $RRT^*=36.24$, $P-RRT=14.65$, $P-RRT^*=24.65$



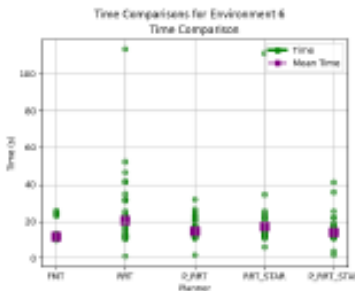
(b) $FMT^*=4.67$, $RRT=8.60$, $RRT^*=6.57$, $P-RRT=10.35$, $P-RRT^*=8.03$



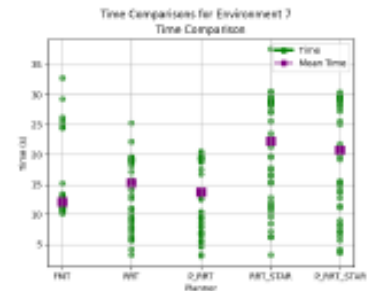
(c) $FMT^*=14.55$, $RRT=20.44$, $RRT^*=15.06$, $P-RRT=20.58$, $P-RRT^*=19.90$



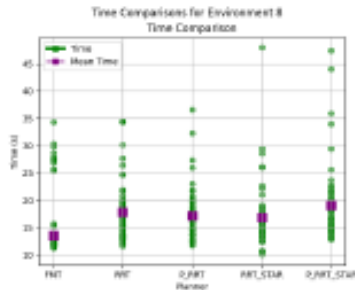
(d) $FMT^*=1.68$, $RRT=4.15$, $RRT^*=3.34$, $P-RRT=5.48$, $P-RRT^*=4.12$



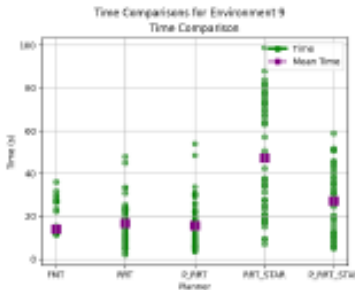
(e) $FMT^*=11.70$, $RRT=20.501$, $RRT^*=14.90$, $P-RRT=17.08$, $P-RRT^*=14.22$



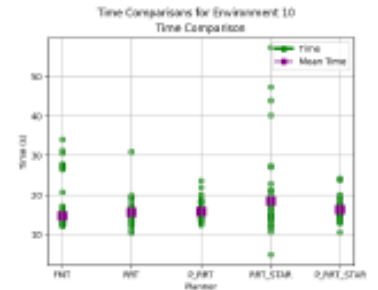
(f) $FMT^*=12.09$, $RRT=15.25$, $RRT^*=13.73$, $P-RRT=22.34$, $P-RRT^*=20.80$



(g) $FMT^*=13.62$, $RRT=17.89$, $RRT^*=17.16$, $P-RRT=16.92$, $P-RRT^*=19.18$



(h) $FMT^*=14.19$, $RRT=16.74$, $RRT^*=15.69$, $P-RRT=47.23$, $P-RRT^*=27.19$



(i) $FMT^*=14.67$, $RRT=15.59$, $RRT^*=15.85$, $P-RRT=18.42$, $P-RRT^*=16.34$

Figure 6: Time Comparison of FMT^* , RRT , $P-RRT$, RRT^* , $P-RRT^*$.

the same time. Such an approach however is computationally expensive and dependent upon the number of GPUs available. The Online FMT* [40] introduces an online sampling, and an online rewiring and pruning strategy. It uses a threshold value, in order to limit the number of nodes. It also uses the current position of the robot as new root of the robot. It also continues to sample but new nodes are instead not added to the tree. Using a new sample as a center, a nearest neighbor search is then done, in order to get the lowest cost node. It then adopts a rewiring strategy reminiscent of RRT*, in order to get a low-cost path, however it also updates the cost function.

A Bidirectional variant of the Fast-Marching Tree Algorithm (BFMT*) [41], involves two different trees, one expanded from the starting position, and another from the goal position. Apart from the traditional alternate bidirectional tree expansion strategy, a balanced tree criteria is introduced, in which the frontier node with minimum cost node is expanded from the frontier of both sides. In addition, two termination conditions are also introduced. Apart from the ‘First Path Criteria’, when there the two trees have connected, the other is when the node selected, is a node in the interior of the other tree. The Heuristic Bidirectional Fast Marching Tree [42], simply introduces a basic heuristic to the BFMT*, speeding up the search process considerably

The Hierarchical Bidirectional FMT* [43], is a bidirectional implementation of the FMT* algorithm, on a re-configurable mobile robot platform. It initially implements the bidirectional FMT*, on a 3DOF space, before using a hybrid sampling strategy on the full 8 DOF state space. The hybrid sampling strategy involves a uniform sampling, and a Gaussian concentrated sampling based on the initial path computed. Another hardware based implementation algorithm, is the Dual Tree FMT*(DTFMT*) Algorithm [44], It involves a search over the self-motion manifold, and a validity checking of nodes, in order to ensure valid motion.

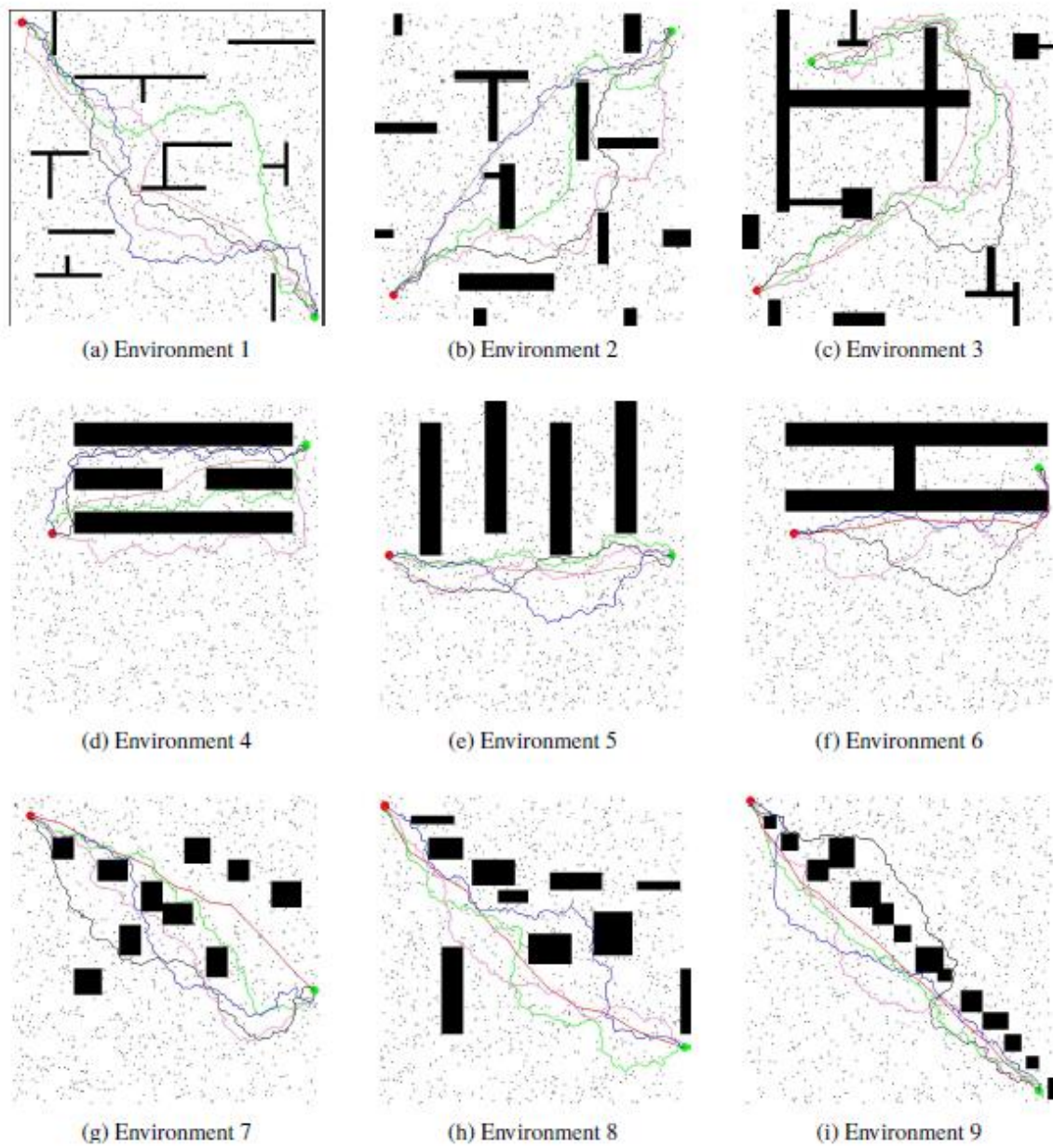


Figure 7: Comparison of FMT* with RRT (black), RRT* (blue), P-RRT* (pink), P-RRT

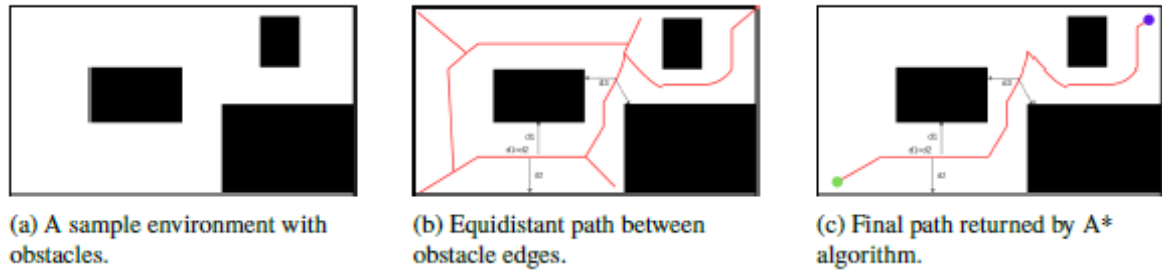


Figure 8: A rendering of Generalized Voronoi Graph with multiple configurations

2.1.3.4 Limitations of FMT* Planner

However, FMT* has some limitation. Firstly, when working with a reduced set of samples the reduction in path quality is drastic. This is demonstrated in Fig 9 (a). Additionally, the planner may struggle to explore the entire space. Secondly, with an increase in the number of samples, there is a tendency to explore useless region which may reduce the speed of convergence. Additionally, the increase in no of samples may also increase the time taken as shown in Fig 9 (b). In order to address these limitation we have devised and implemented a novel approach which meliorates the quality of path alongside the speed of convergence.

