BIDIRECTIONAL AND SEMI-BIDIRECTIONAL RAPIDLY-EXPLORING RANDOM TREE-BASED VARIANTS FOR ROBOT MOTION PLANNING

REZA MASHAYEKHI

FACULTY OF COMPUTER SCIENCE AND INFORMATION TECHNOLOGY
UNIVERSITY OF MALAYA
KUALA LUMPUR

2021

BIDIRECTIONAL AND SEMI-BIDIRECTIONAL RAPIDLY-EXPLORING RANDOM TREE-BASED VARIANTS FOR ROBOT MOTION PLANNING

REZA MASHAYEKHI

THESIS SUBMITTED IN FULFILMENT OF THE REQUIREMENTS FOR THE DEGREE OF DOCTOR OF PHILOSOPHY

FACULTY OF COMPUTER SCIENCE AND INFORMATION TECHNOLOGY
UNIVERSITY OF MALAYA
KUALA LUMPUR

2021

UNIVERSITI MALAYA

ORIGINAL LITERARY WORK DECLARATION

Name of Candidate: **Reza Mashayekhi**

Registration/Matric No.: **WHA150068 - 17028013/1**

Name of Degree: **Doctor of Philosophy (PhD)**

Title of Project Paper/Research Report/Dissertation/Thesis ("this Work"):

Bidirectional and Semi-Bidirectional Rapidly-exploring Random Tree-Based Variants for Robot Motion Planning

Field of Study: **Robotics**

I do solemnly and sincerely declare that:

(1) I am the sole author/writer of this Work;
(2) This work is original;
(3) Any use of any work in which copyright exists was done by way of fair dealing and for permitted purposes and any excerpt or extract from, or reference to or reproduction of any copyright work has been disclosed expressly and sufficiently and the title of the Work and its authorship have been acknowledged in this Work;
(4) I do not have any actual knowledge nor do I ought reasonably to know that the making of this work constitutes an infringement of any copyright work;
(5) I hereby assign all and every rights in the copyright to this Work to the University of Malaya ("UM"), who henceforth shall be owner of the copyright in this Work and that any reproduction or use in any form or by any means whatsoever is prohibited without the written consent of UM having been first had and obtained;
(6) I am fully aware that if in the course of making this Work I have infringed any copyright whether intentionally or otherwise, I may be subject to legal action or any other action as may be determined by UM.

Candidate's Signature Date:

Subscribed and solemnly declared before,

Witness's Signature Date:

Name:
Designation:

BIDIRECTIONAL AND SEMI-BIDIRECTIONAL RAPIDLY-EXPLORING RANDOM TREE-BASED VARIANTS FOR ROBOT MOTION PLANNING

ABSTRACT

Motion planning is involved in various applications such as Unmanned Aerial Vehicles (UAVs), Autonomous Underwater Vehicles (AUVs), driver-less cars, virtual prototyping, biology, and computer graphics. Planners need to find collision-free paths for movable agents from one point to another in state spaces. Path planning is mostly about finding paths in continuous spaces, which is considered as an Np-hard problem. In order to avoid this complexity, planners discretize continuous spaces into discrete spaces to limit the number of states that planners need to check to release paths. There are two types of motion planners: graph-based planners and sampling-based planners. Graph-based methods, such as A*, are efficient. Nonetheless, they need to use a priori approximation of the state space. If these approximations are not chosen accurately, they are not able to provide appropriate solutions. If the selected resolution is low, the output would be low quality. If the selected resolution is high, it is computationally expensive to solve. On the other hand, sampling-based methods do not need to have any resolution for solving planning problems. They use random sampling to avoid prior discretization of state spaces. Rapidly-exploring Random Tree (RRT) is one of the most popular sampling-based methods for single-query planning problems due to its ability to find solutions efficiently. Informed RRT* is an optimized version of RRT, which not only implements the rewiring process to optimize the tree but also limits the search area to a subset of the state space to return near-optimal solutions faster. However, before finding an initial solution, the planner is not able to shrink the problem domain. Therefore, it searches all over the problem domain to be able to find an initial solution. Moreover, unidirectional RRTs, such as Informed

RRT*, take more time to find initial solutions in comparison to the bidirectional RRTs. This thesis proposes two new motion planners, one is a bidirectional motion planner (Informed RRT*-Connect), and another one is a semi-bidirectional motion planner (Hybrid RRT). Informed RRT*-Connect is the informed version of RRT*-Connect that uses direct sampling after an initial solution is found. Unlike RRT*-Connect, the proposed method checks only the states that can potentially provide better solutions than the current solution. On the other hand, Hybrid RRT divides the planning process into three parts: finding initial solutions by a dual-tree search, combining two trees into one, and optimizing the solution. Hybrid RRT implements a dual-tree search to obtain its initial solution, which helps it find solutions faster than unidirectional searches. Then, it combines the start tree and the goal tree of the dual-tree search into one to implement informed sampling on a single tree to optimize the current solution. The simulation has been carried out in the Open Motion Planning Library (OMPL). Planners have been compared in terms of finding initial solutions, success rate, and path length. The simulations show that the proposed methods surpass state-of-the-art motions planners in terms of the success rate and the path length.

Keywords: RRT, Motion Planning, Robotics, Informed sampling.

VARIAN PENJELAJAHAN-PANTAS POHON RAWAK DWI ARAH DAN SEMI-DWI ARAH UNTUK PERANCANGAN PERGERAKAN ROBOT

ABSTRAK

Perancangan pergerakan (motion planning) boleh didapti dalam pelbagai aplikasi seperti Kenderaan Udara Tanpa Manusia (UAVs), Kenderaan Bawah Permukaan Air Berautonomi (AUVs), kereta tanpa kenderaan, pemprototaipan maya, biologi, dan grafik komputer. Perancang perlu mencari jalan bebas perlanggaran untuk agen bergerak dari satu titik ke titik lain di ruang keadaan. Perancangan laluan kebanyakannya mencari laluan dalam ruang yang berterusan, yang dianggap sebagai masalah Np-hard untuk diselesaikan. Untuk mengelakkan masalah ini, para perancang membezakan ruang yang berterusan kepada ruang diskret untuk mengehadkan bilangan keadaan yang perancang perlu semak untuk melepaskan laluan. Terdapat dua jenis perancangan pergerakan: perancangan berdasarkan graf (graph-based planners) dan perancangan beradasarkan pensampelan (sampling-based planners). Kaedah berdasarkan graf seperti A* adalah efisyen. Walaupun begitu, ia perlu menggunakan pendekatan penaakulan ruang keadaan. Sekiranya pendekatan ini tidak dipilih dengan tepat, ia tidak akan dapat memberikan penyelesaian yang sesuai. Sekiranya resolusi dipilih terlalu rendah, penyelesaiannya boleh dijumpai dengan pantas, tetapi berkualiti rendah. Sekiranya resolusi dipilih terlalu tinggi, laluan dipilih akan berkualiti tinggi, tetapi cara penyelesaian graf sedemikian akan menjadi mahal secara komputasi. Sebaliknya, kaedah berasaskan pensampelan tidak perlu mempunyai resolusi untuk penyelesaian masalah perancangan. Ia menggunakan persampelan rawak untuk mengelakkan diskretisasi ruang keadaan sebelumnya. Rapidly-exploring Random Tree (RRT) adalah salah satu kaedah berasaskan pensampelan yang paling popular untuk menyelesaikan masalah perancangan pertanyaan tunggal (single-query), yang telah banyak digunakan

untuk masalah perancangan gerakan kerana keupayaannya untuk mencari penyelesaian dengan cekap. Informed RRT* adalah versi RRT yang dioptimalkan, yang tidak hanya mengimplementasikan proses pewayaran semula (rewiring) untuk mengoptimalkan pohon tetapi juga membatasi kawasan pencarian kepada subset ruang keadaan untuk mendapatkan solusi yang hampir-optimal (near-optimal) dengan lebih cepat. Walau bagaimanapun, sebelum menjumpai penyelesaian awal, perancang tidak mampu untuk mengecilkan domain masalah. Disamping itu, RRT satu arah, seperti Informed RRT*, mengambil lebih banyak masa untuk mencari penyelesaian awal berbanding dengan RRT dwi arah. Tesis ini mencadangkan dua perancangan pergerakan baru, pertama, adalah perancang pergerakan dwi arah (Informed RRT*-Connect), dan satu lagi adalah perancang pergerakan semi dwi arah (Hybrid RRT). Informed RRT*-Connect adalah versi RRT*-Connect yang menggunakan persampelan langsung selepas penyelesaian pertama ditemui. Tidak seperti RRT*-Connect, kaedah yang dicadangkan hanya memeriksa keadaan-keadaan yang berpotensi memberikan penyelesaian yang lebih baik daripada penyelesaian semasa. Sebaliknya, Hybrid RRT membahagikan proses perancangan menjadi tiga bahagian: mencari penyelesaian awal dengan carian dwi-pohon (dual-tree search), menggabungkan dua pokok menjadi satu, dan mengoptimumkan penyelesaiannya. Untuk mendapatkan penyelesaian awal, Hybrid RRT menerapkan carian dwi-pohon, yang membantu ia mencari penyelesaian lebih cepat daripada carian satu arah (unidirectional search). Kemudian, ia menggabungkan pohon permulaan dan pohon matlamat carian dwi-pohon ke dalam satu untuk melaksanakan pensampelan yang dimaklumkan (informed sampling) pada satu pohon untuk mengoptimumkan penyelesaian semasa. Simulasi dijalankan menggunakan Open Motion Planning Library (OMPL). Perancang-perancang dibandingkan dengan masa untuk mencari penyelesaian awal, kadar kejayaan untuk mencari penyelesaian dalam masa terhad, dan jarak laluan. Simulasi menunjukkan bahawa kaedah yang dicadangkan

mencapai penambahbaikan berbanding kaedah sedia ada.

Kata kunci: RRT, Perancangan Pergerakan, Robotik, Persampelan bermaklumat.

ACKNOWLEDGEMENTS

It is my pleasure to have the opportunity to thank all who supports me during this study, without whom this thesis would not have been possible.

First and foremost, I praise and thank God, the most beneficent, the most merciful, for everything. There is no way for me to thank God enough.

I would like to thank my supervisors for their guidance throughout my Ph.D. This work was only possible because they gave me the freedom to pursue the research wherever it led, and for that, I am grateful. It was my honor to work with a knowledgeable team, Associate Prof. Dr. Mohd Yamani Idea Idris, Dr. Mohammad Hossien Anisi, and Dr. Ismail Ahmedy. I appreciate their help and support during research.

I am grateful to my parents for all their endless love and encouragement. They are always supportive and helpful in all aspects of my life. I am also thankful to my dear parents-in-law because of their endless kindness, support, and well-wishes. Moreover, a special thanks to my grandmother, Mahin, who is no longer with us.

I would like to express my sincere gratitude to my beloved wife, Sahar, for her unwavering support and belief in me. She helped me much, motivated me, and made this duration easy and convenient for me.

I am deeply grateful to my siblings, Mojgan, Behnaz, Sharareh, and Amir. They have been supporting me during my Ph.D. study and at any stage of my life. Additionally, thanks to my niece, Melika, and nephews, Amirali and Kourosh.

I am thankful to all my friends, especially those who motivate me to pursue my study. A special thanks to Hamed Esmaeeli, Mahdi Sadat, Amirhossein Azizian, Mina Kargozar, Khatereh Sepehri, Behzad Pournaghi, Alireza Erfani, Amin and Farhad Askarizadeh, and Amin Khodabandeh. They have been kind and helpful. They are like family to me.

TABLE OF CONTENTS

Abstract iii

Abstrak v

Acknowledgements viii

Table of Contents ix

List of Figures xiii

List of Tables xiv

List of Definitions xv

List of Symbols and Abbreviations xvi

CHAPTER 1: INTRODUCTION 1

1.1 Research Background 1

1.1.1 Graph-based Approaches 2

1.1.2 Sampling-based Approaches 3

1.2 Problem Statements 5

1.3 Research Objectives 6

1.4 Research Scope 6

1.5 Overview of the Chapters 7

CHAPTER 2: LITERATURE REVIEW 8

2.1 Planning Problems 8

2.2 Graph-based Searches 10

2.2.1 Near-resolution-optimal Search 11

2.2.2 Incremental Search 12

2.2.3 Anytime Search 13

2.2.4 The problem of graph-based methods 14

2.3 Sampling-based Planners 16

2.3.1 Multi-query problems 17

2.3.2 Single-query problems 19

2.3.2.1 Rapidly-exploring Random Trees (RRTs) 19

2.3.2.2 Nonoptimized RRT 20

2.3.2.3 Asymptotically optimal RRT 20

2.3.2.4 Unidirectional RRT 21

2.3.2.5 Bidirectional RRT 22

2.3.2.6 RRT* 22

2.3.2.7 Sample Distributions 25

2.3.2.8 Heuristic-based Approaches 27

2.4 Discussion 29

CHAPTER 3: METHODOLOGY 33

3.1 Methodology 33

3.1.1 Phase One *(Literature Review)* 33

3.1.2 Phase Two *(The First Proposed Method, Informed RRT*-Connect)* 38

3.1.3 Phase Three *(The Second Proposed Method, Hybrid RRT)* 38

3.1.4 Phase Four *(Evaluation)* 39

3.1.4.1 Conceptual Overview of OMPL 41

3.1.4.2 Implementation of Core Concepts 43

3.1.4.3 benchmarking in OMPL 47

3.2 Summary 51

CHAPTER 4: INFORMED RRT*-CONNECT 52

4.1 Introduction 53
4.2 Background 63
4.2.1 Problem Definition 64
4.2.2 Informed Set 65
4.3 Informed RRT*-Connect *(The first proposed method)* 65
4.3.1 Graph Pruning 68
4.3.2 Direct Sampling of An Ellipsoidal Subset 69
4.3.3 Rewiring Neighborhood 71
4.3.3.1 r-disc variant 71
4.3.3.2 k-nearest variant 72
4.4 Simulation 73
4.4.1 Single Cube 74
4.4.2 Multiple Narrow Passages 76
4.4.3 OMPL App simulations 78
4.4.3.1 *Maze Planar* 79
4.4.3.2 *Home* 79
4.4.3.3 *Apartment Hard* 80
4.5 Discussion 81