2.1.3.5 Secure Tunnel FMT*

Wu *et al.* gave the Secure Tunnel FMT* Algorithm in order to address some of the limitations of FMT*. The algorithm works by first using the Generalized Voronoi Graph (GVG) [45] method to get a well-connected roadmap. This process is discussed in

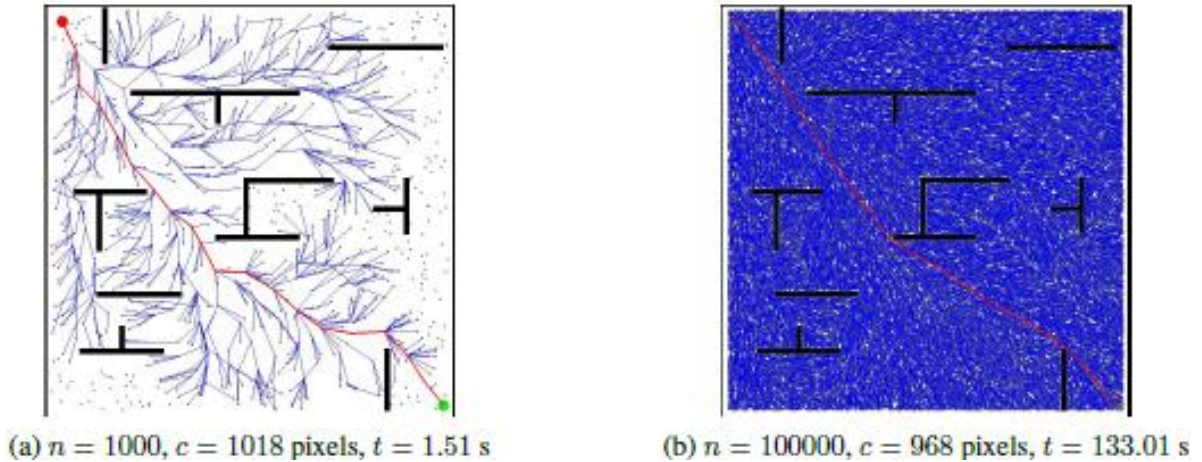


Figure 9: FMT* performance with an increasing number of samples. Here, n is the number

Algorithm 5. It then quickly finds an initial path using the A* algorithm. The GVG planner

Algorithm 7 Generalized Voronoi Graph (GVG) Planner

- 1: **Input:** Start position x_{start} , goal position x_{goal} , map with obstacles O
 - 2: **Output:** Path from x_{start} to x_{goal}
 - 3: Generate the Voronoi graph G from the map using the locations of the obstacles O
 - 4: Find the closest vertex v_{start} on the graph to x_{start}
 - 5: Find the closest vertex v_{goal} on the graph to x_{goal}
 - 6: Compute a path on G from v_{start} to v_{goal} using a graph search algorithm (e.g., A* or Dijkstra)
 - 7: **if** a path exists **then**
 - 8: Follow the path on the Voronoi graph G
 - 9: Refine the path by smoothing or adjusting for robot constraints
 - 10: **else**
 - 11: **Return:** No path found
 - 12: **end if**
 - 13: **Return:** The refined path from x_{start} to x_{goal}
-

works by generating a Voronoi diagram whose edges are equidistant to obstacle space. These edges form the roadmap and we insert the start and goal positions into the roadmap. Then another graph search algorithm, most commonly A* or Dijkstra, is used in order to find the minimum cost path. Figure 8 and Figure 9 show the type of graph created. The path returned is such that it maintains an equal distance from any obstacles at any point. It also has a rough approximation of the general direction of the optimal path.

Therefore, in order to maximize exploitation whilst having a reasonable estimate of the exploration, the path returned is used to create a 'secure tunnel' in order to find an optimal and safe motion path. The initial path generated by ST-FMT* is then discretized using a discretization factor f and to generate a set of points q . Each point is at the center of a circle

$$S = \{Cir(x_{dis,m}, D_r(x_m))\}_{m=0}^q \quad (2.4)$$

The radius of each circle $D_r(x_m)$ is given as the minimum distance to an obstacle. The coverage area of the secure tunnel is dictated by the discretization factor f , which controls the resolution of the initial path. This parameter essentially determines how finely the path is segmented, directly impacting the size and scope of the tunnel created around the path. Once this secure tunnel is defined, a uniform sampling strategy is employed within its bounds. Importantly, as discussed in [54], collision detection during the sampling phase inside the tunnel can be omitted. This omission significantly reduces computational overhead, as collision checking is often a resource-intensive step. By skipping this process in the secure tunnel, the ST-FMT* algorithm is able to enhance its computational efficiency, leading to faster overall performance. The effectiveness of this approach is largely due to the dense distribution of samples within the secure tunnel, particularly around the initial path generated by the Generalized Voronoi Graph (GVG) method. These densely distributed samples allow the Fast Marching Tree (FMT*) algorithm to converge more quickly to an optimal solution, as the samples are focused on a specific region of interest rather than being spread uniformly across the entire environment. This localized sampling strategy contrasts with global uniform sampling approaches, where the distribution of samples is more widespread, potentially leading to inefficiencies and slower convergence times.

In the secure tunnel, however, even with a lower density of samples compared to global sampling, high-quality solutions can be found due to the proximity of the samples to the optimal path. The secure tunnel thus concentrates the sample distribution in areas that are most relevant to the motion planning task, accelerating the convergence of the FMT* algorithm. By focusing computational resources on the most important regions, the algorithm is able to find an optimal path more efficiently than through traditional methods.

Moreover, the combined use of secure tunnel construction and focused sampling not only improves the performance of ST-FMT*, but also has broader applications. This combination can be employed as an independent preprocessing technique in a variety of sampling-based motion planning algorithms. For example, algorithms like Rapidly-exploring Random Trees (RRT) or Probabilistic Roadmaps (PRM) could benefit from this preprocessing step, where the secure tunnel is first established, and then sampling is conducted within this confined region. This approach could reduce computation time, improve solution quality, and provide a framework for more efficient motion planning across diverse robotic systems or autonomous navigation tasks. Thus, the secure tunnel framework represents a powerful tool in motion planning, not only for the ST-FMT* algorithm but also for its potential integration into other sampling-based approaches. Its ability to focus sample distribution and reduce collision checking makes it a highly efficient and effective strategy for generating optimal paths in complex environments.

2.1.3.6 Limitations of ST-FMT*

However, one of the setbacks of both ST-FMT* and OB-FMT* is the requirement of initial preprocessing of the environment in order to define a sampling zone which is computationally expensive. In addition, Hou *et al.* [46] shows that the increase in samples numbers and density has an adverse effect on the time required to converge to a solution. As seen in Fig 10 (b) and (c), the path discretization method used in ST-FMT* gives no guarantee of contiguous samples. Additionally, these drawback extend to the roadmap planning used in [46]. Similarly, as shown in Fig 10 (a), the initial solution by roadmap may not necessarily lie within the bounds of our environment.

Algorithm 8 Secure Tunnel Fast Marching Tree Algorithm (ST-FMT*)

```
1: Input:  $x_{\text{start}}, x_{\text{goal}}, MAP$ 
2: Output:  $\text{Path}(z, T = (V_{\text{open}} \cup V_{\text{closed}}, E))$ 
3:  $\text{RoadMap} \leftarrow \text{GVG}(MAP)$ 
4:  $\text{InitialPath} \leftarrow \mathbf{A}^*(\text{RoadMap})$ 
5:  $\text{SecureTunnel} \leftarrow \text{SafetyAreaBuild}(\text{InitialPath})$ 
6:  $\text{SampleDot} \leftarrow \text{Sampling}(\text{SecureTunnel})$ 
7:  $V \leftarrow \{x_{\text{start}}, x_{\text{goal}}\} \cup \text{SampleDot}$ 
8:  $E \leftarrow \emptyset$ 
9:  $V_{\text{unvisited}} \leftarrow V \setminus \{x_{\text{start}}\}$ 
10:  $V_{\text{open}} \leftarrow \{x_{\text{start}}\}$ 
11:  $V_{\text{closed}} \leftarrow \emptyset$ 
12:  $z \leftarrow x_{\text{start}}$ 
13: while  $z \neq x_{\text{goal}}$  do
14:    $X_{\text{near}} \leftarrow \text{Near}(V_{\text{unvisited}}, z, r_n)$ 
15:    $V_{\text{open,new}} \leftarrow \emptyset$ 
16:   for  $x \in X_{\text{near}}$  do
17:      $Y_{\text{near}} \leftarrow \text{Near}(V_{\text{open}}, x, r_n)$ 
18:      $y_{\text{min}} \leftarrow \arg \min_{y \in Y_{\text{near}}} \{c(y) + \text{Cost}(y, x)\}$ 
19:     if  $\text{CollisionFree}(y_{\text{min}}, x)$  then
20:        $G(x) \leftarrow G(y_{\text{min}}) + \text{Cost}(y_{\text{min}}, x)$ 
21:        $F(x) \leftarrow G(x) + H(x)$ 
22:        $E \leftarrow E \cup \{(y_{\text{min}}, x)\}$ 
23:        $V_{\text{open,new}} \leftarrow V_{\text{open,new}} \cup \{x\}$ 
24:        $V_{\text{unvisited}} \leftarrow V_{\text{unvisited}} \setminus \{x\}$ 
25:     end if
26:   end for
27:    $V_{\text{open}} \leftarrow (V_{\text{open}} \cup V_{\text{open,new}}) \setminus \{z\}$ 
28:    $V_{\text{closed}} \leftarrow V_{\text{closed}} \cup \{z\}$ 
29:   if  $V_{\text{open}} = \emptyset$  then
30:     return Failure
31:   end if
32:    $z \leftarrow \arg \min_{x \in V_{\text{open}}} \{F(x)\}$ 
33: end while
34: return Path
```

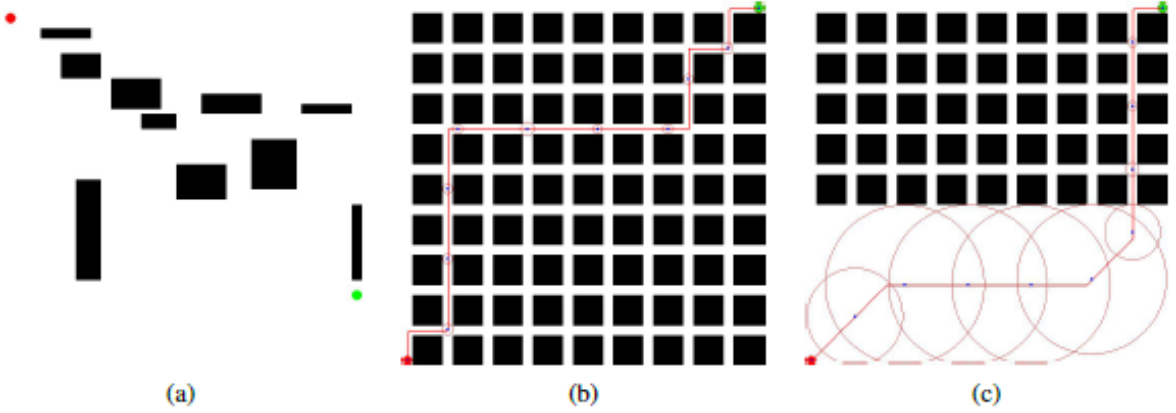


Figure 10: Generalized Voronoi Graph (GVG) planner-based path discretization with secure tunnel construction using path discretization parameter $f=10$. (a) GVG failed to return path in given environment boundaries. (b) GVG returning less ideal result (c) Non contiguous tunnel construction

To address these problem, in this paper we introduce Double-FMT* planner. Inspired by STFMT*, we replace the computationally costly environment decomposition required for the GVG planner with an initial path generated by a reduced no of samples. In addition, we modify the construction of the tunnel to ensure a minimum overlap between the constituent circles of the tunnel. By concentrating a limited batch of samples within the tunnel we can get an improved path with respect to path distance and time. A second planner working within the new samples over the tunnel generates an improved path

2.1.3.6 Artificial Potential Fields

The Artificial Potential Field (APF) algorithm, developed in 1985 by Khatib [47] , has seen a wide variety of use in motion planning [48] [49] [50] [51] [52]. The basic idea, behind APF field is simple. The robot is modelled, as a particle, under the effect of attractive (from goal), and repulsive (from obstacles) potentials. There have been many adaptations of APF and RRT family of path planning algorithms. The Adaptive Potential Guided Directional RRT(APGDRRT), developed by Qureshi [53], is an extension of the earlier PGD-RRT [26]. The basis of APGD is on the basis of computing a random sample using a Randomized Gradient Descent (RGD). It is similar to the Gradient Descent, however unlike the classic version, the next state is not dependent on the previous state. The random

sample, is moved iteratively along the direction of the potential field. APGD-RRT, accelerates the rate of convergence by employing a directional sampling strategy. Building upon this, PIB-RRT* and PB-RRT* [54] introduces the Bi-directional Potential Gradient (BPG), a variation on the original equation, for bi directional search, in cluttered environment. Xinyu [55], introduces a variant of the P-RRT*, called P-RRT*connect, which introduces a switching to classic RRT*, when the robot encounters a local minimum.

2.3.1 Dynamic Motion Planning

Dynamic Environments introduce, a time constraint to the problem of motion planning [56]. In offline implementation of motion planners, dynamic obstacles, can cause the invalidity of computed paths. The dynamic re-planner, is responsible for the modification, and optimization of a given trajectory, in response to the presence of unexpected obstacles. Putatively, there are two broad categorization of algorithm, Reactive algorithms, consider only the current condition of the environment, including the number and position of obstacles, in order to adjust its trajectory. Combining, local and global planning Otte *et al.* [57], introduced the RRTX algorithm, which continually refines an initial trajectory and repairs it in case of obstacles. Time Based-RRT(TB-RRT) [58], pairs each node with a time stamp, and introduces a time constraint, with the goal of reaching the goal position in given time.

Active algorithm assumes knowledge of obstacle trajectories. Incorporating some aspects of Time based RRT, Risk based RRT, [59] model the probability of Collision for dynamic obstacles, as Gaussian Mixture Models (GMM). The search for a feasible path, is guided by the probabilistic risk of collision, and the time stamp of the particular node. The Risk guided search uses a initial reference, based on the classic RRT, computes the probability of collision for both static and dynamic obstacles. It then updates the existing nodes, and updates the weight assigned to each node, with the collision probability and length apparent of path. It then grows the tree in that direction. However, the risk based RRT requires a separate algorithm to track obstacles. Much work has been done, in recent years on the Risk base RRT. The Risk Dual Tree RRT (Risk DTRRT) [59], introduces a dual tree, in order to save the original heuristic trajectory generated by the robot. A Line of Sight (LOS)

algorithm is also used in order for checking the feasibility of a given trajectory bearing in mind the motion constraints on the robot and the presence of obstacles. Pruning is done based on this, hence the need for a second tree, saving original data, in order for rewiring.

Based on a planning-replanning paradigm, the Multi Objective Dynamic RRT* Algorithm (MOD-RRT*) [60], uses a backward expansion from the goal position. The heuristic trajectory generated, is further optimized, using an Ant Colony Optimization, which will be discussed later. For the replanning phase, it involves a implementation of Praeto Dominance comparing length of the path, and the turning angel of each node. Another formulation, based on the dynamic replanning, is the Elastic Band RRT(EBRRT) [61]. It uses Elastic Band theory developed by Quinlan *et al.* [62], which models the trajectory as an elastic band under contractive and repulsive forces. An initial trajectory is generated and modified by the re-planner, based on real time data. The Horizon Based Lazy Optimal RRT(HL-RRT) [63], after generating an initial trajectory, it collects an elite set of nodes. Based on a Gaussian Mixture Model (GMM), it then generates a set of new samples based on the parameter of an Expectation Minimization algorithm, that best fits the GMM. It also performs a lazy collision strategy, by keeping a model of the future using model predictive control and preforming collision detection on a partial portion.

CHAPTER 3: METHODOLOGY

As discussed previously in Chapter 2, the main idea of FMT* is to reduce the number of collision checks by the lazy extension procedure; however the laziness property of FMT* can lead to sub-optimality. But, it has been proved that as the number of sub optimal connections become rare, as number of samples goes to infinity. [7] The AO of the algorithm is also proved to converge in probability, a mathematically weaker notation.

One of the optimization direction we have identified is the need for a optimizing and improving existing paths. Similar to different members of the RRT family of algorithms the optimization methodology is a sampling strategy [9] [29] [40] [53]. The tunnel, strategy is applied in order to take advantage of a rapidly computed initial path, as biasing the sampling based on an existing path [43]. A Hybrid Sampling involving the use of Uniform Sampling with Gaussian Sampling, has also been used [37]. Rewiring and Single Query Reconnecting

In Chapter 2 we have proved that there is an inverse relationship between the number of samples and time taken for the planner to return a solution to the problem. Similarly, earlier work on the RT-FMT* has further highlighted this problem because as the number of samples and sampling density increase there is an increase in the time taken to compute a solution. This can have a negative solution in the case of dynamic obstacles where fast computation is necessary in order to compute a fast path to the obstacle.

3.1 Dual Tree Fast Marching Tree (DT-FMT*)

In this section we present the DT-FMT* whose detailed pseudocode is given in Algorithm 9 and 10. Fig 11 gives a brief overview of DT-FMT* in action.

3.1.1 Working Principle

Our entire approach is based on two-stage batch sampling of the environment. For a total number of samples n_{total} , we subdivide the samples into two batches. During the preprocessing stage we use the reduced amount of samples to compute a rough initial path based on those values. One of the advantages of sampling based planners is that they provide a much more computationally efficient solution [21] and we utilize this by using the standard FMT* algorithm to construct an initial path. Doing so allows us to get an approximate direction of the optimal path. After discretizing the initial path into a set of equidistant points based on our path discretizing factor f , we construct a tunnel to encompass the entire region where we believe the optimal path is likely to lie. Then, we generate $\frac{N_{total}}{2}$ new samples in the above mentioned region and utilize the FMT* planner to find a new path.

Algorithm 9 Double Tree Fast Marching Tree Algorithm (DT-FMT*)

```
1: Input:  $x_{\text{start}}, x_{\text{goal}}, N, \text{environment}$ 
2: Output:  $\text{Path}(z, T = (V_{\text{open}} \cup V_{\text{closed}}, E))$ 
3:  $\text{InitialPath} \leftarrow \text{FMT}^*(x_{\text{start}}, x_{\text{goal}}, N//2, \text{environment})$ 
4:  $\text{Tunnel} \leftarrow \text{ComputeTunnel}(\text{InitialPath})$ 
5:  $\text{TunnelSamples} \leftarrow \text{Sampling}(\text{Tunnel})$ 
6:  $V \leftarrow \{x_{\text{start}}, x_{\text{goal}}\} \cup \text{TunnelSamples}$ 
7:  $E \leftarrow \emptyset$ 
8:  $V_{\text{unvisited}} \leftarrow V \setminus \{x_{\text{start}}\}$ 
9:  $V_{\text{open}} \leftarrow \{x_{\text{start}}\}$ 
10:  $V_{\text{closed}} \leftarrow \emptyset$ 
11:  $z \leftarrow x_{\text{start}}$ 
12: while  $z \neq x_{\text{goal}}$  do
13:    $X_{\text{near}} \leftarrow \text{Near}(V_{\text{unvisited}}, z, r_n)$ 
14:    $V_{\text{open,new}} \leftarrow \emptyset$ 
15:   for  $x \in X_{\text{near}}$  do
16:      $Y_{\text{near}} \leftarrow \text{Near}(V_{\text{open}}, x, r_n)$ 
17:      $y_{\text{min}} \leftarrow \arg \min_{y \in Y_{\text{near}}} \{c(y) + \text{Cost}(y, x)\}$ 
18:     if  $\text{Collision}(y_{\text{min}}, x) == \text{False}$  then
19:        $G(x) \leftarrow G(y_{\text{min}}) + \text{Cost}(y_{\text{min}}, x)$ 
20:        $E \leftarrow E \cup \{(y_{\text{min}}, x)\}$ 
21:        $V_{\text{open,new}} \leftarrow V_{\text{open,new}} \cup \{x\}$ 
22:        $V_{\text{unvisited}} \leftarrow V_{\text{unvisited}} \setminus \{x\}$ 
23:     end if
24:   end for
25:    $V_{\text{open}} \leftarrow (V_{\text{open}} \cup V_{\text{open,new}}) \setminus \{z\}$ 
26:    $V_{\text{closed}} \leftarrow V_{\text{closed}} \cup \{z\}$ 
27:   if  $V_{\text{open}} = \emptyset$  then
28:     return Failure
29:   end if
30:    $z \leftarrow \arg \min_{x \in V_{\text{open}}} \{F(x)\}$ 
31: end while
32: return Path
```

The tunnel effectively limits the search space to a smaller, more relevant portion of the environment hence improving path quality. These tunnel samples help refine the path by introducing more candidate waypoints, ensuring that the final path is both feasible and efficient. By focusing the sampling within the tunnel, we increase our chances of finding an optimal or near-optimal solution while avoiding unnecessary exploration of distant areas.

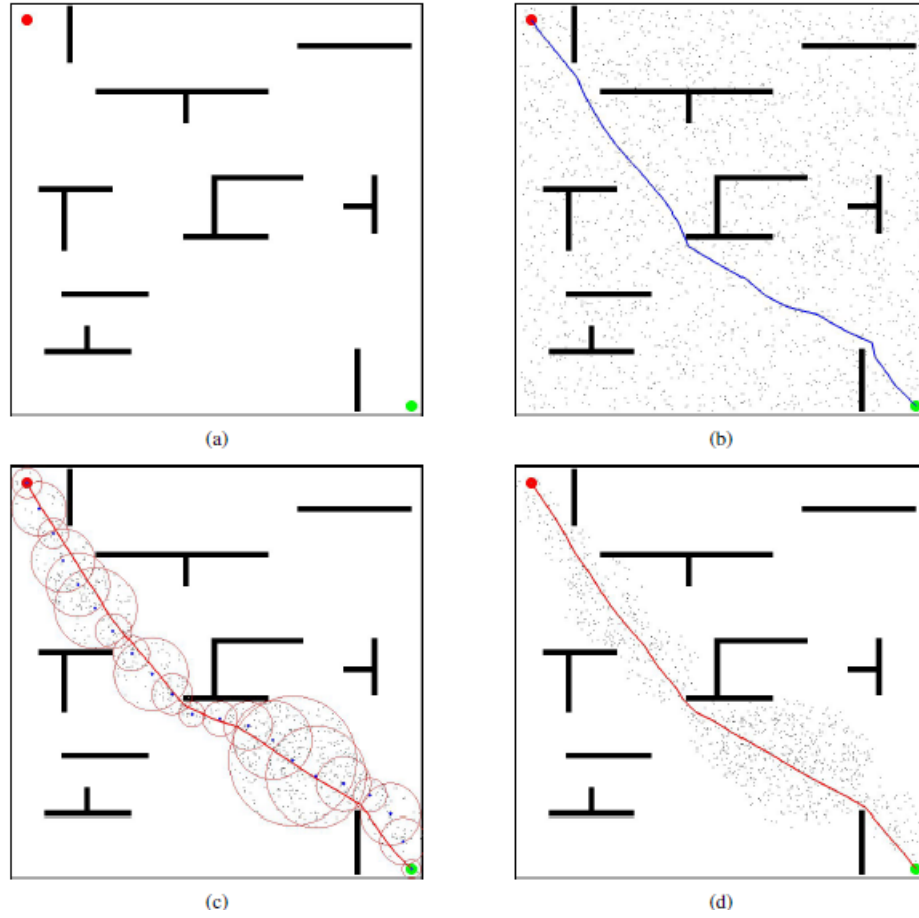


Figure 11: An overview of the DT-FMT* algorithm. (a) A given environment map. (b) Initial path computed with $n = 1000$. (c) Initial path discretized with $f = 10$ and overlap = 10, new samples $n = 1000$. (d) New path computed $c = 971.24$, $t = 1.422$.