CHAPTER 5: HYBRID RRT 88

5.1 Introduction 90
5.2 Hybrid RRT *(The second proposed method)* 92
5.2.1 Hybrid RRT Algorithm 92
5.2.1.1 *FindFirstSolution* Function 93
5.2.1.2 *CombineTwoTrees* Function 96

5.2.1.3 *OptimizeTree* Function 101

5.3 Simulation 105

5.3.1 Find Initial Solutions 106

5.3.1.1 *BugTrap_planar* 106

5.3.2 Find near-optimal Solutions 107

5.3.2.1 *Maze_planar* 107

5.3.2.2 *Home* 108

5.3.2.3 *Twistycool* 108

5.3.3 The proposed methods comparison 109

5.4 Discussion 109

5.4.1 Finding initial solutions 109

5.4.2 Hybrid RRT comparison with the existing methods 113

5.4.3 Informed RRT*-Connect versus Hybrid RRT 115

5.4.4 Summary 116

CHAPTER 6: DISCUSSION AND CONCLUSION 118

References 124

List of Publications and Papers Presented 136

LIST OF FIGURES

Figure 1.1: Autonomous Robots 1

Figure 2.1: Different graph resolutions 15

Figure 2.2: Mapped obstacles in different graph resolutions 16

Figure 2.3: RRT algorithm sample run 23

Figure 3.1: Workflow of the study 34

Figure 3.2: OMPL API Overview 42

Figure 4.1: The informed set 66

Figure 4.2: Single Cube and Multiple Narrow Passage environments 74

Figure 4.3: One run of the planners in the Single Cube scenario 74

Figure 4.4: Single Cube result graph 75

Figure 4.5: Multiple Narrow Passages example 77

Figure 4.6: Multiple narrow passages result graph 78

Figure 4.7: OMPL App scenarios 79

Figure 4.8: Success Rate Graph 82

Figure 4.9: Solution Length Graph 83

Figure 5.1: Combining two trees into one. 99

Figure 5.2: *BugTrap_planar* state space 105

Figure 5.3: OMPL App scenarios 106

Figure 5.4: Success Rate Graph 110

Figure 5.5: Solution Length Graph 111

Figure 5.6: Comparison graphs of the proposed methods 112

LIST OF TABLES

Table 4.1: Simulation results of Informed RRT*-Connect 85

Table 5.1: Time needed to find initial solutions 107

Table 5.2: Simulation results of Hybrid RRT 114

Table 5.3: Comparison result of the proposed methods 116

LIST OF DEFINITIONS

Definition 2.1: The feasible motion planning problems..... 8

Definition 2.2: The optimal planning problem. 9

Definition 2.3: Probabilistic completeness..... 17

Definition 2.4: Almost-sure asymptotic optimality. 17

LIST OF SYMBOLS AND ABBREVIATIONS

$\emptyset$: The empty set.

σ^* : An optimal path.

σ' : A feasible path.

c_{best} : The best obtained path cost.

c_{min} : The cost of a direct line connected the start location to the goal location.

E : The set of edges.

G : The graph that consists of vertices and edges, $G = (V, E)$.

$min_{x_{soln}}$: The minimum solution cost.

$previous\_c_{best}$: The second short path cost.

$sigma$: A sequence of states.

$Tree_a$: First tree of bidirectional RRTs.

$Tree_b$: Second tree of bidirectional RRTs.

V : The set of vertices.

X : The State Space, Configuration Space, Environment.

x : A state in the state space, $x \in X$.

X_{free} : Collision-free States.

X_{goal} : The goal area.

x_{goal} : The goal location.

X_{near} : A set of near vertices to the sample.

$x_{nearest}$: The nearest vertex to the sample.

x_{new} : The new vertex.

X_{obs} : Obstacle States.

x_{parent} : The parent of the state, x.

x_{rand} : A random state.

X_{soln} : The solution set.

x_{start} : The start location.

X_f : A subset of the state space.

AUVs : Autonomous Underwater Vehicles.

DOF : Degree of Freedom.

OMPL : Open Motion Planning Library.

PRMs : Probabilistic roadmaps.

RRT : Rapidly-exploring Random Tree.

RRT* : Optimized version of RRT.

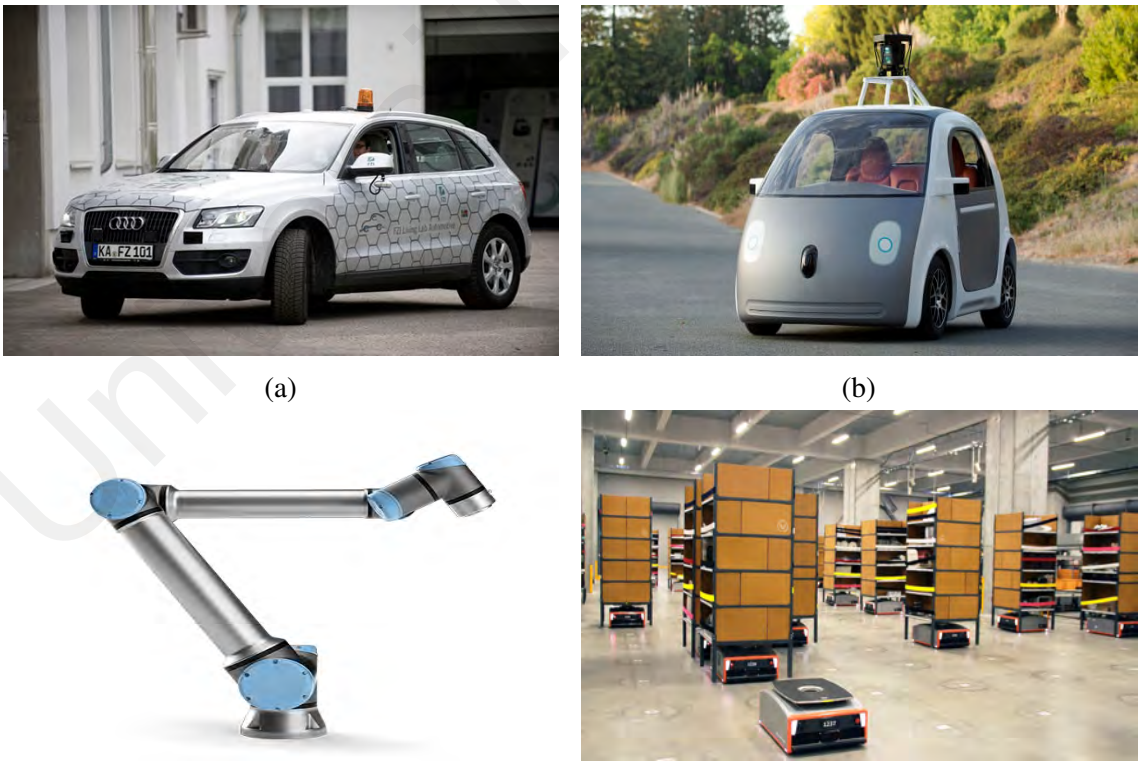
UAVs : Unmanned Aerial Vehicles.

CHAPTER 1: INTRODUCTION

This chapter provides an overview of this research including a background of motion planning (Section 1.1), the problem statement (Section 1.2), the research objectives (Section 1.3), the research scope (Section 1.4), and an overview of the following chapters (Section 1.5).

1.1 Research Background

Autonomous robots have become one of the most demanding fields of research due to their applications in many fields in which they would work better and safer than humans to accomplish tasks. Autonomous robots have many applications such as self-driving cars (Fulgenzi et al., 2008; González et al., 2015; Klemm et al., 2015; Kuwata et al., 2009; Lan & Di Cairano, 2015; Paden et al., 2016; Yoon & Crane, 2011), Unmanned Aerial



(a) (b)

(c) (d)

Figure 1.1: (a) CoCar, a vehicle modified for autonomous driving, (b) Google Self-Driving Car, (c) Universal Robots industrial collaborative robot arm, UR10e, (d) GreyOrange warehouse robots.

Vehicles (UAVs) (Zammit & Van Kampen, 2018), warehouse automation (Wohlsen, 2014), industrial robots (Ellekilde & Petersen, 2013), welding multi-degree of freedom (DOF) robots (Olsen & Petersen, 2007), and domestic robots (Srinivasa et al., 2010) (Srinivasa et al., 2012), (Figure 1.1).

One of the most important parts of autonomous robots is *motion planning* (LaValle, 2006). Motion planning is about finding a sequence of configurations which helps a robot start moving from an initial location, the *start location*, and reach within the desired region, the *goal location*. This set of connected configurations, the *path*, has to be collision-free. In other words, motion planners must provide a set of motions that allow robots to avoid collisions with the obstacles around them. Moreover, the path must be feasible, which means that the configurations must be performable by the robot.

Motion planning is mostly about finding paths in continuous state spaces. Finding paths in continuous state spaces is a challenging task. Therefore, motion planners need to simplify their searches in order to return their solutions in an appropriate amount of time. There are several approaches to limit the number of states, such as limiting the search to a subset of the state space, reducing the number of samples by making a grid (*Graph-based methods*), or randomly sampling the state space (*Sampling-based methods*).

1.1.1 Graph-based Approaches

Graph-based methods create maps from continuous state spaces in order to have a finite number of states to check. There are several graph-based methods, such as Dijkstra (Dijkstra et al., 1959) and A* (Hart et al., 1968). These methods are able to return optimal solutions in the given resolution. Therefore, they are considered as *resolution-optimal* planners. They will return failure if there is no solution to the problem. Moreover, these algorithms guarantee to find a solution to a given problem, if one exists. Thus, they are also categorized as *resolution complete* planners.

Informed graph-based searches, such as A* (Hart et al., 1968), are able to find resolution-optimal solutions as well as guarantees on their efficient searches. Therefore, they are successful in many continuous planning problems. In spite of their benefits, they need to discretize the problem domain with a predefined resolution. Low-resolution graphs will lead to solving the problem quickly, but the result would be low-quality. High-resolution graphs will result in high-quality solutions (Bertsekas, 1975), but they may lead to complex graphs, which could be computationally expensive. This problem will be getting worse with the increase of the problem dimension and size, which is known as the *curse of dimensionality* (Bellman, 1954; Bellmann, 1957). In other words, graph-based algorithms have problems with scaling with the problem dimension and size (Gammell et al., 2014).

1.1.2 Sampling-based Approaches

In spite of graph-based methods, Sampling-based methods are not creating a priori graph from state spaces. They incrementally search the problem domain by taking random samples. This process increases the resolution incrementally until an appropriate solution is found. There are many well-known sampling-based approaches, such as Probabilistic Roadmaps (Kavraki et al., 1996) and Rapidly-exploring Random Tree (RRT) (LaValle, 1998).

Sampling-based methods are able to return solutions when the number of samples goes to infinity. This is the definition of *Probabilistic complete*. It means that the probability of returning a solution if one exists will go to one if the number of samples goes to infinity.

RRT is a single-query motion planner that explores state spaces by taking random samples and then grow its tree toward the random samples. It stops searching the state space once it could add a sample from the goal area to its tree. Although RRT could find solutions relatively fast, its solutions remain suboptimal (Karaman et al., 2011).

RRTs are *anytime* approaches, which means that they can be stopped at any time during

their exploration process and return their best-obtained results. However, if they are stopped before finding an initial solution, they will return no solution, which is not an appropriate result for anytime searches. Therefore, it is important to find an initial solution as fast as possible to be able to have at least a feasible path to return whenever the planner is stopped.

RRT* (Karaman & Frazzoli, 2011) is an asymptotically optimal version of RRT, which incrementally optimize its solutions. It has the unity probability of returning the near-optimal solutions when the number of samples goes to infinity. Planners like RRT* are called *almost-surely asymptotically optimal*. The accuracy of this method increases proportionally to the number of samples. In other words, RRT* is able to return optimal solutions, if the number of samples goes to infinity. Unlike RRT, RRT* will not stop exploring the state space after an initial solution is found. It keeps sampling the state space not only for finding other solutions but also for improving the existing solution with future samples.

RRT* could return near-optimal solutions. However, it is inefficient to sample all over the state space in order to improve the quality of the existing solution due to its nature, single-query planner (Gammell et al., 2014). It is better to shrink the state space into different areas and then prioritize them. Then, look through the areas based on their priorities. Many works have been published to solve this problem. They prioritize different parts of a state space, and then start sampling the areas that have more possibility to improve their solutions, such as RRT*-Smart (Nasir et al., 2013) and Tropistic RRT* (Wang et al., 2018). Although these planners could solve some problems faster than other RRT-based methods, their sampling methods are nonuniform. Nonuniform sample distributions divide a state space to different regions and search the region that has a higher possibility of having a solution. However, in many scenarios, the only solution may lie in the region with

lower possibility. Therefore, their reduced sampling probability can decrease performance or even preclude finding a solution.

Informed RRT* (Gammell et al., 2018, 2014) is a version of RRT*, which not only prioritizes the state space based on the cost of the existing solution but also keep uniformly sampling state spaces.

Informed RRT* acts similarly to RRT* before any solution is found. Unlike nonuniform sampling-based methods, Informed RRT* go through all over the state space to be able to find an initial solution by uniformly sampling the state space.

After finding an initial solution, Informed RRT* limits the search area into an ellipsoidal subset of the state space, which has the start and goal locations as its focal points, and its eccentricity is determined according to the minimum cost of the existing solution. This subset is an admissible estimation of all possible solutions better than the current one. In other words, there is no better solution than the existing one, which lies outside of the ellipsoidal subset. Limiting the state space to one of its areas helps the planner to be able to return near-optimal solutions faster than other planners that keep sampling the whole area of the state space.