3.1.2 Tunnel Construction

The details of tunnel construction are given in Algorithm 4. For an initial path given by P , we discretize the path to get a series of equidistant points:

$$x_{disc,1}, x_{disc,2}, x_{disc,4}, \dots, x_{disc,i}$$

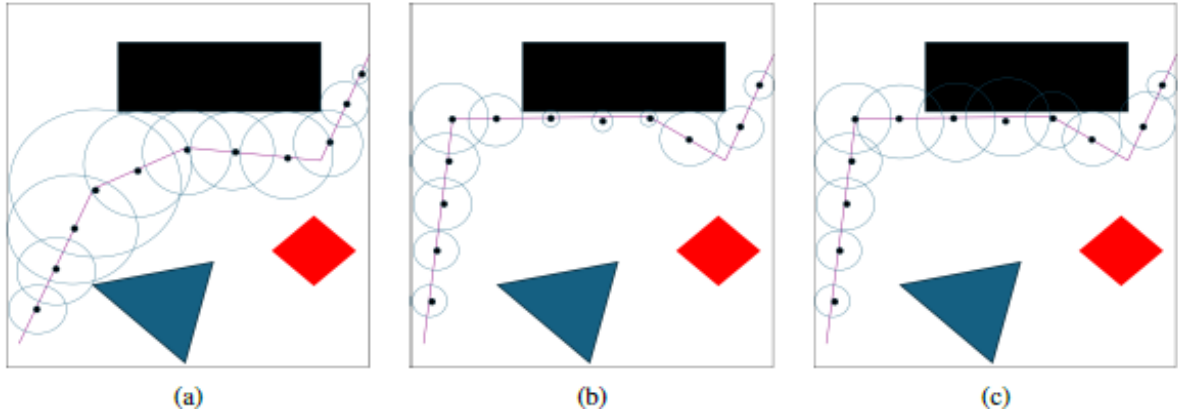


Figure 12: Tunnel construction methodology for DT-FMT* (a) An idealized case when path discretized points and radius construction ensure contiguous samples. (b) A less ideal case when path discretization is not enough to ensure contiguous samples. (c) M

where i is the total number of points. To compute this, we use the discretization parameter f , given as $[1, 5, 10, 20, \dots]\%$. There is an inverse relation between f and computational efficiency. After getting the path points, Algorithm 3, line 7 computes the nearest distance to an obstacle in order to get an initial radius $D_r(x_i)$ based on the distance to the nearest obstacle.

Algorithm 10 ComputeTunnel

```
1: Input: point_list
2: Output: modified_radius_list
3: point_list  $\leftarrow$  point_list  $\cup$  {start, end}
4: radius_list  $\leftarrow$   $\emptyset$ 
5: modified_radius_list  $\leftarrow$  Copy(radius_list)
6: for each point  $\in$  point_list do
7:   radius  $\leftarrow$  Dist2Obstacle(point)
8:   radius_list  $\leftarrow$  radius_list  $\cup$  {radius}
9: end for
10: for  $i \leftarrow 0$  to length(radius_list) - 2 do
11:   point  $\leftarrow$  point_list[ $i$ ]
12:   radius  $\leftarrow$  radius_list[ $i$ ]
13:   next_point  $\leftarrow$  point_list[ $i + 1$ ]
14:   next_point_radius  $\leftarrow$  radius_list[ $i + 1$ ]
15:   if IsOverlap(point,) then
16:     new_radius  $\leftarrow$ 
       ModifyRadius(radius, next_point_radius)
17:     modified_radius_list  $\leftarrow$ 
       Insert(modified_radius_list,  $i$ , new_radius)
18:   end if
19: end for
20: return modified_radius_list
```

As shown in Fig 12, the use of an initial path based on FMT* can create a scenario where, in case of path abutting an obstacle, the value of $D_r(x_i)$ to be such that it does not guarantee a contiguous tunnel region. Additionally, such 'obstacle hugging' behaviour of a path, while representative of a near-optimal path in many situations e.g in a cluttered environment, narrow corridor e.t.c, means that in a situation where the path point $x_{disc,i}$ lies just on the edge of the obstacle in X_{free} the value of $D_r(x_i)$ would be minuscule. As we have to generate a number of samples within the points constituting the tunnel, it is essential that the samples should be adjacent to each other i.e there are no large gaps within the sampling regions that result in our planner failing to return a solution. In case of a minuscule value of $D_r(x_i)$ the path discretization factor f would need to be decreased in order to generate minimum points for contiguous samples. As shown in [64] this has the effect of increasing computational cost.

Our solution to avoid this is to incorporate an overlap factor α in order to ensure contiguous samples without increasing computational efficiency. The modified circle for each discrete point $x_{disc,i}$ with an overlap factor α is defined as:

$$S = Cir(x_{disc,i}, D_r'(x_i)) \quad (3.1)$$

With

$$D_r'(x_i) = \max\left(\frac{d(x_{disc,i}, x_{disc,i+1})}{1-\alpha}\right)$$

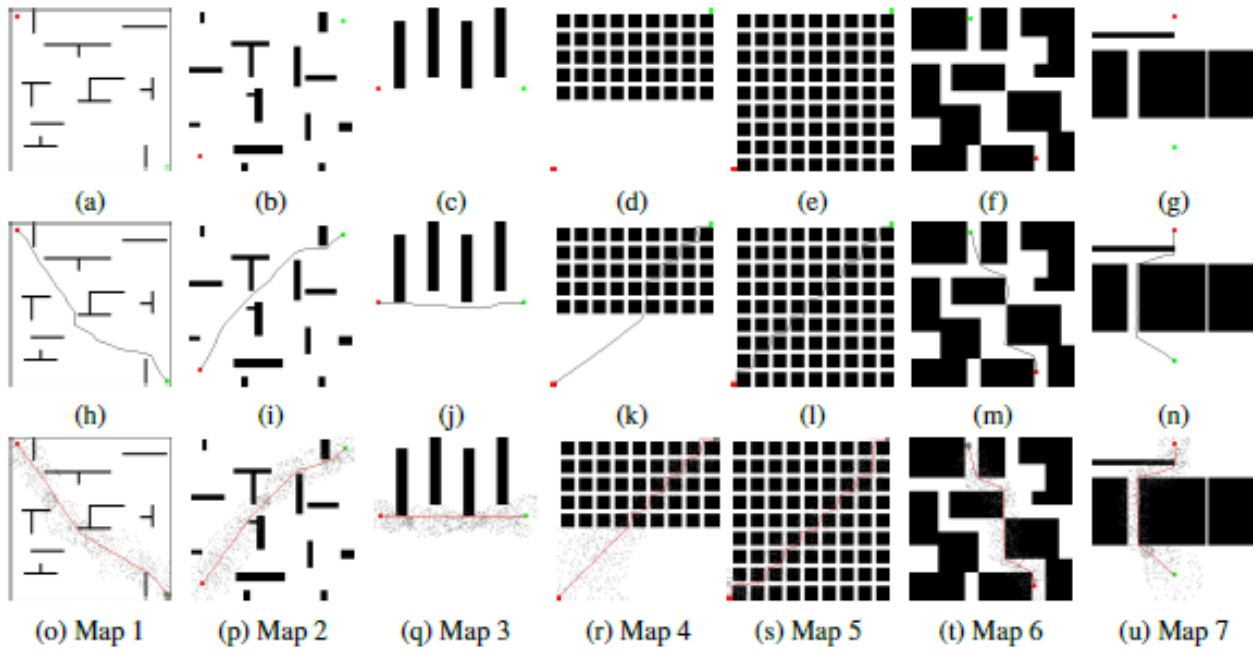


Figure 13: Rendering of the performance of DT-FMT* in the seven different environments. (a)-(g) blank environment used in the experiments. (g)-(i) is the initial path computed. (o)-(u) is final path computed.

After tunnel computation, uniform sampling is performed in order to generate the reduced batch of samples. Bialkowski et al. [57] showed that the collision checking can be omitted during the tunnel's sampling processing order to improve the computational efficiency.

The samples within the secure tunnel provide a high-quality solution even with a reduced sample set.

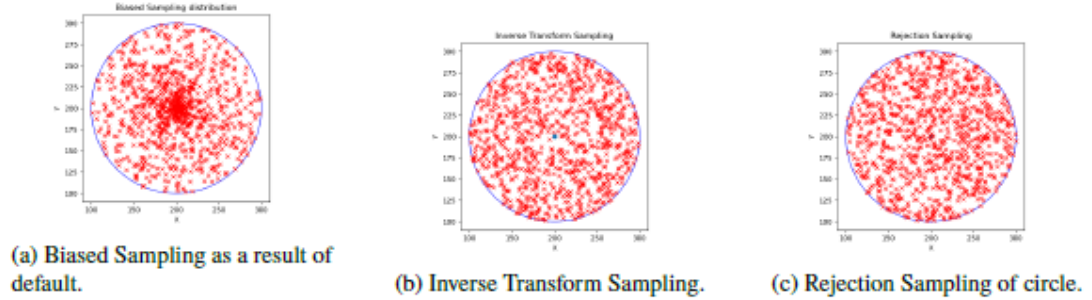


Figure 14: Different type of sampling distributions.

3.1.2 Tunnel Construction

After we have formed the centers of the circles by discretizing our path, the next step in our equation is to generate a set of new samples within the circles. One of the most common methods of sampling within a circle is to take a random value of an angle within $(0, 2\pi)$, then taking a random value of radius between $(0, R)$. However, after following this strategy, our results show it does not generate a uniform distribution across a circle. While, it has been proved by Janson *et al.* that the FMT* is valid for non-uniform sampling distributions, however, we do not want our final path to be as independent as possible from the initial path computed. Fig 14 shows the biased path generated by using this approach.

One solution to this is given as such: generate points uniformly within a square region, ranging from 0 to 1 in both radius and angle. Afterwards, we apply a transformation to adjust their distribution. This transformation maps the generated point's first coordinate to the square root of the original value multiplied by the circle's radius.

The transformation is instead given by:

$$x = \sqrt{r}\cos\theta, y = \sqrt{r}\sin\theta \quad (3.2)$$

During our tunnel construction we have utilized this when computing new samples over the tunnel that we have constructed. Another solution is rejection sampling. This works by generating points on a square enclosing the circle, and then filtering out the points that do not lie within the bounds of a circle. The details are given by Algorithm 12. This is shown in 3.4 (c) and as it shows there it also generates an unbiased sampling distribution on the circle. We have not used this version as it is computationally more expensive. In order to sample within the entire circle, firstly we choose a random circle center and also compute a random point across a unit circle. We then scale it for that particular circle

Algorithm 11 Generating a Random Point on a Circle

```

1:  $\theta \leftarrow \text{getRandomAngle}(0, 2\pi)$ 
2:  $r \leftarrow \text{getRandomRadius}(0, R)$ 
3: Set  $x \leftarrow r \cdot \cos(\theta)$ 
4: Set  $y \leftarrow r \cdot \sin(\theta)$ 
5: return  $(x, y)$ 

```

center. The full results using all three sampling paradigms is shown in Fig 15

Algorithm 12 Rejection Sampling

```

1: while True do
2:    $x \leftarrow \text{random}() \times 2 - 1$ 
3:    $y \leftarrow \text{random}() \times 2 - 1$ 
4:   if  $x \times x + y \times y < 1$  then
5:     return  $x, y$ 
6:   end if
7: end while

```

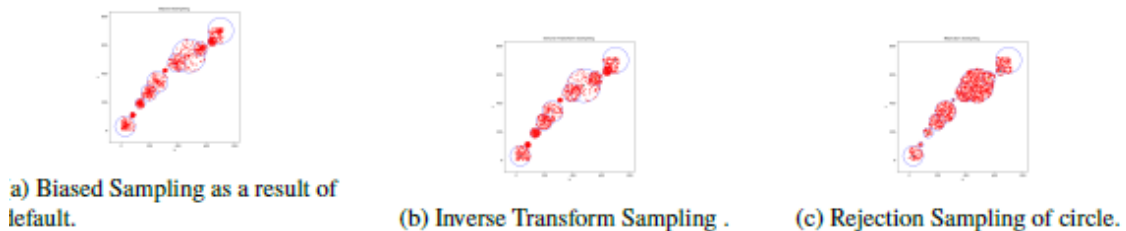


Figure 15: Different type of sampling distributions.

3.2: Reduced Sampling Re-planning Fast Marching Tree (RRFMT*)

In this section we introduce our original work for dynamic motion planning. As mentioned previously one of the problems facing one of the only extant motion planning for dynamic environment is the lack of optimality guarantees as a result of changing radius [10]. Another problem facing the FMT* Algorithm is that there is the increased computation time taken as a result of increasing samples. Contradictorily, the better path quality is only possible as a result of increased sampling density. In order to solve this problem in the presence of dynamic obstacles, hereby, in this paper defined as obstacles that appear and disappear rapidly.

The following section provides greater detail about the process behind the RR-FMT*.

3.2.1 Working Principle

Our algorithm starts by computing an initial path of the environment. In order to get the best possible path, we utilize the DT-FMT* that we introduced earlier in this chapter in order to quickly converge to a solution that is within a reasonable value of the optimal path, but which requires an egregiously lesser amount of iterations to compute. Fig 16 shows the exact procedure we have used.

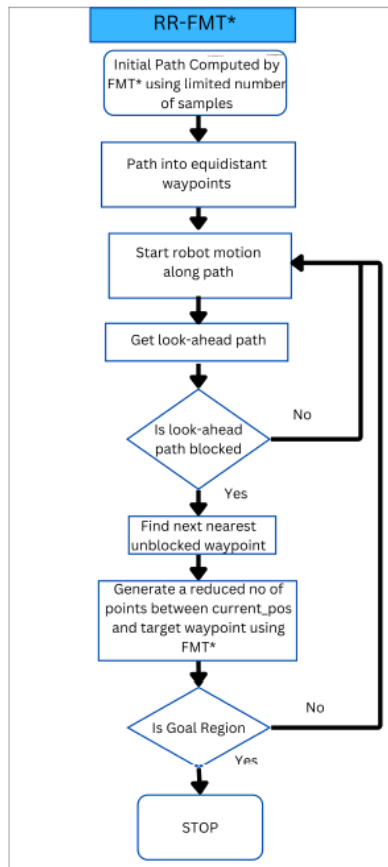


Figure 16: Procedure of Reduced Sampling Re-planning FMT*

After we generate an initial path, we, first, discretize the path in order to get a set of equidistant points on the map. Then, we start to transverse the path way-points. As the robot starts to move across the map, it is constantly beset by dynamic obstacles. In order to test the our algorithm rigorously, we have set the obstacle to be totally random with respect to time and geometry. The total number of dynamic obstacles is also set to be totally random, albeit with an upper bound of 10. This allows us to have a much better and challenging environment for our planner.

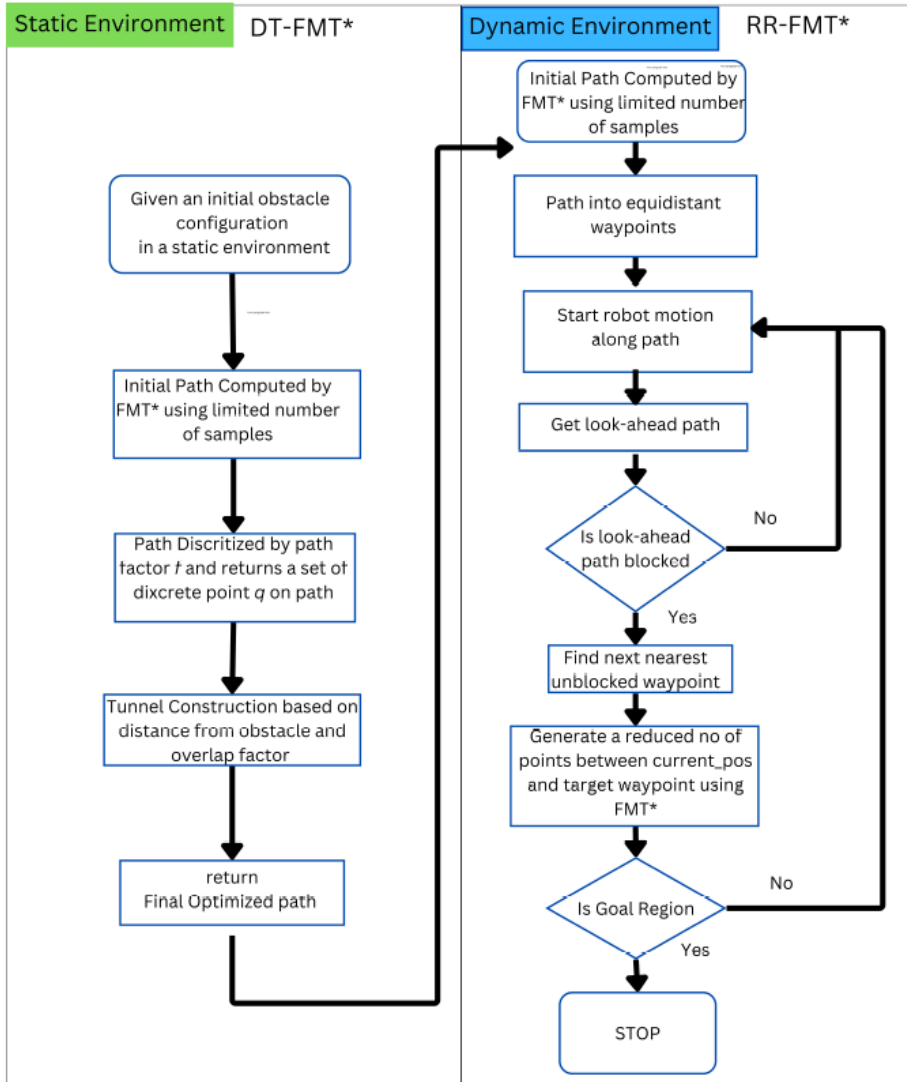


Figure 17: Complete process of Dynamic Path computation

Algorithm 13 Reduced Samples Replanning Fast Marching Tree

```
1: Input: Initial trajectory (waypoints), Map, Goal
2: Output: Replanned trajectory (updated path)
3: Trajectory  $\leftarrow$  Initial trajectory
4: LookAheadPath  $\leftarrow$  empty
5: CurrentWaypoint  $\leftarrow$  get_next_waypoint(Trajectory)
6: Goal  $\leftarrow$  read_goal()
7:  $t \leftarrow$  get_current_time()
8: while Goal not reached do
9:   observe_environment()
10:  LookAheadPath  $\leftarrow$  get_lookahead_path(CurrentWaypoint)
11:  if obstacle_detected(LookAheadPath) then
12:    Trajectory  $\leftarrow$  replan_path(CurrentWaypoint, LookAheadPath)
13:    NextWaypoint  $\leftarrow$  get_next_unblocked_waypoint()
14:    NewTrajectory  $\leftarrow$  FMT*(current_position, NextWaypoint)
15:    Trajectory  $\leftarrow$  concatenate(NewTrajectory)
16:  else
17:    move_robot_towards(CurrentWaypoint)
18:  end if
19:  if robot_reached(CurrentWaypoint) then
20:    if CurrentWaypoint == Goal then
21:      print("Goal reached")
22:      stop_robot()
23:      break
24:    else
25:      CurrentWaypoint  $\leftarrow$  get_next_waypoint(Trajectory)
26:    end if
27:  end if
28:  update_display()
29:  clock.tick(10)
30: end while
```

Algorithm 14 Reduced Samples

```
1: Input: CurrentPosition, GoalPosition, Map, ExtraBounds
2: Output: Set of sampled points between CurrentPosition and GoalPosition
3: Path  $\leftarrow$  empty
4: SampleCount  $\leftarrow$  100
5: /* Define bounds for sampling */
6: LowerBound  $\leftarrow$  CurrentPosition - ExtraBounds
7: UpperBound  $\leftarrow$  GoalPosition + ExtraBounds
8: /* Perform batch sampling within the defined region */
9: for  $i = 1$  to SampleCount do
10:  SamplePoint  $\leftarrow$  random_sample(LowerBound, UpperBound)
11:  /* No collision checking at this stage*/
12:  append(SamplePoint, Path)
13: end for
14: return Path
```

3.2.1.1 Function Definition

Algorithm 15 Supporting Functions

```
1: function GET_NEXT_WAYPOINT(Trajectory)
2:   if Trajectory is not empty then
3:     return next waypoint in Trajectory
4:   else
5:     return Goal ▷ If no more waypoints, set the goal as the target
6:   end if
7: end function
8: function GET_LOOKAHEAD_PATH(CurrentWaypoint)
9:   return path segment from current robot position to CurrentWaypoint
10: end function
11: function OBSTACLE_DETECTED(LookAheadPath)
12:   observe_environment_for_obstacles()
13:   if obstacle found on LookAheadPath then
14:     return True
15:   else
16:     return False
17:   end if
18: end function
19: function REPLAN_PATH(CurrentWaypoint, LookAheadPath)
20:   calculate a new path that avoids obstacles and leads to CurrentWaypoint
21:   return new replanned trajectory
22: end function
23: function MOVE_ROBOT_TOWARDS(CurrentWaypoint)
24:   move robot one step closer to CurrentWaypoint
25: end function
26: function ROBOT_REACHED(CurrentWaypoint)
27:   if robot position is close to CurrentWaypoint then
28:     return True
29:   else
30:     return False
31:   end if
32: end function
33: function STOP_ROBOT
34:   stop robot's movement
35: end function
36: function UPDATE_DISPLAY
37:   refresh screen and display updates
38: end function
```

RR-FMT* leverages the property of FMT* Algorithm that we have discussed earlier: as the number of samples increase so does the time required. The inverse is also true, as the number of samples decrease so too does the time required. In case of dynamic environments one key necessity is for a planner to have low latency. We solve this by firstly, decreasing the space to one in between the start point and the nearest unblocked way-point. This distance is typically very low and by distributing our samples to across the space between the goal position and the current position we ensure quick computation of the path to rejoin the optimal path.