1.2 Problem Statements

Although Informed RRT* can return near-optimal solutions faster than RRT*, it is required to find an initial solution to shrink the state space to a subset. Informed RRT* samples all over the state space to find initial solutions similarly to RRT* before an initial solution is found.

Informed RRT* can only expedite the optimization process, but not the process of finding initial solutions. It is because that Informed RRT* and RRT* act similarly before an initial solution is found.

Moreover, nonoptimized RRTs are expanding over state spaces faster than asymptotically

optimal RRTs such as RRT* and Informed RRT*. It is because that optimized versions run their optimization processes in each iteration, which are time-consuming procedures. Therefore, the optimization process makes optimized versions slower than nonoptimized versions, which do not need to spend time on optimization. The problem statements of this research are listed below:

- Unidirectional RRTs such as RRT* and Informed RRT* are relatively slow planners in terms of finding initial solutions.
- Asymptotically optimal versions RRT are taking time for optimizing processes, which makes them slow to find initial solutions.

1.3 Research Objectives

The objectives of this study can be identified as follows:

- To investigate the existing single-query sampling-based approaches.
- To design and develop an asymptotically optimal bidirectional RRT, which uses informed sampling after finding initial solutions.
- To design and develop a semi-bidirectional version of RRT that uses nonoptimized bidirectional RRT to find an initial solution and then uses asymptotically optimal unidirectional RRT for optimization.
- To compare the performance of the proposed methods with the existing ones.

The proposed methods should be more successful planners than state-of-the-art motion planners, such as RRT* and Informed RRT*, in terms of anytime capability.

1.4 Research Scope

This study focuses on single-query sampling-based methods. The proposed methods are new versions of RRT, which are asymptotically optimal RRTs. The proposed methods

will be developed in the Open Motion Planning Library (OMPL), which gives the ability to compare them with the existing methods.

OMPL App offers several scenarios, including 2D, 3D, and 6D. In order to show a comprehensive overview of the abilities of the proposed method in solving motion planning problems, scenarios from all types are selected for comparison of the proposed methods with state-of-art motion planners.

1.5 Overview of the Chapters

Chapter 2 provides the necessary background and literature review for the existing motion planners, including graph-based and sampling-based approaches. The research methodology is presented in Chapter 3.

Chapter 4 introduces Informed RRT*-Connect (Mashayekhi, Idris, Anisi, Ahmedy, & Ali, 2020), which is a bidirectional RRT-based motion planner. It has two trees, one rooted in the start location, while another is rooted in the goal location. Informed RRT*-Connect is an almost-surely asymptotically optimal motion planner, which focuses its search to a subset of the problem domain to be able to return near-optimal solutions faster than other optimized versions of RRT that look through all possible states after finding initial solutions.

Chapter 5 presents another almost-surely asymptotically optimal planner, Hybrid RRT (Mashayekhi, Idris, Anisi, & Ahmedy, 2020). Hybrid RRT uses a dual-tree search to be able to find an initial solution as fast as possible. It then combines these two trees into one with the aim of optimizing only one tree, which is faster than optimizing two trees. Chapter 6 provides the conclusion of the thesis.

CHAPTER 2: LITERATURE REVIEW

This chapter is a background of path planning, feasible paths, and optimal paths for robots in continuous state spaces (Section 2.1). Motion planning is about searching for a sequence of motions that robots could perform to go from one location to another while avoiding any collisions with the obstacles in their ways. A path is considered feasible when all its parts could be performable for the robot due to the different constraints. Optimal path planning is about searching for a path within several feasible paths, which minimizes a particular path cost function.

Motion planning problems are mostly about finding paths in continuous spaces. However, looking for a path in a continuous space will be complex and difficult. Therefore, motion planners normally discretize state spaces then searching through the states. Discretization helps planners have a limited number of states to check to release their outputs. The discretization of state spaces is mostly divided into two groups: graph-based methods (Section 2.2) and sampling-based methods (Section 2.3).

This work concentrates on motion planning problems that use sampling-based methods. It implements informed sampling (Gammell et al., 2018, 2014), which limits the search area to a subset in order to return solutions faster due to having a smaller number of states to be checked in comparison to the whole state space. This thesis implemented informed sampling on sampling-based methods, which provides two new motion planners that act better than state-of-the-art algorithms (Section 2.4).

2.1 Planning Problems

Motion planning problems in robotics is about finding paths in state spaces to allow robots to move from one location to another. The definitions of feasible motion planning and optimal motion planning are represented in Definition 2.1 and Definition 2.2, respectively.

Definition 2.1 ***The feasible motion planning problems.*** *Let $X \subseteq \mathbb{R}^n$ be the n-dimensional planning environment, $X_{obs} \subset X$ be the states that occupied by obstacles, and $X_{free} = cl(X \backslash X_{obs})$ be the set that are collision-free, where $cl(\cdot)$ represents the closure of a set. Let $x_{start} \in X_{free}$ be the start state that a robot is standing at the begging of planning and $X_{goal} \subset X_{free}$ be a set of states that the robot needs to reach there to accomplish its task. Let $\sigma : [0, 1] \rightarrow X_{free}$ be a sequence of states that robot could perform, and Σ be the set of all such nontrivial paths.*

The motion planning will be defined as looking for any solution from this set, $\sigma' \in \Sigma$, that connects x_{start} to $x_{goal} \in X_{goal}$,

$$\sigma' \in \{\sigma \in \Sigma \mid \sigma(0) = x_{start}, \sigma(1) \in X_{goal}\}.$$

Definition 2.2 ***The optimal planning problem.*** *The optimal motion planning is about searching among the feasible path to minimize a given path cost function, $\sigma^* \in \Sigma$.*

$$\sigma^* = \arg_{\sigma \in \Sigma} \min\{c(\sigma) \mid \sigma(0) = x_{start}, \sigma(1) \in X_{goal}\}.$$

Motion planning problems can be categorized into two groups, multi-query and single-query problems. Multi-query problem is about finding solutions between several start locations, x_{start}, and goal locations, x_{goal}. Multi-query problems use the same information to connect several points in the same search space together so that they often perform a calculation before performing any motions to obtain a map from the search space.

Single-query problems, on the other hand, are about finding a solution between a single start location and a single goal location so that it is not appropriate to create a map from the state space, which can connect almost all parts of the state space. It is necessary to find the solution as efficiently as possible without consuming the time for any preprocessing.

2.2 Graph-based Searches

Graph-based methods usually create a graph from the state space to have a limited number of states, *vertices*, that are connected via edges. Many robotics cost functions such as path length satisfy Bellman's principle of optimality (Bellman, 1954; Bellmann, 1957), and the resulting discrete problem can be solved with dynamic programming (Bellman, 1954; Larson, 1967).

Their performance is depended on the *a priori* discretization. If a graph-based algorithm guarantees to find a solution to a given problem at the chosen resolution, if one exists, it is considered as *resolution complete*. If the algorithm guarantees to find the optimal solution at a given resolution, if one exists, it is considered as *resolution optimal*. By definition, resolution optimality implies resolution completeness.

Dynamic programming techniques solve optimal planning problems by iteratively calculating vertices' costs. Some of the graph-based methods calculate costs between every pair of vertices in their graphs (Floyd, 1962; Ingerman, 1962; Roy, 1959; Warshall, 1962). They use this information to find optimal solutions, especially for multi-query problems. On the other hand, some other graph-based methods calculate the costs for a single vertex to every other vertex in the graph (Bellman, 1958; Ford Jr, 1956; Moore, 1959). These methods are used in multi-query problems that have a start location but several goal locations.

These two groups of graph-based methods keep searching all over the graph in order to return resolution-optimal solutions. However, this approach is inefficient for single-query problems. In other words, it is unnecessary to look through all over the graph to find only one path to connect a start location to a goal location. In such cases, the resolution-optimal solution could be found as soon as a solution is found, and all potentially better solutions are infeasible. These methods, *ordered searches*, guarantee that solutions would be found

only after checking all the possibly better solutions. Therefore, it avoids searching all over the graph to return resolution-optimal solutions for single-query motion planning problems.

Dijkstra's algorithm (Dijkstra et al., 1959) finds the resolution-optimal solutions by calculating the cost-to-come to vertices from the start location. Dijkstra's algorithm will stop searching once it found a solution. In other words, Dijkstra's algorithm finds resolution-optimal solutions by looking through the vertices with lower costs-to-come. This method creates a queue from the vertices sorted by cost-to-come. During each iteration, it removes the lowest-cost vertex in the queue and calculates the costs-to-come of the descendants of the removed vertex. This process will be continued until the goal is reached.

A* algorithm (Hart et al., 1968) avoids the low-cost vertices, which are not able to provide better paths. This method combines the cost-to-come of a vertex with its estimated cost-to-go, which is the estimated cost of going from a vertex to the goal. This heuristic allows A* to use the obtained information so as to find a resolution-optimal solution by checking vertices that potentially capable of providing better solutions.

2.2.1 Near-resolution-optimal Search

The algorithms like A* expand a large number of vertices to find a solution, which may not be suitable for many practical motion planning problems. It would be acceptable to reduce the time and effort of searching by relaxing optimality. An algorithm is called *near-resolution-optimal* if it could find a solution near the optimal solution if one exists.

Weighted A* (Pohl, 1970, 1973) is a version of A* that is near-resolution-optimal. It prioritizes the vertices that are closer to the goal to find solutions with fewer vertices. Another version of Weighted A* called Lazy Weighted A* (Cohen et al., 2015). This method postpones the cost calculation until it is necessary to reduce the number of

calculations and find solutions faster than the standard version of Weighted A*.

Multi-Heuristic A* (Aine et al., 2016) is another version of A* that is able to find near-resolution-optimal solutions by combining a single admissible heuristic with multiple arbitrarily inadmissible heuristics. This will provide some information to be incorporated into searches, including dynamically generated heuristics (Islam et al., 2015), while maintaining a bound on resolution quality.

These methods may be able to find solutions faster A* in some scenarios, but they are not able to return resolution-optimal solutions. Instead, they return near-resolution-optimal solutions.

2.2.2 Incremental Search

In many motion planning problems, it is normal that the environment changes during the planning time and/or motion execution. Therefore, A* has to restart planning to find a new solution for the recently changed environment. This is a drawback of A*, which results in extensive calculation whenever a single change occurred in the state space. In order to solve this problem, some methods have been proposed to use information from previous planning for the next round of planning. These methods are categorized as incremental searches.

Dynamic A* (Stentz, 1997) and derived methods (Ferguson & Stentz, 2005; Koenig & Likhachev, 2005; Stentz et al., 1995) are incremental searches that are suitable for motion planning problems in unknown environments. This method starts planning by considering the obstacle-free environment wherever it is unknown. Then, it will update the plan once it receives updated information about the unknown part of the environment. Therefore, the robot could plan and move in an unknown environment efficiently by performing a series of paths. Although each path is resolution-optimal, the whole path from the start to the goal is suboptimal.

Another incremental search A*-based method is Lifelong Planning A* (LPA*) (Koenig et al., 2004). This algorithm adapts A* to handle the graphs with changing edge weights. This method can adapt to changes without recalculating the whole graph. LPA* guarantees to be resolution completeness and resolution-optimality even some changes happen to the graph. LPA* uses the information of the unchanged part of the graph after some changes happen to the graph.

D* and LPA* maintain resolution optimality by spread changes through their searches, which leads to a time-consuming process. It would be better to decrease the planning time by relaxing optimality. Therefore, some other methods have been proposed, such as Truncated D* and Truncated LPA* (Aine & Likhachev, 2016). These methods find near-resolution-optimal solutions instead of resolution-optimal. As a result, they can reduce the planning time.

Although the methods like LPA* may need fewer vertices than A* to find a solution to a changeable environment, they are only able to return resolution-optimal paths. Moreover, in unchanged environments, they need the same number of vertices as A* to return their solutions.

2.2.3 Anytime Search

In many motion planning problems, planners have a limited amount of time to return solutions. A* method needs to have enough time to be able to find its solutions, which are the resolution-optimal ones. Therefore, before finding resolution-optimal solutions, it does not have any solution to return. As a result, this characteristic is not acceptable in many applications in which planners must be able to return the best possible solution in a limited time. In order to achieve this goal, planners should be able to find an initial solution which is suboptimal and then spend the remaining time of planning to improve the quality of the path. Such methods would be able to stop at any time to return an intermediate solution.

These methods are called *anytime searches*.