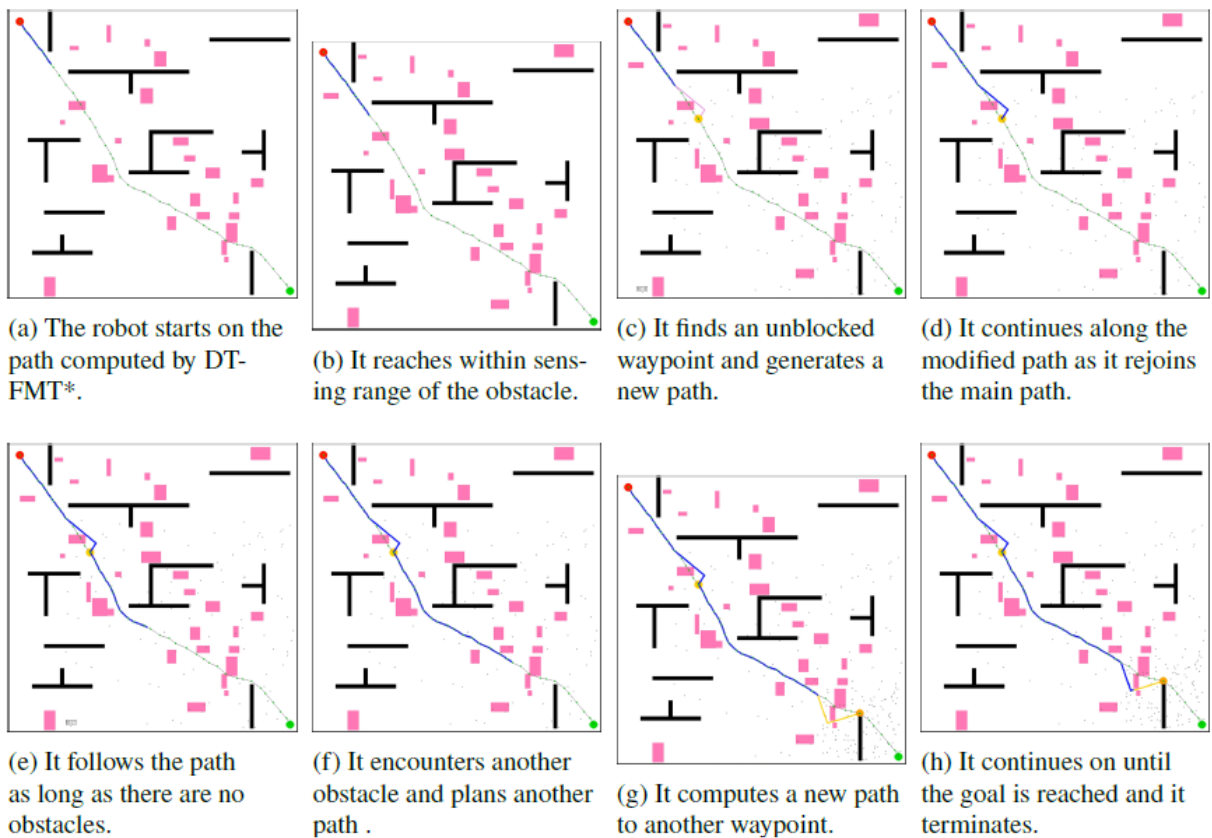


Figure 18: Working procedure of RR-FMT*.

CHAPTER 4: EXPERIMENTATION AND DISCUSSION

In this section we present the results of our experiments. In order to empirically prove the viability of our planner we have conducted two sets of test against both the standard FMT* planner and also against the ST-FMT* planner. All of our experiments were preformed during the course of a few days using a Windows 11, 64 GB RAM, and AMD processor. Table 4.1 gives the result of our comparison with FMT*. The programming language used was Python 3.11 using Visual Studio interface. It is important to note that two different sets of experiments were run separate from each other. The planner was evaluated on seven different 2d environment maps from literature given in Fig 3.3 (a)-(g). We have set the size of our environments as 720 x 720 pixels as we find that it is acceptable distance required.

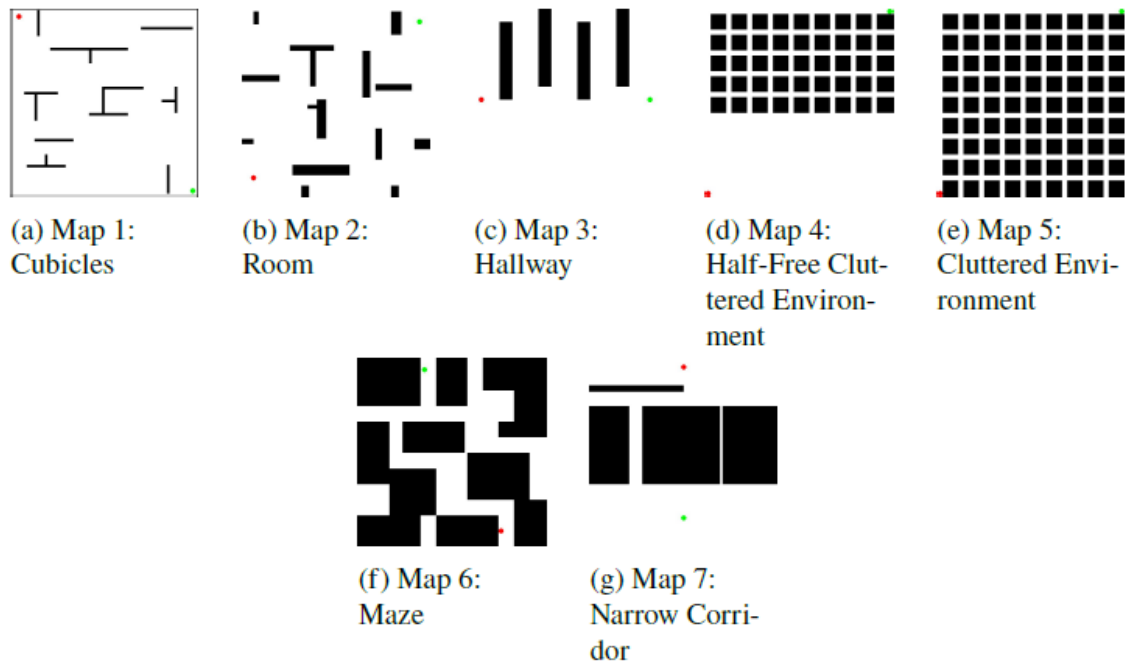


Figure 19: List of Classical Motion Planning Environments used in this work.

Both the FMT* and ST-FMT* planners that we used in our experiments have shared functionality with respect to collision checking, tree growth e.t.c. Since our main focus with DTFMT* is improving path performance with a reduced number of , our total number of samples have remained the same for each planner. The value we are using is 2000 samples. In case of our planner DT-FMT* we have divided these samples into batches of 1000 for each stage of the planning process. The number of iterations to be run in order to evaluate performance is 130 and the average values of path cost and time were calculated.

Map	c*	Initial Path Distance	Final Path Distance	Initial Time (s)	Final Time (s)
Map 1	965	989.9624	973.2381	1.5775	1.4756
Map 2	895	917.8420	902.4884	1.3114	1.3464
Map 3	649	659.7693	651.7199	0.8913	1.0092
Map 4	1078	1140.1551	1122.3930	5.0553	0.5650
Map 5	1136	1241.6540	1192.4367	3.4418	3.9211
Map 6	770	800.2739	777.1681	1.0754	1.2528
Map 7	1065	1136.6432	1071.2102	1.0540	1.0863

Table 1: Path Distance and Time Comparison

Map	c*	Planner	Path Cost	Time (s)	t, p values
Map 1	965	DT-FMT*	973.24	3.05	Distance: t = 13.85 p = 6.37e-34
		FMT*	982.82	3.71	Time: t = 12.46 p = 4.28e-20
Map 2	895	DT-FMT*	902.49	2.66	Distance: t = 18.56 p = 3.45e-49
		FMT*	911.45	3.20	Time: t = 14.36 p = 2.58e-33
Map 3	649	DT-FMT*	651.72	1.90	Distance: t = 18.26 p = 6.48e-42
		FMT*	657.52	2.42	Time: t = 15.73 p = 3.42e-30

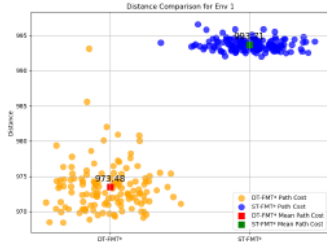
Table 2: Comparison with FMT*

Map	c*	Planner	Path Cost	Time (s)	t, p values
Map 4	1078	DT-FMT*	1122.39	5.06	Distance: t = 0.44 p = 0.66 Time: t = 0.73 p = 0.54
		FMT*	1127.44	5.62	Distance: t = 7.71 p = 3.33e-13 Time: t = 6.98 p = 1.19e-10
Map 5	1136	DT-FMT*	1192.44	7.36	Distance: t = 20.41 p = 4.75e-50 Time: t = 16.00 p = 7.32e-40
		FMT*	1211.29	8.10	Distance: t = 16.72 p = 1.72e-43 Time: t = 14.53 p = 4.12e-35
Map 6	770	DT-FMT*	777.17	2.33	
		FMT*	788.01	2.69	
Map 7	1065	DT-FMT*	1071.21	2.14	
		FMT*	1107.66	2.57	

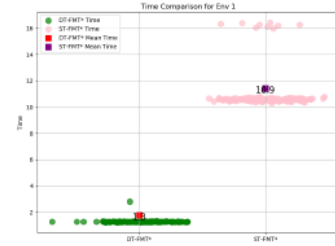
Map	c*	Planner	Path Cost	Time (s)	t and p values
Map 1	965	DT-FMT*	973.4798	1.2791	t dist = 66.3156, p dist = 1.6817e-109 t time = 80.6109, p time = 1.2206e-119
		ST-FMT*	993.7137	10.9327	
Map 2	895	DT-FMT*	904.3894	1.0800	t dist = 75.2391, p dist = 4.8513e-110 t time = 149.9489, p time = 5.7006e-157
		ST-FMT*	954.9683	9.8956	
Map 3	649	DT-FMT*	651.8916	0.7299	t dist = 18.6315, p dist = 3.96273e-45 t time = 231.3086, p time = 5.0914e-215
		ST-FMT*	655.5222	6.5546	
Map 4	1078	DT-FMT*	1122.3930	2.2937	t dist = 8.9804, p dist = 2.6236e-15 t time = 160.3799, p time = 3.0699e-162
		ST-FMT*	1140.5072	20.1590	
Map 5	1136	DT-FMT*	1193.4175	3.8099	t dist = 45.9593, p dist = 2.1143e-83 t time = 150.6859, p time = 6.8969e-167
		ST-FMT*	1287.7622	47.4827	
Map 6	770	DT-FMT*	777.7545	1.0161	t dist = 5.0337, p dist = 9.05871e-07 t time = 153.6632, p time = 1.4976e-184
		ST-FMT*	779.4894	5.4533	
Map 7	1065	DT-FMT*	1072.7599	0.9545	t dist = 0.9545, p dist = 5.2558e-92 t time = 5.2558e-92, p time = 1.3200e-2284
		ST-FMT*	1156.1979	7.4787	t time = 5.2558e-92, p time = 1.3200e-2284

Table 3: Comparison with ST-FMT* (preprocessing included)

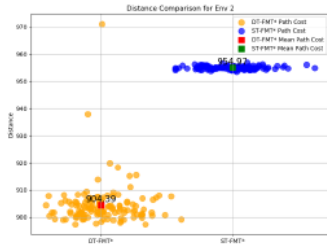
Similarly, the Welch's t-test was also employed to compare the performance metrics of DTFMT* with respect to FMT* and ST-FMT*. We have used Welch's t-test as it provides a more robust comparison metric especially when the variance of the two input datasets differs.



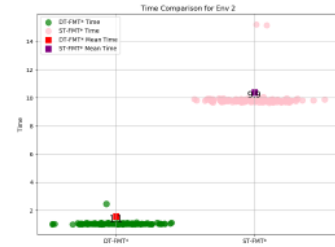
(a) Map 1 Distance



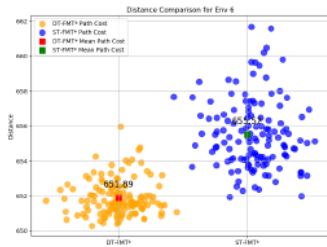
(b) Map 1 Time



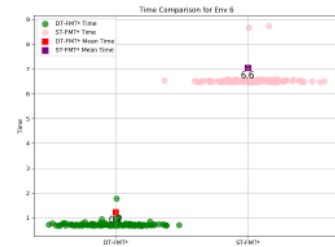
(c) Map 2 Distance



(d) Map 2 Time

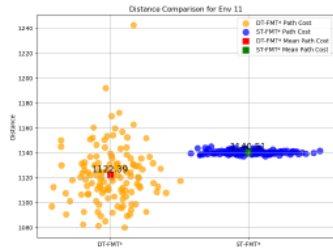


(e) Map 3 Distance

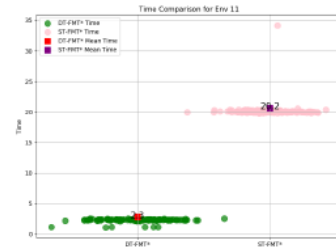


(f) Map 3 Time

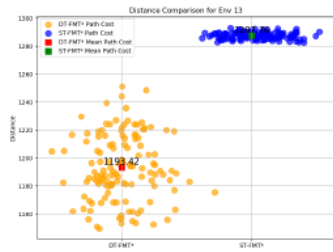
Figure 20: Distance and Time Comparisons Map 1-3



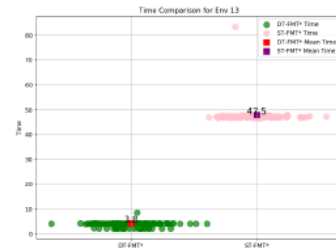
(a) Map 4 Distance



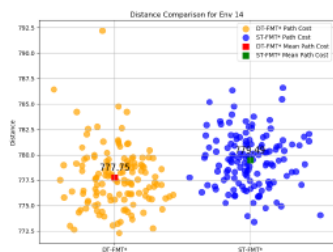
(b) Map 4 Time



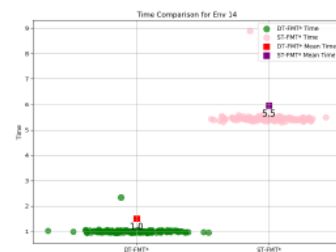
(c) Map 5 Distance



(d) Map 5 Time



(e) Map 6 Distance



(f) Map 6 Time

Figure 21: Distance and Time Comparison for Maps 4-6

4.1 Dynamic Environments

Using the same experimental setup we have also tested our planner in a dynamic constraint. Our dynamic environments as shown in Fig 4.5 involves a time varying series of obstacles. The configuration of the environment, is not know to the robot beforehand and it must modify its path as soon as the obstacles start to appear. We have compared the performance of RRFMT* to the Dynamic version of the RRT* which also rewires its path when in the presence of obstacles.

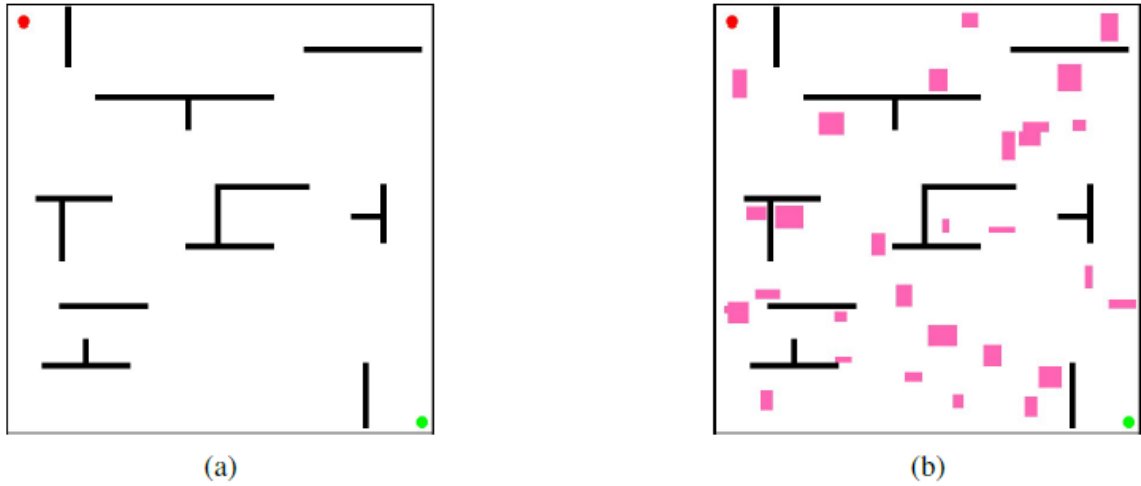


Figure 22: Status of the environment at different values of time

Map	Distance (Dynamic RRT*)	Distance (RR-FMT*)	Time (Dynamic RRT*) (s)	Time (RR-FMT*) (s)
1	1744.378	1031.897	69.593	39.789
2	1701.811	987.686	88.110	38.872
8	1435.941	881.950	98.711	31.574

Table 4: Path Distance and Time Comparison (Dynamic)

Map	T value (Distance)	P-Value (Distance)	T value (Time)	P-Value (Time)
1	18.737	3.053e-19	3.035	8.773e-03
2	22.685	4.052e-37	5.482	3.579e-05
8	11.507	2.866e-13	7.256	2.803e-05

Table 5: T-test (Distance and Time)

4.1 Analysis

Our results in Tables 4.1 4.3 show the improvement in path quality that has been done as a result of our planner. We have exceeded the amount of iterations typically performed to show the validity of our motion planning algorithm and in order to solidify that have also performed the t-test. Across seven different environments and two planner, our values of t and p lie within the acceptable range in order to make our results statistically significant. Similarly we have done the same with the RR-FMT* and compared it to three different dynamic environments. Hereto, the results show a wide variety of improvement over the dynamic RRT* planner and robot we have used at identical velocities.

CHAPTER 5: CONCLUSION AND FUTURE WORK

During the course of this work, we have proposed two different variants of the FMT*: the DT-FMT*, an efficient asymptotically optimal sampling-based planning algorithm designed to improve path quality with reduced sampling for mobile robot applications; and the RR-FMT* algorithm designed to provide an asymptotically optimal solution for mobile robot planning in the presence of dynamic obstacles. The core of the DT-FMT* planner lies in its secure tunnel preprocessing and centralized exploration strategy. This approach allows the planner to achieve high-quality solutions with fewer samples, thereby improving both computational efficiency and path planning performance. The use of a reduced sample distribution, facilitated by secure tunnel construction, ensures that the planner remains highly efficient while maintaining optimal path quality. We prove this in our experiments in the previous sections. The RR-FMT* planner utilizes a similar strategy in order to generate the final path from a position of proximity of one blocked way-point to another. We overcome the inverse relationship between time and sample numbers by generating a very small number of samples between the current position and the goal position which allows us to get a fast solution whilst retaining optimality in the presence of dynamic obstacles. Our simulation and experimental results confirm the effectiveness of DT-FMT*, demonstrating faster convergence and improved performance compared to existing algorithms. These results underscore the planner's capability to provide high-quality solutions with reduced sampling effort, making it highly suitable for environments with limited computational resources or strict time constraints.

Future work will explore extending DT-FMT* to more dynamic and complex environments, such as those with moving obstacles or unknown terrains. Additionally, the algorithm could be adapted real-time applications through high-frequency planning. Integrating DT-FMT* into a hierarchical planning framework could further enhance its adaptability, allowing for robust global and local planning in diverse robotic applications.

REFERENCES

- [1] S. M. LaValle, *Planning Algorithms*. Cambridge University Press, 2006.
- [2] Z. Zuo, C. Liu, Q.-L. Han, and J. Song, “Unmanned aerial vehicles: Control methods and future challenges,” *IEEE/CAA Journal of Automatica Sinica*, vol. 9, no. 4, pp. 601–614, 2022.
- [3] M. Z. Zahid, M. Nadeem, and M. Ismail, “Numerical study of submarine launched underwater vehicle,” in *2020 17th International Bhurban Conference on Applied Sciences and Technology (IBCAST)*, 2020, pp. 472–476.
- [4] M. Elbanhawi and M. Simic, “Sampling-based robot motion planning: A review,” *IEEE Access*, vol. 2, pp. 56–77, 2014.
- [5] A. Valero-Gomez, J. Gómez, S. Garrido, and L. Moreno, “Fast marching methods in path planning,” *IEEE Robotics and Automation Magazine*, vol. 20, pp. 111 – 120, 12 2013.
- [6] M. Otte and E. Frazzoli, “Rrtx: Asymptotically optimal single-query sampling-based motion planning with quick replanning,” *The International Journal of Robotics Research*, vol. 35, no. 7, pp. 797–822, 2016. [Online]. Available: <https://doi.org/10.1177/0278364915594679>
- [7] L. Janson, E. Schmerling, A. Clark, and M. Pavone, “Fast marching tree: A fast marching sampling-based method for optimal motion planning in many dimensions,” *The International Journal of Robotics Research*, vol. 34, no. 7, pp. 883–921, 2015, pMID: 27003958. [Online]. Available: <https://doi.org/10.1177/0278364915577958>
- [8] J. D. Gammell, T. D. Barfoot, and S. S. Srinivasa, “Batch informed trees (bit*): Informed asymptotically optimal anytime search,” *The International Journal of*

Robotics Research, vol. 39, no. 5, pp. 543–567, 2020. [Online]. Available: <https://doi.org/10.1177/0278364919890396>

- [9] J. Xu, K. Song, D. Zhang, H. Dong, Y. Yan, and Q. Meng, “Informed anytime fastmarching tree for asymptotically optimal motion planning,” *IEEE Transactions on Industrial Electronics*, vol. 68, no. 6, pp. 5068–5077, 2021.
- [10] J. Silveira, K. Cabral, S. Givigi, and J. A. Marshall, “Real-time fast marching tree for mobile robot motion planning in dynamic environments,” in *2023 IEEE International Conference on Robotics and Automation (ICRA)*, 2023, pp. 7837–7843.
- [11] I. Giorgi, F. A. Tiroto, O. Hagen, F. Aider, M. Gianni, M. Palomino, and G. L. Masala, “Friendly but faulty: A pilot study on the perceived trust of older adults in a social robot,” *IEEE Access*, vol. 10, pp. 92 084–92 096, 2022.
- [12] S. Mussakhojayeva and A. Sandygulova, “Cross-cultural differences for adaptive strategies of robots in public spaces,” in *2017 26th IEEE International Symposium on Robot and Human Interactive Communication (RO-MAN)*, 2017, pp. 573–578.
- [13] S. O. Oruma and S. Petrovic, “Security threats to 5g networks for social robots in public spaces: A survey,” *IEEE Access*, vol. 11, pp. 63 205–63 237, 2023.
- [14] S. Gao and C. Du, “Design of garbage automatic sorting and disposal robot,” in *2023 International Symposium on Intelligent Robotics and Systems (ISoIRS)*, 2023, pp. 81–85.
- [15] S. Ono, Y. Okazaki, K. Kanetsuna, and M. Mizumoto, “Egocentric, altruistic, or hypocritic?: cross-cultural study of choice between pedestrian-first and driver-first of autonomous car,” *IEEE Access*, vol. 11, pp. 108 716–108 726, 2023.