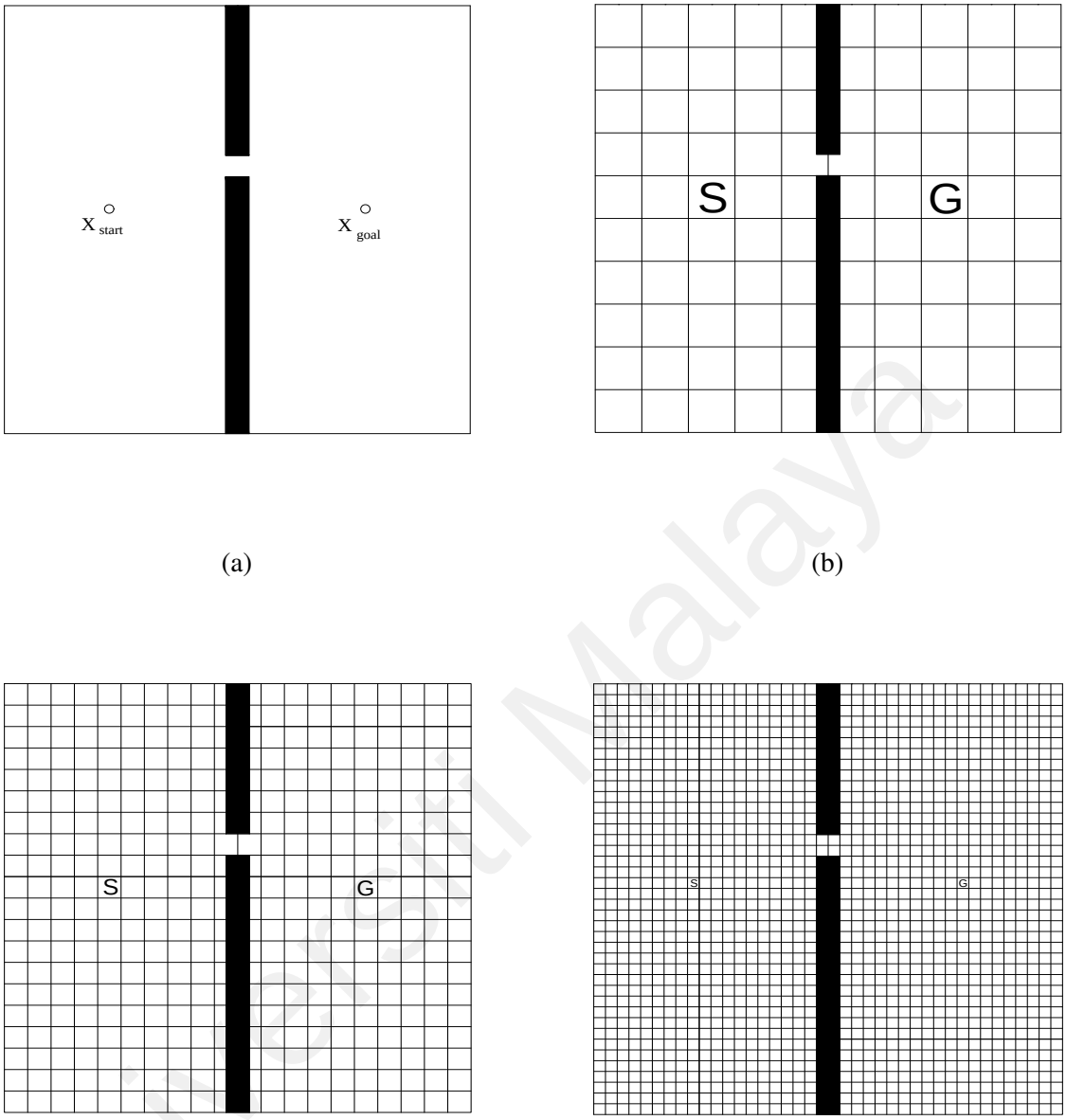
Some variations of A* have been proposed, combining near-resolution-optimal and incremental search techniques to overcome this limitation. Anytime Repairing A* (Likhachev et al., 2008, 2004) and Anytime Dynamic A* (Likhachev et al., 2005) are two examples of this combination. They first find an initial suboptimal solution and then try to improve the quality of their solutions. These methods not only able to return resolution-optimal solutions eventually but also can return near-resolution-optimal solutions whenever the planning time is finished.

Although these methods can find initial suboptimal solutions with fewer vertex expansion in comparison to A*, they need more vertices to be able to return resolution-optimal solutions than A*.

2.2.4 The problem of graph-based methods

The performance of graph-based methods is heavily related to the accuracy of the approximation graphs that they generate from state spaces. There have been many works on this part to obtain different types of techniques from the discretization of a continuous space into a graph. These methods include regular graphs (Lozano-Pérez & Wesley, 1979), graphs with random perturbations (Sallaberger & D'Eleuterio, 1995), graphs with Kinodynamic velocity and acceleration constraints (Donald et al., 1993), hierarchical, nonuniform, and multi-resolution graphs (Chen & Hwang, 1998).

In order to discrete the environment, a resolution required to be defined. Figure 2.1 shows a continuous space, which discretized by several resolutions. If a low-resolution graph is selected for discretization, it then finds the solution quickly, but it would be a low-quality solution. If a high-resolution graph is selected, the motion planner can solve the problem accurately (Bertsekas, 1975), but it needs a complex graph that is generally expensive to be explored. These effects would be intensified by the state dimension and



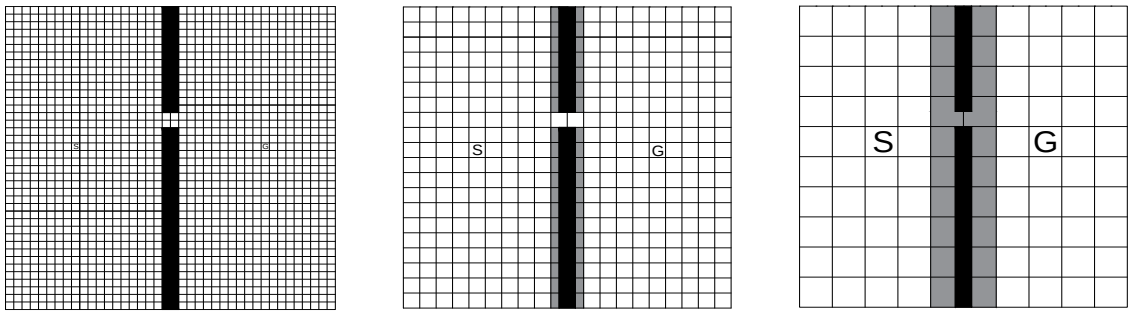
(a) (b)

(c) (d)

Figure 2.1: (a) The continuous state space, (b), Low-resolution graph, (c), Appropriate resolution graph, (d), High-resolution graph.

the graph size (Bellman, 1954).

Figure 2.2 shows the obstacles that mapped in different graph resolutions. It can be seen that the problem offers solutions with the high-resolution (Figure 2.2a) and an appropriate resolution graphs (Figure 2.2b), while for low-resolution graph (Figure 2.2c), it does not



(a) (b) (c)

Figure 2.2: (a) High-resolution graph has solutions for graph-based methods, (b), Appropriate resolution graph has offered solution, (c), Low-resolution graph provides no solution for this problem.

have any solution.

As a result, one of the most critical decisions of graph-based planners is to select an appropriate resolution, which should be low enough to be searched quickly as well as high enough to provide an acceptable result. The appropriate resolution is different from one space to another so that it is a time-consuming process to find a proper resolution for a specific problem.

2.3 Sampling-based Planners

Sampling-based approaches are other methods to solve continuous planning problems, which are simpler than graph-based methods in many scenarios. The accuracy of these methods will be increased with more samples they take. The anytime ability let them avoid a priori discretization as well as reducing the curse of dimensionality. They are also able to improve the performance of systems with differential constraints (LaValle & Kuffner Jr, 2001).

These methods are often categorized as probabilistic complete, which means that the probability of a finding a solution goes to one, if one exists, and if the number of sampler goes to infinity (Definition 2.3).

Definition 2.3 ***Probabilistic completeness.*** *Motion planners will be considered probabilistically completeness, when with an infinite number of sampling, the probability of finding a solution goes to one, if one exists.*

$$\liminf_{q\to\infty} P(\sigma_q \in \Sigma, \sigma_q(0) = x_{start}, \sigma_q(1) \in X_{goal}) = 1,$$

where q is the number of samples, σ_q is the path that the planner returned by using all the samples, and Σ is the set of all collision-free and feasible paths.

Definition 2.4 ***Almost-sure asymptotic optimality.*** *Motion planners will be considered Almost-sure asymptotic optimal, when with an infinite number of sampling, the probability of converging asymptotically to optimum, if one exists.*

$$P\left(\limsup_{q\to\infty} c(\sigma_q) = c(\sigma^*)\right) = 1,$$

where q is the number of samples, σ_q is the path that the planner returned by using all the samples, and σ^* is the optimal solution of the planning problem, and $c(\cdot)$ is the path cost.

Sampling-based methods are very popular due to their ability to solve high-dimensional problems faster than graph-based methods. The sampling-based methods are usually categorized into two groups, *Multi-query problems*, and *Single-query problems*.

2.3.1 Multi-query problems

If a motion planning problem has several starts and goal locations, while the robot and obstacles are not changed, it leads to multi-query planning problems.

In this case, a *roadmap* is required to be obtained from the problem domain in order to be used several times. The roadmap should be able to solve several multiple-goal queries efficiently. Therefore, before solving the problem, a roadmap must be created from the problem domain. As a result, a time before start finding solutions is required to be specified for creating the roadmap, which is called *preprocessing phase*.

After creating an appropriate roadmap from the problem domain, it is time for *solving phase*. In this phase, planners receive several starts and goal locations, and then they attempt to connect them to their roadmaps. Once a set of start and goal locations is connected to the roadmap, it is solved. The general framework presented here was mainly introduced in (Kavraki et al., 1996) under the name probabilistic roadmaps (PRMs).

PRM divides the planning time into two phases: *learning* and *query* phases. In the learning phase, it makes a roadmap from the state space by sampling it and then connect them if they are collision-free. The second phase, the query, is about solving a specific problem by using the roadmap information. In this phase, the planner has a start location and goal location, which must be connected to the roadmap. Once the connections between the start location and the goal location with the roadmap are found, the planner can connect them via the roadmap and then release the path. PRM is a probabilistically complete motion planner (Kavraki et al., 1998), and with an appropriate connection, the scheme is also almost-surely asymptotically optimal (Karaman & Frazzoli, 2011).

The obtained roadmap can be reused for many times as needed to solve the different problems with different start locations and goal locations. It is due to that the roadmap has already made a graph from the collision-free portion of the state space. This characteristic makes PRM a suitable motion planner for multi-query motion planning problems.

Although PRM can work efficiently in multi-query planning problems, it is an inefficient method for single-query motion planning problems. In order to alleviate this problem, some versions of PRM such as Lazy PRM (Bohlin & Kavraki, 2000), and Quasi-random Lazy PRM (Branicky et al., 2001) have been proposed, which try to reduce the computational cost of the unused roadmap. However, it is not as efficient as incremental searches that continue their search until a solution is found.

2.3.2 Single-query problems

Single-query planning problems include a set of single start and single goal locations. For solving single-query problems, planners are required to search the problem domain until an appropriate solution is found, if at least one exists. Otherwise, it needs to return failure, if no solution exists, or time of planning is over.

Most single-query sampling-based motion planning algorithms grow a tree from the start locations and try to expand it toward random samples.

2.3.2.1 Rapidly-exploring Random Trees (RRTs)

RRT solve motion planning problems by growing a tree from the start location and expanding it toward random samples. The random sampling guides the tree to the unexplored portion of the planning problem domain through Voronoi bias (Voronoi, 1908). RRT returns the solution once it could sample the goal area. RRT is a probabilistically complete planner. RRT-based motion planners' algorithms have several common steps, which are listed below:

1. **Initialization:** In this step, the planner adds the start location to an empty tree as its first *vertex*.
2. **Sampling method:** A sample from the state space needs to be taken in order to grow the tree toward it.
3. **Local planning method:** In this step, the planner needs to find an appropriate vertex in its tree to connect it to the sampled point. Moreover, it should take several considerations into account, such as the feasibility of the connection.
4. **Insert an edge in the graph:** After finding an appropriate connection to the sample point, it is time to add it to the tree.
5. **Check for a solution** This step checks whether an appropriate solution is found. An

Initial solution will be found once a sample within the goal area has been added to the tree. If an initial solution is found, the algorithm will stop checking the state space and returning the solution.

6. **Return to step 2**

RRT (LaValle, 1998), has shown an outstanding performance in practice. There are several types of RRT:

- Nonoptimized RRTs
- Asymptotically Optimal RRTs
- Unidirectional RRTs
- Bidirectional RRTs

2.3.2.2 Nonoptimized RRT

Nonoptimized versions of RRT have no procedure to optimize the exploring tree. Their goal is to find a feasible path that connects the start location to the goal location. These planners normally start exploring state spaces and stop once an initial solution is found. There have been several proposed nonoptimal RRTs such as the standard version of RRT (LaValle, 1998; LaValle & Kuffner Jr, 2001), RRT-Connect (Kuffner & LaValle, 2000), Safe-RRT (Pepy & Lambert, 2006), RRT-Plan (Burfoot et al., 2006), Particle RRT (Melchior & Simmons, 2007), Transition-based RRT (Jaillet et al., 2008), T-RRT (Aguinaga et al., 2008), Metric Adaptive RRT (Lee et al., 2008), Closed-loop RRT (Kuwata et al., 2008, 2009), and Retraction-Based RRT (Pan et al., 2010). This type of RRT can explore state spaces efficiently. However, their solutions remain considerably nonoptimal.

2.3.2.3 Asymptotically optimal RRT

Asymptotically optimal RRTs include a process that improves the quality of their solutions. The accuracy of their solutions increases proportionally to the number of

samples. In other words, by increasing the number of samples, planners are able to produce better solutions. Therefore, the optimized versions of RRT keep searching the state space after a solution is found in order to find asymptotically optimal solutions. Some of the well-known asymptotically optimal RRTs are RRT* (Karaman et al., 2011), BT+RRT* (Perez et al., 2011), Kinodynamic RRT* (Webb & Van Den Berg, 2013), RRT$^{\#}$ (Arslan & Tsiotras, 2013), T-RRT* (Devaurs et al., 2015), BIT* (Gammell et al., 2015), Lower Bound Tree-RRT (Salzman & Halperin, 2016), HARRT* (Qureshi & Ayaz, 2016), SST (Li et al., 2016), RABIT* (Choudhury et al., 2016), Informed RRT* (Gammell et al., 2018, 2014), RRT* GL (Aguilar et al., 2017), and Neural RRT* (Wang et al., 2020).

Although asymptotically optimal versions of RRT are able to return near-optimal solutions, they are slower than the nonoptimized versions due to the time required for the optimization process in each iteration.

2.3.2.4 Unidirectional RRT

Unidirectional RRTs explore state spaces by a single tree. This tree is rooted in the start location, and then it grows toward the samples. An initial solution will be found, once a sample that is located within the goal area is added to the tree. As a result, this sample can connect the goal area to the start location. There are many Unidirectional RRTs such as RRT (LaValle, 1998; LaValle & Kuffner Jr, 2001), RRT* (Karaman et al., 2011), RRT*-Smart (Nasir et al., 2013), Informed RRT* (Gammell et al., 2018, 2014), RRT* FN (Tong et al., 2019), and Optimized RRT-A* (Ayawli et al., 2019).

Unidirectional RRTs may take a long time to have a vertex located within the goal area, especially when the goal is hidden beyond several narrow passages, such as bug-trap-like state spaces.

2.3.2.5 Bidirectional RRT

Bidirectional RRTs implement two trees for exploring state spaces, one tree is rooted in the start location, while another one is rooted in the goal location. In each iteration, a sample is taken, then the planner grows one of its trees toward the sampled point. If the sample can be added to the tree, then the newly added vertex will be considered as a sample for another tree. Then, the planner swaps the trees for the next iteration. An initial solution will be found, once a connection between two trees are established. There are several bidirectional RRTs including RRT-Connect (Kuffner & LaValle, 2000), RET (Martin et al., 2007), CellBiRRT (Fragkopoulos & Graeser, 2010), RRT*-Connect (Klemm et al., 2015), Intelligent Bidirectional-RRT* (Qureshi & Ayaz, 2015), Bidirectional Spline-RRT* (Sudhakara et al., 2017), Bidirectional Potential Guided RRT* (Xinyu et al., 2019), and Goal-biased Bidirectional RRT (Liu et al., 2019).

2.3.2.6 RRT*

RRT takes a random sample from the state space at each iteration. It then uses this sample to expand its tree toward the sample. RRT has a predefined parameter, *RRT step*. The planner finds the nearest tree vertices to the sample, and then if the distance between the nearest vertex and the sample is lesser than the RRT step and it is collision-free, it will be added to the tree as a new vertex. If the distance between the sample and the nearest vertex is bigger than the RRT step, then another point in the direction of the sample but with an RRT step away from the nearest vertex will be added to the tree. An illustration of the RRT algorithm is presented in Figure 2.3. Although RRT is a simple single-query planner, in practice, it has a small likelihood of directly sampling the goal to bias the search toward a solution.

To solve the problem of sampling the goal area, Kuffner and LaValle proposed another version of RRT, RRT-Connect (Kuffner & LaValle, 2000). RRT-Connect is a bidirectional

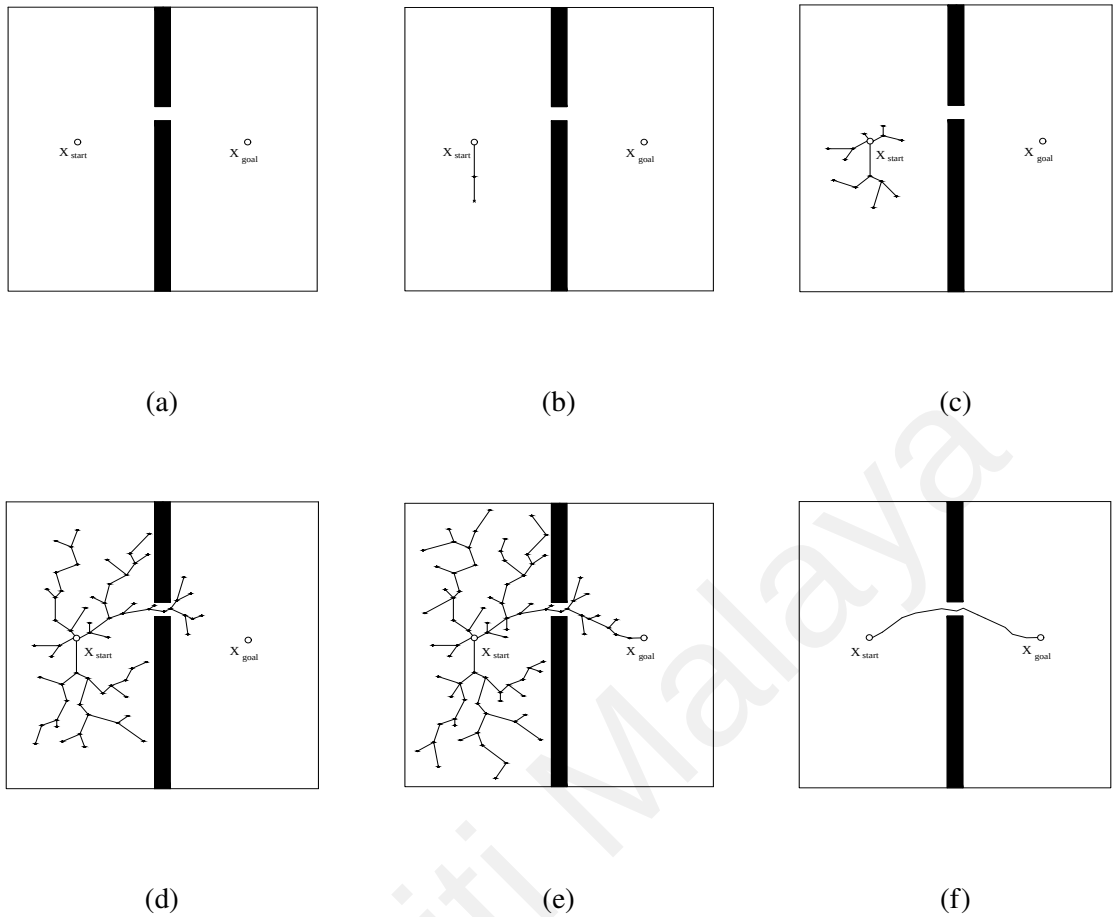


Figure 2.3: (a) The state space in which RRT needs to find a solution. (b) A sample is taken but it is far from the x_{start}; therefore, another point in the same direction with one *RRT step* away from x_{start} is added. (c)(d) The expansion of RRT tree in the state space. (e) The tree could sample X_{goal}. (f) The solution, which RRT could find for this problem.

version of RRT, which has two trees for exploring the state space. These trees are rooted in the start location and the goal location. They are growing alternately toward the newest taken sample and the newest vertex in the other tree. RRT-Connect stops searching once a connection between the two trees is established. Having two trees helps RRT-Connect be able to solve many problems faster than RRT. Similarly to RRT, RRT-Connect is probabilistically complete, and it is not almost-surely asymptotically optimal.

RRT and RRT-Connect can expand their trees quickly into unexplored regions of state spaces. However, during the expansion, the costs-to-come of vertices is not going to be changed. Therefore, the solutions provided by them are suboptimal (Karaman & Frazzoli,

2011). RRT solution quality can be improved by modifying existing connections during the search to improve the costs-to-come of vertices incrementally.

RRT* and Rapidly-exploring Random Graph (RRG) (Karaman & Frazzoli, 2011) change RRT to check incoming and outgoing edges of new vertices in order to improve the costs-to-come of existing vertices. RRG uses all these edges to build a graph while RRT* finds those that locally minimize cost-to-come and maintains a tree. Both algorithms are probabilistically complete and almost-surely asymptotically optimal (Karaman & Frazzoli, 2011).

RRT* implements a local rewiring method in order to optimize its tree. When it gets a sample, it will connect it to the nearest vertex, which has the lowest cost-to-come in comparison to other near vertices. Then, the newly added vertex will be considered as a potential parent for nearby vertices. In other words, if near vertices can reduce their costs-to-comes by replacing their parents with the newly added vertex, then they will switch their parents. This method helps the existing vertices to benefit from future sampling.

There is a parameter that influences the performance of RRT*, which is the neighborhood size. The neighborhood should be small enough in order to be easily searched as well as large enough to be able to improve the tree quality. Karaman and Frazzoli (Karaman & Frazzoli, 2011) proposed two different methods for defining the neighborhood, one is k-nearest vertices, and another is the vertices that are located within a distance, r.

These incremental searches have the ability to search the continuous planning problems and stop when an appropriate solution is found. Therefore, they avoid unnecessary computational efforts. However, the methods such as RRT* will find asymptotically optimal solutions to every state in the state space so that it is an inefficient way due to their single-query nature. In order to solve this problem, several methods have been proposed, such as different sampling methods, ordered approximation, and ordered searches. These

methods could improve performance, but they do so without addressing all the challenges of simultaneous approximation and searching for a continuous planning problem.

2.3.2.7 Sample Distributions

Sampling-based methods explore state spaces by taking samples from them and then connecting the samples together. Therefore, their performance is affected by the methodology of sample distributions. There are several sample distributions such as uniform distribution (Karaman & Frazzoli, 2011), which are easy to analyze and appropriate for many scenarios. However, in some scenarios, it is better to sample the state space sections based on their prioritizes, which help improve the real-time performance of the motion planner. These types of sampling are known as nonuniform distributions.

There are many sampling distributions, such as:

- **Medial axis:** This sampling method increases the probability of sampling around the Voronoi graph so as to guide the roadmap generation to capture the shape of state spaces (Guibas et al., 1999; Holleman & Kavraki, 2000; Wilmarth et al., 1999).
- **Boundary:** The sampling is guided toward obstacles boundaries as opposed to collision-free part of the state space (Amato & Wu, 1996).
- **Gaussian:** This method is acting similarly to the boundary method, which means that it increases the probability of sampling near obstacles. In other words, the vertices are scattered according to obstacles and collision information (Boor et al., 1999).
- **Bridge-test:** This sampling method has been proposed to improve the ability of sampling-based methods in sampling narrow passages. It detects narrow passages by taking to configurations from the state space as well as their midpoint. If both configurations are located within the obstacle-occupied portion of the state space

and the midpoint is in the collision-free part of the state space so that a narrow passage is detected (Z. Sun et al., 2005).

- **Hybrid:** This sampling is about combining different sampling methods to benefit from their upsides. Adaptive hybrid sampling (Hsu & Sun, 2004) combines narrow passage detection with uniform sampling to be able to increase sampling probability in narrow passages. Uniform sampling allows it to maintain Randomization, which is an advantage in solving several planning problems. Another version of hybrid sampling has been proposed, which combines the Medial axis and narrow sampling (Thomas et al., 2007).
- **Goal Biasing:** This method tries to connect the goal location to the tree. It works in the way that after getting several random samples, the goal location will be considered as a new sample. Therefore, there are some predefined parameters, such as the number of random samples in each iteration before considering the goal configuration as a sample.

There are some other methods that guide the exploration, including:

- Dynamic-Domain RRT (Yershova et al., 2005), this method limits the sampling domain of the boundary nodes to a predetermined radius ball to consider it as an alternative to the visibility region.
- Adaptive Dynamic-Domain RRT (Jaillet et al., 2005), ADD RRT is an updated version of DDT RRT, in which the planner decides the size of the ball's radius according to the extension success rate of each boundary node.
- Utility-RRT (Burns & Brock, 2007), this method influences the direction and length of extension.
- Obstacle-based RRT collects obstacles information and then select the growth direction based on the gathered information (Rodriguez et al., 2006; Yeh et al.,

2012).

- There are some other proposed techniques that sample the different sections of the state space by prioritizing them (Akgun & Stilman, 2011; Jaillet et al., 2008; Kim et al., 2014; Nasir et al., 2013; Zucker et al., 2013).

Although nonuniform sampling techniques could solve some scenarios faster, it may not be able to find useful solutions in many scenarios. For example, if the solution lies in undersampled areas so that these methods may not even find a solution to the problem.

In order to solve this issue, nonzero sampling methods have been proposed, which is the combination of nonuniform and uniform sampling methods. It helps planners guarantee their probabilistic completeness. However, it will result in reducing nonuniformity by increasing sampling from the less prioritized areas of state spaces. This ratio is a user-defined parameter, which can vary from one problem to another.

Nonuniform sampling is affecting the almost-surely optimality of planners. Planners such as RRT* proof their optimality and rewiring neighborhoods based on uniform sampling so that violating this will invalidates their claim about being almost-surely optimal planners.

2.3.2.8 Heuristic-based Approaches

Sampling-based methods incrementally search state spaces and discretize them by random samples. They are able to stop the search anytime and return their best-obtained results. In other words, they can keep exploring the state space until the desired solution is found. In single-query motion planning problems, it is inefficient to keep searching all over the state space to be able to return the desired solution.

As a result, other motions planners have been proposed, using heuristic concepts to reduce the computational complexity of optimizing the obtained solution. Heuristically

Guided RRT (Urmson & Simmons, 2003) implements a method for sample rejection so as to simulate the heuristic search concept in RRT-based algorithms. This planner accepts or rejects the samples based on their values relative to the obtained tree. This method tries to scan a portion of the state space that has more changes for high-quality solutions. Moreover, it is often able to return better solutions faster than the standard version of RRT.

Several other techniques incrementally create trees by using some heuristics, such as Randomized A* (Diankov & Kuffner, 2007), Hybrid Randomized A* (Teniente & Andrade-Cetto, 2013), and sampling-based A* (Persson & Sharf, 2014). These methods are prioritizing state spaces based on their heuristics, aiming to first sampling the problem domain regions in which high-quality solutions are more likely to be found.

Using heuristics helps predict potential high-quality solutions. However, there is a challenge to balance between the solution quality and time of planning. This problem can be solved in many ways such as defining minimum sampling probability for each region (e.g., hRRT (Urmson & Simmons, 2003)), a maximum local sample density (e.g., RA* (Diankov & Kuffner, 2007) and HRA* (Teniente & Andrade-Cetto, 2013)), and an inverse correlation between local sample density and expansion priority (e.g., SBA* (Persson & Sharf, 2014)).