- [16] H. Dong, “Research progress and review on service interaction between intelligent service robots and customers,” in *2023 International Conference on Service Robotics (ICoSR)*, 2023, pp. 1–8.
- [17] Z. Jiang, J. Xu, H. Li, and Q. Huang, “Stable parking control of a robot astronaut in a space station based on human dynamics,” *IEEE Transactions on Robotics*, vol. 36, no. 2, pp. 399–413, 2020.
- [18] S. M. LaValle, *Planning Algorithms*. USA: Cambridge University Press, 2006.
- [19] J. R. Sánchez-Ibáñez, C. J. Pérez-del Pulgar, and A. García-Cerezo, “Path planning for autonomous mobile robots: A review,” *Sensors*, vol. 21, no. 23, 2021. [Online]. Available: <https://www.mdpi.com/1424-8220/21/23/7898>
- [20] L. Dong, Z. He, C. Song, and C. Sun, “A review of mobile robot motion planning methods: from classical motion planning workflows to reinforcement learning-based architectures,” *Journal of Systems Engineering and Electronics*, vol. 34, no. 2, pp. 439–459, 2023.
- [21] 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.
- [22] Y. B. Jmaa and D. Duvivier, “A review of path planning algorithms,” in *Intelligent Systems Design and Applications*, A. Abraham, A. Bajaj, T. Hanne, P. Siarry, and K. Ma, Eds. Cham: Springer Nature Switzerland, 2024, pp. 119–130.
- [23] A. Artuñedo, G. Corrales, J. Villagra, and J. Godoy, “Machine learning based motion planning approach for intelligent vehicles,” in *2020 IEEE Intelligent Vehicles Symposium (IV)*, 2020, pp. 963–970.

- [24] J. Wang, T. Zhang, N. Ma, Z. Li, H. Ma, F. Meng, and M. Q. Meng, “A survey of learning-based robot motion planning,” *IET Cyber-Systems and Robotics*, 2021. [Online]. Available: <https://api.semanticscholar.org/CorpusID:236381976>
- [25] S. M. LaValle, “Rapidly-exploring random trees: a new tool for path planning,” *The Annual Research Report*, 1998. [Online]. Available: <https://api.semanticscholar.org/CorpusID:14744621>
- [26] A. H. Qureshi, K. F. Iqbal, S. M. Qamar, F. Islam, Y. Ayaz, and N. Muhammad, “Potential guided directional-rrt* for accelerated motion planning in cluttered environments,” in *2013 IEEE International Conference on Mechatronics and Automation*, 2013, pp. 519–524.
- [27] Rodriguez, X. Tang, J.-M. Lien, and N. Amato, “An obstacle-based rapidly-exploring random tree,” in *Proceedings 2006 IEEE International Conference on Robotics and Automation, 2006. ICRA 2006.*, 2006, pp. 895–900.
- [28] J. Nasir, F. Islam, U. Malik, Y. Ayaz, O. Hasan, M. Khan, and M. S. Muhammad, “Rrt*-smart: A rapid convergence implementation of rrt*,” *International Journal of Advanced Robotic Systems*, vol. 10, no. 7, p. 299, 2013. [Online]. Available: <https://doi.org/10.5772/56718>
- [29] J. D. Gammell, S. S. Srinivasa, and T. D. Barfoot, “Informed rrt*: Optimal sampling-based path planning focused via direct sampling of an admissible ellipsoidal heuristic,” in *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2014, pp. 2997–3004.
- [30] J. Kuffner and S. LaValle, “Rrt-connect: An efficient approach to single-query path planning,” in *Proceedings 2000 ICRA. Millennium Conference. IEEE International Conference on Robotics and Automation. Symposia Proceedings (Cat. No.00CH37065)*, vol. 2, 2000, pp. 995–1001 vol.2.
- [31] G. Sánchez and J.-C. Latombe, “A single-query bi-directional probabilistic roadmap planner with lazy collision checking,” in *Robotics Research*, R. A. Jarvis

- and A. Zelinsky, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2003, pp. 403–417.
- [32] J. Zhong and J. Su, “Triple-rrts for robot path planning based on narrow passage identification,” in *2012 International Conference on Computer Science and Information Processing (CSIP)*, 2012, pp. 188–192.
- [33] D. Ferguson and A. Stentz, “Anytime rrts,” in *2006 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2006, pp. 5369–5375.
- [34] W. Wang, Y. Li, X. Xu, and S. X. Yang, “An adaptive roadmap guided multi-rrts strategy for single query path planning,” in *2010 IEEE International Conference on Robotics and Automation*, 2010, pp. 2871–2876.
- [35] O. Khatib, *The Potential Field Approach And Operational Space Formulation In Robot Control*. Boston, MA: Springer US, 1986, pp. 367–377. [Online]. Available: <https://doi.org/10.1007/978-1-4757-1895-926>
- [36] R. Bohlin and L. Kavraki, “Path planning using lazy prm,” in *Proceedings 2000 ICRA. Millennium Conference. IEEE International Conference on Robotics and Automation. Symposia Proceedings (Cat. No.00CH37065)*, vol. 1, 2000, pp. 521–528 vol.1.
- [37] C. Zhong and H. Liu, “A region-specific hybrid sampling method for optimal path planning,” *International Journal of Advanced Robotic Systems*, vol. 13, no. 2, p. 71, 2016. [Online]. Available: <https://doi.org/10.5772/63031>
- [38] D. Hsu, T. Jiang, J. Reif, and Z. Sun, “The bridge test for sampling narrow passages with probabilistic roadmap planners,” in *2003 IEEE International Conference on Robotics and Automation (Cat. No.03CH37422)*, vol. 3, 2003, pp. 4420–4426 vol.3.
- [39] B. Ichter, E. Schmerling, and M. Pavone, “Group marching tree: Sampling-based approximately optimal motion planning on gpus,” in *2017 First IEEE International Conference on Robotic Computing (IRC)*. Los Alamitos, CA, USA:

IEEE Computer Society, apr 2017, pp. 219–226. [Online]. Available:
<https://doi.ieeecomputersociety.org/10.1109/IRC.2017.72>

- [40] B. Chandler and M. A. Goodrich, “Online rrt* and online fmt*: Rapid replanning with dynamic cost,” in *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2017, pp. 6313–6318.
- [41] J. Starek, J. Gómez, E. Schmerling, L. Janson, L. Moreno, and M. Pavone, “An asymptotically optimal sampling-based algorithm for bi-directional motion planning,” *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems. IEEE/RSJ International Conference on Intelligent Robots and Systems*, vol. 2015, 07 2015.
- [42] W. Gao, Q. Tang, J. Yao, Y. Yang, and D. Yu, “Heuristic bidirectional fast marching tree for optimal motion planning,” in *2018 3rd Asia-Pacific Conference on Intelligent Robot Systems (ACIRS)*, 2018, pp. 71–77.
- [43] W. Reid, R. Fitch, A. H. Göktoğgan, and S. Sukkarieh, *Motion Planning for Reconfigurable Mobile Robots Using Hierarchical Fast Marching Trees*. Cham: Springer International Publishing, 2020, pp. 656–671. [Online]. Available: <https://doi.org/10.1007/978-3-030-43089-442>
- [44] J. Xia, Z. Jiang, H. Zhang, R. Zhu, and H. Tian, “Dual fast marching tree algorithm for human-like motion planning of anthropomorphic arms with task constraints,” *IEEE/ASME Transactions on Mechatronics*, vol. 26, no. 5, pp. 2803–2813, 2021.
- [45] H. Choset and J. Burdick, “Sensor based planning. i. the generalized voronoi graph,” in *Proceedings of 1995 IEEE International Conference on Robotics and Automation*, vol. 2, 1995, pp. 1649–1655 vol.2.
- [46] J. Hou, Z. Liu, and H. Su, “Obstacle based fast marching tree for global motion planning,” in *IECON 2022 – 48th Annual Conference of the IEEE Industrial Electronics Society*, 2022, pp. 1–6.

- [47] O. Khatib, “Real-time obstacle avoidance for manipulators and mobile robots,” in *Proceedings. 1985 IEEE International Conference on Robotics and Automation*, vol. 2, 1985, pp. 500–505.
- [48] J. Amirian and M. Jamzad, “Adaptive motion planning with artificial potential fields using a prior path,” 10 2015, pp. 731–736.
- [49] N. Zhang, Y. Zhang, C. Ma, and B. Wang, “Path planning of six-dof serial robots based on improved artificial potential field method,” in *2017 IEEE International Conference on Robotics and Biomimetics (ROBIO)*, 2017, pp. 617–621.
- [50] G. Shao, Z. Li, Y. Wen, and L. Zhuang, “The behavior coding of artificial life body based on dynamic potential field approach,” in *2006 6th World Congress on Intelligent Control and Automation*, vol. 1, 2006, pp. 2546–2550.
- [51] N. He, Y. Su, j. Guo, X. Fan, Z. Liu, and B. Wang, “Dynamic path planning of mobile robot based on artificial potential field,” in *2020 International Conference on Intelligent Computing and Human-Computer Interaction (ICHCI)*, 2020, pp. 259–264.
- [52] W. Chao, M. Feng, W. Qing, and W. Shuwu, “A situation awareness approach for usv based on artificial potential fields,” in *2017 4th International Conference on Transportation Information and Safety (ICTIS)*, 2017, pp. 232–235.
- [53] A. H. Qureshi, S. Mumtaz, K. F. Iqbal, B. Ali, Y. Ayaz, F. Ahmed, M. S. Muhammad, O. Hasan, W. Y. Kim, and M. Ra, “Adaptive potential guided directional-rrt*,” in *2013 IEEE International Conference on Robotics and Biomimetics (ROBIO)*, 2013, pp. 1887–1892.
- [54] 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. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S0921889017309387>

- [55] W. Xinyu, L. Xiaojuan, G. Yong, S. Jiadong, and W. Rui, "Bidirectional potential guided rrt* for motion planning," *IEEE Access*, vol. 7, pp. 95 046–95 057, 2019.
- [56] S. Petti and T. Fraichard, "Partial motion planning framework for reactive planning within dynamic environments," 09 2005.
- [57] J. Bialkowski, M. Otte, S. Karaman, and E. Frazzoli, "Efficient collision checking in sampling-based motion planning via safety certificates," *The International Journal of Robotics Research*, vol. 35, no. 7, pp. 767–796, 2016. [Online]. Available: <https://doi.org/10.1177/0278364915625345>
- [58] A. Sintov and A. Shapiro, "Time-based rrt algorithm for rendezvous planning of two dynamic systems," *2014 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 6745–6750, 2014. [Online]. Available: <https://api.semanticscholar.org/CorpusID:3211158>
- [59] G. Chi, C. Wang, J. Wang, and M. Meng, "Risk-dtrrt-based optimal motion planning algorithm for mobile robots," *IEEE Transactions on Automation Science and Engineering*, vol. PP, pp. 1–18, 11 2018.
- [60] J. Qi, H. Yang, and H. Sun, "Mod-rrt*: A sampling-based algorithm for robot path planning in dynamic environment," *IEEE Transactions on Industrial Electronics*, vol. PP, pp. 1–1, 06 2020.
- [61] J. Wang, M. Meng, and O. Khatib, "Eb-rrt: Optimal motion planning for mobile robots," *IEEE Transactions on Automation Science and Engineering*, vol. PP, pp. 1–11, 04 2020.
- [62] S. Quinlan and O. Khatib, "Elastic bands: connecting path planning and control," *[1993] Proceedings IEEE International Conference on Robotics and Automation*, pp. 802–807 vol.2, 1993. [Online]. Available: <https://api.semanticscholar.org/CorpusID:5641886>

- [63] Y. Chen, Z. He, and S. Li, "Horizon-based lazy optimal rrt for fast, efficient replanning in dynamic environment," *Auton. Robots*, vol. 43, no. 8, p. 2271–2292, Dec. 2019. [Online]. Available: <https://doi.org/10.1007/s10514-019-09879-8>
- [64] Z. Wu, Y. Chen, J. Liang, B. He, and Y. Wang, "St-fmt*: A fast optimal global motion planning for mobile robot," *IEEE Transactions on Industrial Electronics*, vol. 69, no. 4, pp. 3854–3864, 2022.