The minimum sampling probability approach searches for multiple independent solutions without considering their costs by allowing exploratory states to be sampled in each iteration. Unlike A*, methods such as hRRT, HRA*, and SBA* may keep searching other regions of the problem domain even when there is a possibility to be connected to the goal, which is an inefficient approach.

Maximum local sample density, on the other hand, expediting the planning process by limiting the number of children for the vertices. However, it will affect the anytime ability of planners in comparison to RRT (LaValle, 1998), which could sample the state space

until an appropriate solution is found. RA*-like methods get samples from the state space until a priori maximum resolution has reached, and then they restart their planning process if the resolution is insufficient.

These methods that use heuristics to limit the problem domain. They have, however, several parameters to be defined by the user. Consequently, their performance will vary from one scenario to another and/or from one user to another.

Gammell *et al.* (Gammell et al., 2014) introduced the subset of the problem domain, depending on the path length of the obtained solution. This subset is a function of the obtained solution cost. It is proven that better solutions than the existing ones are lying within the subset if there is any. Therefore, it is needed to explore the subset to find better solutions than the existing one. This subset cannot be obtained without having an initial solution. Moreover, this subset does not have any parameter to be tuned by a user. They proposed Informed RRT* (Gammell et al., 2018, 2014), which implements this subset. Although this planner can limit the problem domain to a subset, it requires finding an initial solution to focus its searching in a subset of the problem domain. Moreover, Informed RRT* is not the fastest RRT-based method in terms of finding initial solutions. In fact, there are several versions of RRT, which can establish their initial solutions faster than Informed RRT*.

This thesis seeks to efficiently implement informed sampling that benefits from uniform sampling on other RRT-based methods and combines their abilities so as to propose better motion planners, which are able to return near-optimal solutions faster than the-state-of-arts algorithms.

2.4 Discussion

Motion planning is one of the fundamental tasks for autonomous robots that need to plan their movement to accomplish their duties without having any collision with other

nearby obstacles. Therefore, a successful motion planner can be defined as a planner that is able to provide near-optimal and feasible (e.g., collision-free) solutions in a short period of time. Almost all planners make either a graph from the problem domain or randomly sample the problem domain. Each approach has its pros and cons.

Graph-based methods, which create a graph from the state space before start planning, need to have a resolution for their graphs. If the resolution is selected too low, then it may lose solutions. If the resolution is selected too high, it will be a time-consuming process to check all possible states. Therefore, the resolution is a problem-specific parameter that needs to be defined based on the problem requirements.

In spite of their limitations, graph-based methods are effective in many scenarios due to the efficiency of their searches. Informed graph-based searches, such as A*, search in order of potential solution quality. Moreover, they are optimally efficient in the number of expanded vertices (Hart et al., 1968). Therefore, they are appropriate choices for the problems in which an appropriate resolution can be selected.

On the other hand, sampling-based methods approximate the problem domain by taking a random sample from it. Their approximation can improve by getting more samples and more connections between samples. However, having many samples and their connections can lead to expensive computation to solve the problem. Although sampling-based methods avoid a priori discretization of the problem domain, they need to expand their samples incrementally. Searching all over the problem domain gives the ability to find a solution from the start location to any other vertices in the tree, but it is computationally expensive and inefficient for single-query planning problems.

In spite of their limitations, they are efficient due to their anytime characteristics. Incremental sampling-based methods, such as RRT*, can return almost-surely asymptotically optimal solutions in an anytime manner (Karaman & Frazzoli, 2011). Consequently, they

are suitable choices for planning problems that have an unknown amount of time for finding an appropriate solution without choosing any resolution.

Several works combine sampling-based with graph-based to benefit from the advantages of both approaches. Informed RRT* (Gammell et al., 2018, 2014) implements a heuristic to limit problem domain to a subset of it in order to expedite the process of returning near-optimal solutions. However, this subset can be formed after an initial solution is found. Informed RRT* is not a fast planner in terms of finding initial solutions due to its nature, which is a unidirectional search, and it includes the rewiring process, which consumes time.

This thesis proposes two new motion planners (Informed RRT*-Connect and Hybrid RRT) that shrink the problem domain so as to limit the search area and return near-optimal solutions in less time compared with other state-of-the-art motion planners.

Informed RRT*-Connect firstly implements a dual-tree version of RRT*, RRT*-Connect, to be able to find initial solutions faster then RRT* and then limit the search area to a subset according to the length of obtained solutions. Informed RRT*-Connect works faster than Informed RRT*.

Hybrid RRT is another motion planner that works faster than Informed RRT*-Connect in terms of finding initial solutions. It divides the planning time into three different phases. The first phase is to find an initial solution. In order to find the initial solution as fast as possible, Hybrid RRT implements RRT-Connect, which is a dual-tree version of RRT. Therefore, it can find initial solutions faster than RRT. Moreover, RRT-Connect does not have any procedure for optimizing the tree so that it helps the planner return initial solutions faster than planners that include the optimization process. The second phase is to combine the two trees of phase one into one tree so as to make it easier and faster to optimize. The third phase is to optimize the obtained tree from phase two and keep

searching the problem domain until the desired solution is found.

CHAPTER 3: METHODOLOGY

This chapter introduces the methodology of the presented research. The methodology of this research has been divided into four different phases, which are mentioned below:

- The Literature Review
- The First Proposed Method
- The Second Proposed Method
- The Evaluation

These four phases have been thoroughly discussed in Section 3.1.

3.1 Methodology

The research workflow has been illustrated in Figure 3.1, which includes four main phases. Phase one (Section 3.1.1) is the literature review. Phase two (Section 3.1.2) is about the first proposed method, which is a bidirectional RRT*-based method that implements informed sampling after an initial solution is found. In phase three (Section 3.1.3), another RRT-based method has been proposed. This method combines all types of sampling-based methods categories to benefits from their advantages. Finally, phase four (Section 3.1.4) is about evaluating the proposed methods by comparing them with the existing methods as well as comparing them with each other.

3.1.1 Phase One *(Literature Review)*

A literature review of motion planning has been carried out in this phase. Motion planning is a crucial task of robotic systems operating in unknown environments. Any robot that needs to accomplish a task by changing its position from one configuration to another needs to have a motion planner to provide a feasible and collision-free set of configurations. This path could guide the robot from its initial configuration to the

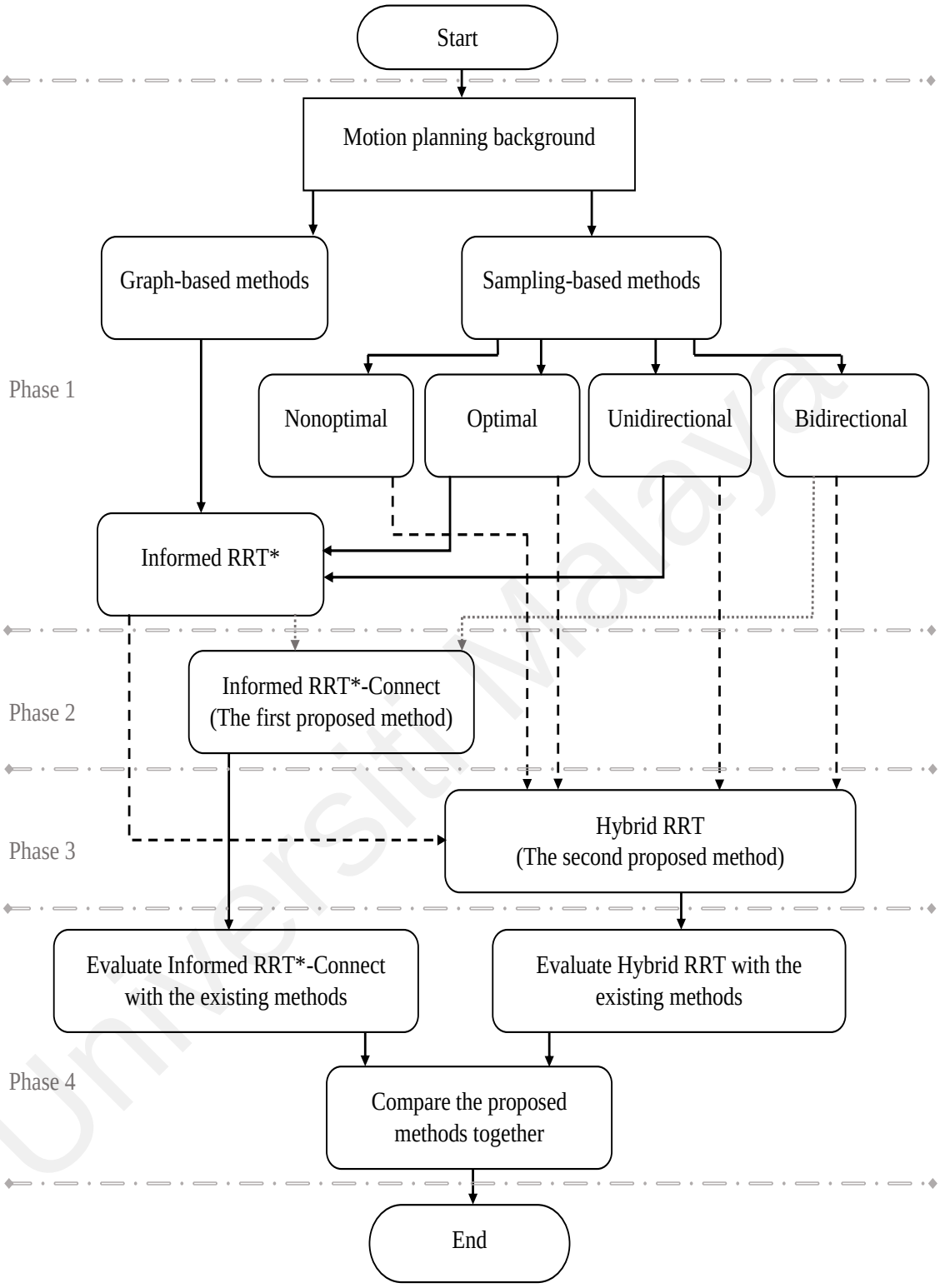


Figure 3.1: Workflow of the study

target configuration. Different types of motion planners offer various paths for a particular motion planning problem. Therefore, it is necessary to have some criteria to compare their performances against each other. Some of these criteria are the planning time and the cost of the path. Successful motion planners can return high-quality paths (e.g. low-cost paths) in a reasonable amount of time.

Most motion planning problems are about finding paths in continuous state spaces. Therefore, planners have infinite states to check so as to release their solutions. In order to solve this problem, planners discretize state spaces by either making a priori graph or random sampling.

Using a priori discretization requires defining the graph resolution before performing the search. Creating a predefined graph from a state space makes the problem easier to be solved. However, finding an appropriate resolution is not an easy task to be accomplished. If the chosen resolution is too low, then obtained solution will have an insufficient quality. Moreover, some narrow passages may not be found due to low resolution. On the other hand, if the chosen resolution is too high, then the output would be high-quality. However, the resulting representation will be prohibitively expensive to search in. Therefore, it is challenging to find an appropriate resolution, which could be high enough to be able to return high-quality solutions, as well as low enough to be searched quickly. The appropriate resolution is problem specific. In other words, it is different from one problem to another. Therefore, it is not possible to define the same resolution for a wide range of problems. It means that the planners need to have a new resolution for each problem that they are going to solve. As a result, it is a time-consuming process to find an appropriate resolution for each motion planning scenario. Another problem with graph-based methods is that their efficiency diminishes with the increase of problem dimension.

Sampling-based methods, on the other hand, avoid defining a priori approximations.

They usually discretize state spaces by random samples. Their resolutions will be increased by the increase in the number of samples accordingly. There are two types of sampling-based methods, multi-query, and single-query. Multi-query planning problems require to solve several problems in a state space. Therefore, they normally create a roadmap from the state space and then try to connect different parts of the state space by going through the roadmap. For that reason, they need to have a learning phase in which they create their roadmaps from state spaces. After creating the roadmap, the planner receives several start locations and goal locations. It then tries to connect these locations to its roadmap. Once these locations are connected to the roadmap, they are connected together via the roadmap. Single-query problems, On the other hand, are about finding a path between a start location and a goal location. It means that single-query planners are not required to create a roadmap from the state space to solve several problems. As a result, they can avoid spending time on the learning curve. They normally expand a tree from the start location and stop searching once they can get within the goal area. This thesis focuses on single-query motion planning problems.

Among single-query planners, RRT has shown a significant improvement over other single-query planners due to its success in practice. RRTs can be categorized into four groups, nonoptimal, asymptotically optimal, unidirectional, and bidirectional. Nonoptimal versions are designed to find initial solutions. They do not have any procedure for optimizing their solutions so that they can be expanded rapidly over the collision-free portion of the state space. However, their solutions are suboptimal. On the other hand, asymptotically optimal versions improve the quality of their paths by rewiring the edges of their trees. In each iteration, they first get a sample and then expand their trees toward the sample, and then they call the rewiring process to optimize the trees around the newly added tree vertex. Unidirectional RRTs grow a tree from the start location and expend it

toward random samples. Their initial solution is found when they could sample the goal area. Although they can rapidly explore state spaces, it may spend time to sample goal areas hidden beyond narrow passages. Bidirectional RRTs grow two trees simultaneously, one from the start location, and another from the goal location. They try to make a connection between their two trees to release solutions. A random sample is taken in each iteration; then, the planner expands one of its trees ($Tree_a$) toward it. If the planner could add a new vertex to $Tree_a$, it then tries to make a connection between its two trees, by considering the newly added vertex of $Tree_a$ as a sample for its other tree ($Tree_b$). At the end of each iteration, the planner swaps its two trees. In other words, it gets a sample and then expand $Tree_b$ toward it. Afterward, it considers the newly added vertex of $Tree_b$ as a sample for $Tree_a$.

The performance of RRT-based methods is dependent on their sampling techniques. Most of them use unordered sampling methods. Unordered sampling-based planners simultaneously search for every state in the problem domain. This becomes prohibitively expensive as state dimension increases and eventually outweighs the advantages of anytime representations in incremental sampling-based planners. In order to solve this issue, many methods have been proposed that use heuristics to improve the anytime capability of their methods. However, they could achieve it by using nonuniform sampling approaches.

Informed RRT* uses heuristics to improve sampling-based planners without sacrificing the benefits of either informed search or anytime approximation. It does so by carefully applying techniques from graph-based searches to the sampling-based method. However, Informed RRT* needs to have an initial solution to be able to shrink the state space into a subset. In other words, it works like other unordered sampling-based methods before finding an initial solution. Informed RRT* could limit the state space into one of its subsets only after finding an initial solution. Before finding initial solutions, Informed

RRT* works like RRT*.

3.1.2 Phase Two *(The First Proposed Method, Informed RRT*-Connect)*

In this phase, the first proposed method, Informed RRT*-Connect, has been designed. This planner is an asymptotically optimal version of RRT. It is a bidirectional RRT, which helps it find initial solutions faster than unidirectional asymptotically optimal versions of RRT.

Informed RRT*-Connect starts exploring state spaces by implementing two trees, one from the start location and one form the goal location. Informed RRT*-Connect keeps rewiring its two trees so as to improve their qualities. After finding an initial solution, the planner limits the state space into a subset, which includes states that could potentially improve the quality of the current solution. Limiting the state space into a smaller space helps planners to improve its solution quality in lesser time. Informed RRT*-Connect will stop searching the state space when either an appropriate solution is found or planning time is over. Informed RRT*-Connect has three main parts that are listed below:

1. The planner starts exploring state spaces by two RRT* trees, one rooted in the start location and another one in the goal location.
2. After an initial link between the two trees is found, the planner must calculate the solution cost, then limits the state space to a subset to be able to return near-optimal solutions faster. The size of the subset is defined based on the current shortest path.
3. At the end of each iteration, the minimum cost must be recalculated to adjust the eccentricity of the subset.

3.1.3 Phase Three *(The Second Proposed Method, Hybrid RRT)*

RRTs are either nonoptimal or optimal. Moreover, they are either unidirectional or bidirectional. Each group has its benefits over others. In this phase, a planner has been

proposed that combines these groups to benefit from their advantages.

The proposed method, Hybrid RRT, starts exploring state spaces by a nonoptimal, bidirectional RRT, which helps it find initial solutions faster than not only unidirectional RRT but also optimal RRTs.

Afterward, it combines its two trees into one tree and then shrinks the state space into a subset, which its size is based on the length of the initial solution. It then keeps searching for better solutions. Hybrid RRT idea has several parts that are mentioned below:

1. Nonoptimized RRTs are faster than optimized versions in terms of finding an initial solution. The planner finds initial solutions by implementing a nonoptimized method. Moreover, bidirectional RRTs are faster than unidirectional ones in terms of finding initial solutions.
2. After finding an initial solution, the planner improves the quality of the existing solutions by implementing the optimization process.
3. Single tree optimization is less computationally expensive than two trees, so that the planner integrates its two trees into one so as to optimize only one tree.
4. The planner shrinks state spaces into their subsets so as to return near-optimal solutions faster.

3.1.4 Phase Four *(Evaluation)*

For the evaluation, the proposed methods must be developed in a framework in which other benchmarking sampling-based approaches have already been developed in order to be able to compare them with the proposed methods.

There are several packages for motion planning that can be used for this purpose, such as VIZMO++ (Estrada et al., 2006), Object-Oriented Programming System (OOPSMP) (Plaku et al., 2007), OpenRAVE (Diankov, 2010), and Open Motion Planning Library

(OMPL) (Sucan et al., 2012). VIZMO++ and OOPSMP are no longer maintained so that they are not suitable for recent planners to be developed in them. OpenRAVE is an open-source library, which is designed to be a complete package for robotics. It has geometry representation, collision checking, grasp planning, forward and inverse kinematics for several roots, controllers, motion planning algorithms, simulated sensors, and visualization tools. OMPL, on the other hand, has been designed to focus totally on sampling-based methods. It is a clear mapping between theoretical concepts and abstract classes in the implementation. Therefore, it is convenient to integrate OMPL with a variety of front-ends and other libraries. OMPL is the main motion planning library for the Robot Operating System (ROS) (Quigley et al., 2009) and MoveIt! (Chitta et al., 2012). OMPL has more resources on implementing a much broader variety of sampling-based algorithms than other packages.

OMPL is a C++ library, which is thread-safe. It offers not only Python bindings, but also tools for benchmarking of different sampling-based methods. OMPL represents the search space in a generic way so as to maximize the range of applications for the included planning problems. OMPL state spaces include operations on states such as distance evaluation, test for equality, interpolation, as well as memory management for states ((de)allocation and copying). Moreover, every state spaces have their own format to store states. This format is not exposed outside the implementation of the state space. For operating on the states, the planners rely on the generic functions offered by state spaces. Therefore, this approach makes OMPL be functional to any state space that may be defined, as long as the expected generic functionality is provided.

OMPL has a graphical user interface, which is known as OMPL.app. It is a software that is an excellent introduction to OMPL. Moreover, it is an example of the integration of OMPL with a third-party library for collision checking and loading of 3D meshes, and a

GUI toolkit. Additionally, it allows users to benchmark their new and existing planners on rigid body motion planning problems using a command-line tool. OMPL.app has three different parts:

- A C++ library that contains the binding to third-party libraries.
- A set of command-line demos that highlight significant features of this library.
- A graphical user interface.

The library has the functionality to load meshes in a wide variety of formats using the Open Asset Importer Library (Assimp). Therefore, users can create models of robots and environments in programs like 3DS Max, Blender, SolidWorks, and SketchUp, and then use them in OMPL.app. The collision checking in the OMPL.app is supported by using PQP library (Larsen et al., 2000) and FCL library (Pan et al., 2012). OMPL.app is completely developed in Python.

3.1.4.1 Conceptual Overview of OMPL

This section comes from OMPL paper (Sucan et al., 2012). The intention of OMPL is to be used for various purposes, including industry, research, and education. Therefore, the most important criteria of OMPL were:

a) Clarity of concepts: OMPL has been designed to have a set of components that corresponding to known concepts in sampling-based motion planning, which is indicated in Figure 3.2.
b) Efficiency: OMPL has been developed totally in C++ and is thread-safe.
c) Integration with other robotic software: OMPL offers abstract interfaces, which can be implemented in other software. The dependencies of OMPL are minimal. It only needs Boost C++ libraries. Another feature of OMPL is that it can be integrated with Python modules.

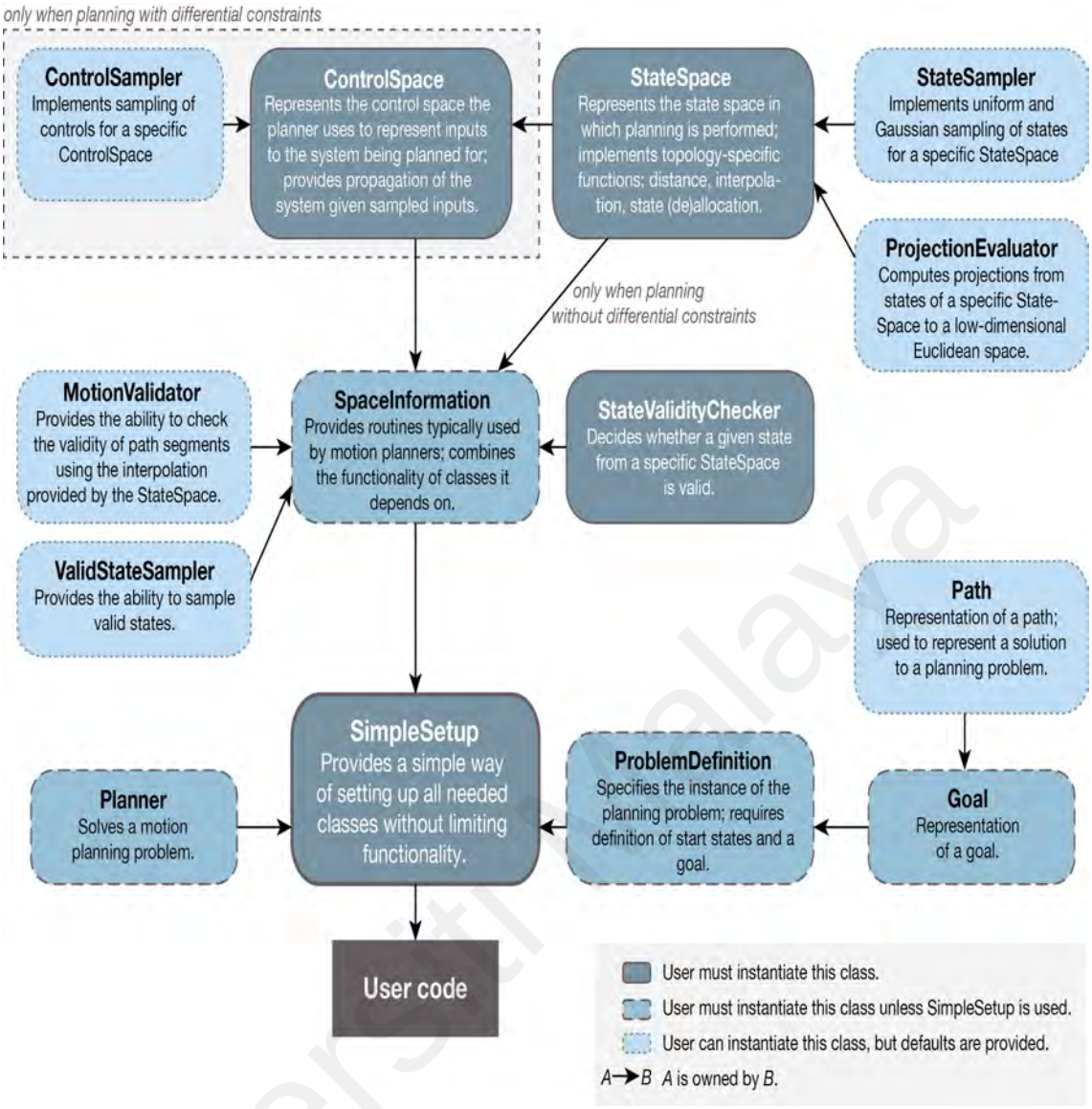


Figure 3.2: Overview of OMPL structure.

d) Straightforward integration of external contributions: OMPL developers tried to make minimalist API constraints for planning algorithms. Therefore, new methods and contributions could be conveniently possible.

OMPL does not have any representation of robots' configuration spaces. Therefore, it does not offer any built-in collision checker or any visualization. OMPL only offers sampling-based algorithms. This approach gives the ability to OMPL to be used for generic searches in high-dimensional continuous state spaces. The collision checking and other geometric representations have been left to the user. As a result, it gives the user freedom to define different types of collision checking so that the sampling-based methods can be used in various scenarios.

3.1.4.2 Implementation of Core Concepts

A comprehensive overview of the core concept of OMPL has been shown in Figure 3.2, which is a high-level overview of OMPL classes and their relationships. This section is quoted from OMPL paper (Sucan et al., 2012).

3.1.4.2.1 *States, Controls, and Spaces*

As it is mentioned earlier, OMPL generically defines state spaces. The state spaces consist of interpolation, memory management, distance evaluation, as well as equality test. Moreover, the configuration spaces have their format to store their states. These formats are not exposed outside the configuration spaces. In order to implement actions on states, the OMPL algorithms use only the generic functionalities offered by configuration spaces. This methodology helps OMPL be useable in any configuration spaces that could provide generic functionality.

Additionally, OMPL is also able to mix configuration spaces. Furthermore, OMPL includes a means of combining state spaces using the class *CompoundStateSpace*. A combined state space implements the functionality of a regular state space on top of the corresponding functionality from the maintained set of state spaces. This allows trivial construction of more complex state spaces from simpler ones. For example *SE3StateSpace*

(the space of rigid body transformations in 3D) is just a combination of *SO3StateSpace* (the space of rotations) and *RealVectorStateSpace* (the space of translations). Instances of *CompoundStateSpace* can be constructed at run time, which is necessary for constructing a state space from an input file specification, as is done, for example, in ROS. For a mobile manipulator one could construct a *CompoundStateSpace* with the two arms and the mobile base as sub-state spaces. An arm typically has a number of rotational joints and can be modeled by either a *RealVectorStateSpace* (if the joints have limits) or a *CompoundStateSpace* with copies of SO(2). The state space for the base can simply be SE(2) (the space rigid body transformations in the plane).

State spaces optionally include specifications of projections to Euclidean spaces (*ProjectionEvaluator*). Several sampling-based planning algorithms use Low-dimensional Euclidean projections to guide their search for a feasible path, as it is much easier to keep track of coverage (i.e., which areas have been sufficiently explored and which areas should be explored further) in such low-dimensional spaces.

In addition to states and state spaces, some algorithms in OMPL require a means to represent controls. Control spaces (*ControlSpace*) mirror the structure of state spaces and provide functionality specific to controls so that planning algorithms can be implemented in a generic way. The only available implementations of control spaces are the Euclidean space and a space for discrete modes because so far, there has not been a need for control spaces with more complex topologies. However, the API allows one to define such control spaces.

3.1.4.2.2 ***State Validation and Propagation***

Whether a state is valid or not depends on the planning context. In many cases state validity simply means that a robot is not in collision with any obstacles, but in general any condition on a state can be used. Based on a given state validity checker, a default

MotionValidator is constructed that checks whether the interpolation between two states at a certain resolution produces states that are all valid. However, it possible to plug in a different *MotionValidator*. For example, one might want to add support for continuous collision checking, which can adaptively check for collisions and provide exact guarantees for state validity.

3.1.4.2.3 ***Samplers***

The fundamental operation that sampling-based planners perform is sampling the space that is explored. Additionally, when considering controls in the planning process, sampling controls may be performed as well.

To support sampling functionality, OMPL includes four types of samplers: state space samplers (*StateSampler*), valid state samplers (*ValidStateSampler*), control samplers (*ControlSampler*), and directed control samplers (*DirectedControlSampler*).

State space samplers are implemented as part of the StateSpace they can sample since they need to be aware of the structure of the states in that space. For instance, uniformly sampling 3D orientations is dependent on their parametrization. Three sampling distributions are implemented by every state space sampler: uniform, Gaussian, and uniform in the vicinity of a specified point. This first sampler is necessary to sample over the entire space, while the latter two are used for sampling states near a previously generated state. This is the most basic level of sampling.

Valid state samplers provide the interface for implementing different sampling strategies. The probability distribution of these samplers depends on the algorithm used and is not imposed as part of the API. The implementation of valid state samplers relies on the existence of a state space sampler and a state validator (*StateValidityChecker*). A common approach to constructing valid state samplers is to repeatedly call a state space sampler until the state validator returns true. Several valid state samplers have been implemented

in OMPL: e.g., a uniform valid state sampler (*UniformValidStateSampler*), two samplers (*GaussianValidStateSampler*, ObstacleBased- *ValidStateSampler*) that generate valid samples near invalid ones (which is often helpful in finding paths through narrow passages).

When considering controls in the planning process, a means to generate controls is also necessary. This functionality is attained using control samplers, which are implemented as part of the control spaces (*ControlSpace*) they represent. Additionally, a notion of direction is also important in some planners: controls that take the system towards a particular state are desired, rather than simply random controls. This functionality is achieved through the use of directed control samplers (derived from the *DirectedControlSampler* class).

3.1.4.2.4 ***Goal Representations***

OMPL uses a hierarchical representation of goals. In the most general case, a Goal can be defined by an *isSatisfied()* function that when given a state, reports whether that state is a goal state or not. While this very general implicit representation is possible, it offers planners no indication of how to reach the goal region. For this reason, *isSatisfied()* optionally reports a heuristic distance to the goal region, which is not required to be a metric.

GoalRegion is a refinement of the general Goal representation, one that explicitly specifies the distance to the goal using a *distanceGoal()* function. The *isSatisfied()* function is then defined to return true when *distanceGoal()* reports distances smaller than a user set threshold. *GoalRegion* is still a very general representation but allows planners to bias their search towards the goal. A refinement of *GoalRegion* is *GoalSampleableRegion*, one which additionally allows drawing samples from the goal region. *GoalState* and *GoalStates* are concrete implementations of *GoalSampleableRegion*.

For practical applications it is often possible to sample the goal region, but the sampling process may be relatively slow (e.g., when using numerical inverse kinematics solvers).

For this reason a refinement of *GoalStates* is defined as well: *GoalLazySamples*. This refinement continuously draws samples in a separate sampling thread, and allows planners to draw samples from the goal region without waiting, after at least one sample has been produced by the sampling thread.

3.1.4.2.5 ***Planning Algorithms***

OMPL includes two types of motion planners: ones that do not consider controls when planning and ones that do. If a planning algorithm can be used to plan both types of motions, with and without controls (e.g., RRT), two separate implementations are provided for that algorithm, one for each type of computed motion. This choice was made for efficiency reasons. With additional levels of abstraction in the implementation it would have been possible to avoid separate implementations. The downside would have been that the implementation of planners would have had to follow a strict structure, which makes the implementation of new algorithms more difficult and possibly less efficient.

For purely geometric planning (i.e., controls are not considered), the solution path is constructed from a finite set of segments, and each segment is computed by interpolation between a pair of sampled states (*PathGeometric*). Several geometric planning algorithms are implemented in OMPL.

When controls are considered, the solution path is constructed from a sequence of controls (*PathControl*). Control-based planners are typically used when motion plans need to respect differential constraints as well. Several algorithms for planning with differential constraints are implemented in OMPL as well, including RRT.

3.1.4.3 benchmarking in OMPL

OMPL offers benchmarking tools to compare the abilities of different motion planning algorithms together. In order to do it, OMPL includes a class, which enables users to solve

a particular motion planning problem repeatedly with different conditions such as various motion planners, different types of state space samplers, and different parameters.

3.1.4.3.1 *Benchmark Configuration File*

In order to start benchmarking, *OMPL.app* has a command-line program called *ompl_benchmark*. This command reads a text-based configuration file using an ini style format with sets of *Key-Value* pairs. This format is the same as the format that *OMPL.app* Graphical User Interface (GUI) uses. Currently, the functionality of the ompl_benchmark class is only for geometric motion planning problems in *SE(2)* and *SE(3)*, as well as kinodynamic planning problems for certain systems.

There are several parameters that need to be defined for benchmark scenarios, which are listed below:

- **name**: This is an identifying name for the motion planning problem to be solved.
- **robot**: Robots in OMPL.app are represented by mesh files, which are the geometry descriptions of robots. This item is the path to the mesh file of the robot.
- **start.[x|y|z|theta], start.axis.[x|y|z]**: These values are describing the start location of the robot in the state space. In two-dimensional state spaces, the orientation is only specified with a **start.theta**, while in three-dimensional state spaces, the axis-angle orientation will be used.
- **goal.[x|y|z|theta], goal.axis.[x|y|z]**: These values are representing the goal location in the state space. They act similarly to the start location in 2D and 3D environments.
- **world**: Like the robot, the state space is represented by a mesh file describing the state space geometry. This parameter consists of the path to the mesh file. If this parameter is not defined, OMPL assumes that the robot will operate in an empty workspace.

- **objective**: Some motion planning algorithms in OMPL have more than one path optimization function. Therefore, it is necessary to define which optimization criterion needs to be selected. This parameter defines the type of optimization, which could have one of these values:
  - *length*
  - *max_min_clearance*
  - *mechanical_work*

  If this parameter is not defined, OMPL considers the length as its optimization parameter for paths.
- **objective.threshold**: This parameter is related to the previous one, *objective*. This is the threshold of the defined optimization parameter. When the planners achieve a path with a better cost than the threshold, it then stops exploring the state space and returns the solution. If this parameter is unspecified, OMPL considers the best possible solution to be found. For example, if the length is the optimization criterion, OMPL sets the threshold on zero, which means that the planner keeps searching the state space until the planning time is finished.
- **control**: OMPL.app has some built-in kinodynamic systems, which are listed below:
  - *kinematic_car*
  - *dynamic_car*
  - *blimp*
  - *quadrotor*

  If the parameter is unspecified, OMPL uses rigid-body planning.
- **sampler**: OMPL offers several sampling methods, which are:
  - *uniform*

  - *gaussian*
  - *obstacle_based*
  - *max_clearance*

  If this parameter is not defined, *uniform* will be selected for sampling state spaces.
- **volume.[min|max].[x|y|z]**: This parameter is used to define the boundary of the state space. OMPL.app assumes a tight bounding box around the workspace, start location, and goal location if this parameter is undefined.
- **time_limit**: This is the amount of time that each motion planner has for planning.
- **mem_limit**: This parameter determines the maximum amount of memory that each planner has during planning. This amount is in MegaByte (MB).
- **run_count**: This is the number of runs each planner in the same scenario. In other words, it is the repeat time of the experiment.
- **output**: This is the path for the benchmark log file. If unspecified, the log file will be saved in the same folder as the configuration file.
- **save_paths**: This is an optional parameter with different values:
  - *none*: It saves no solution at all.
  - *all*: It saves all solutions, including approximate solutions.
  - *shortest*: It only saves the shortest obtained solution.
- **planner**: It includes all motion planning algorithms that are targeted for benchmarking.

The proposed methods have been developed in OMPL. The OMPL benchmarking tools give the ability to compare the proposed methods with existing methods. For benchmarking purposes, the proposed methods must be added to the OMPL framework.

The first proposed method, Informed RRT*-Connect, was compared to RRT*, RRT*-

Connect, and Informed RRT* using the benchmarking tools of OMPL. The second proposed method, Hybrid RRT, was compared to RRT, RRT-Connect, RRT*, Informed RRT*, and BIT*. After comparing each proposed method with the existing methods, the proposed methods were compared together. In each scenario, the planners were run for 100 times, and then their obtained results were compared. The planners were compared in terms of path length as well as planning time.

3.2 Summary

This chapter described the Methodology of this research, which includes the literature review, proposing new methods, and comparing the proposed methods with the existing method. The methodology flowchart is shown in Figure 3.1, which includes four main phases.

Open Motion Planning Library (OMPL) has been implemented as the framework of this research. The proposed methods have been developed in C++, and they are integrated with the OMPL framework. The benchmarking tools of OMPL have been used for comparing the capability of the proposed methods in comparison with the existing methods in terms of path length and planning time.

CHAPTER 4: INFORMED RRT*-CONNECT

This chapter introduces Informed RRT*-Connect, which is a dual-tree, almost-surely asymptotically optimal RRT-based motion planner. It implements two RRT* trees, one rooted in the start location, while another is rooted in the goal location. These two trees try aggressively to establish a connection with each other. Once a link is established, an initial solution is found. The first solution consists of two parts, one part from the start tree, and another part is from the goal tree. Afterward, the planner shrinks the state space to an ellipsoidal subset according to the length of the initial solutions. The subset is an estimation of the states that can potentially improve the solution quality. Searching a subset of the state space instead of the whole state space helps planners find optimal solutions faster. Informed RRT*-Connect keeps searching the subset of the state space in order to improve the quality of the solution. It will stop searching either the planning time is over, or an appropriate solution is found.

Section 4.1 reviews the existing works related to Informed RRT*-Connect, including the standard version of RRT, one of the well-known almost-surely asymptotically optimal version of RRT, RRT*, and Informed RRT*, which is a version of RRT* than uses informed sampling. Section 4.2 presents the necessary background for this chapter, including the problem definition, the informed set of sampling, and the algorithms of related works. Section 4.3 introduces Informed RRT*-Connect approach and the related subroutines of informed sampling. Section 4.4 provides some simulations that compare Informed RRT*-Connect performance against RRT*, RRT*-Connect, and Informed RRT* Section 4.5 evaluates the simulation's results as well as concluding the chapter.

4.1 Introduction

Sampling-based methods such as Probabilistic Roadmap (PRMs) (Kavraki et al., 1996) and Rapidly-exploring Random Trees (RRTs) (LaValle, 1998; LaValle & Kuffner Jr, 2001) use random sampling to avoid constructing a graph of the configuration space. They have shown practically a significant impact upon the high-dimensional state spaces. Most of them are probabilistically complete, which indicates that the planner will return a solution with a sufficient number of iterations if any solution exists. In other words, the probabilistically complete planners keep searching the state spaces, and they do not have any criteria to stop searching. If a given state space offers no solution, the planner cannot detect it, so that it keeps searching the state space so as to find a solution. In order to solve this issue, the probabilistic searches have other limitations such as planning time and/or the number of iterations. Time-limited searches start exploring state spaces, and if they could not find an initial solution within the given time, they stop searching the state spaces and return failure. Iteration-limited searches act similarly to time-limited ones. They keep searching a given state space. They will stop searching if they iterate themselves to the maximum number of iteration. If they could not find any solution before the iterations reach their maximum number, they will return failure. It is because the sampling-based methods have an infinite number of samples to check. Therefore, in practice, it is not possible to check an infinite number of samples to return solutions, so that they should be limited to a limited number of samples.

Among randomized methods, RRTs (LaValle, 1998; LaValle & Kuffner Jr, 2001) have shown a significant performance for single-query planning problems. The standard version of RRT grows as an incremental tree rooted in the start configuration over the collision-free portion of the state space. It stops growing the exploring tree once it spotted one vertex in the goal configuration.

Algorithm 1 presents RRT in which V, E, and G stand for the vertices set, the edges set, and the tree, respectively. Line 1 initializes the vertex set and the edge set. The vertex set gets the start location as its first vertex, and the edge set gets null. It is because there are no edges in the tree yet. Line 2 is where the vertex set and the edge set union is considered as the tree, G. The next line, Line 3, is a loop that can have several stoppage criteria. It can be stopped after a limited number of iterations, such as in this algorithm, it will go through for n iterations. However, there are other stoppage criteria, such as the planning time. For example, the loop could potentially run for infinite iterations, and stop whenever its time is up. Line 4 calls the $Sample$ function. The $Sample$ function returns a random state, and then stores it in x_{rand}. In the next line, Line 5, the returned sample, x_{rand}, and the tree will be passed to the $Extend$ function in order to extend the tree toward the x_{rand}. $Extend$ function gets the tree and x_{rand} as its input arguments. It should first find the nearest vertex to x_{rand} in order to consider it as a potential parent for x_{rand}. If the connection between the potential parent and x_{rand} is collision-free (being collision-free could be different from one planning problem to another), it then adds a new edge to the tree. The new edge connects x_{rand} to its parent. Therefore, the tree gets expanded toward the newly taken sample, x_{rand}. Finally, when the iterations are completed, the obtained tree will be returned (Line 7).

Algorithm 1 RRT Algorithm, $RRT(x_{start})$

1: $V \leftarrow \{x_{start}\}; E \leftarrow \emptyset;$
2: $G \leftarrow (V,\ E);$
3: **for** $i\ =\ 1\ to\ n$ **do**
4: $\quad x_{rand} \leftarrow\ Sample;$
5: $\quad Extend(G = (V, E), x_{rand});$
6: **end for**
7: **return** G

Algorithm 2 outlines the $Extend$ function, which first finds the nearest tree's vertex to x_{rand} (Line 2). $Nearest$ function goes through all the tree's vertices to find the nearest one

to x. Line 3 implements different constraints via the $Steer$ function. In other words, $Steer$ function examines the validity of the given edge regarding externally given constraints by what is usually called a local planner. It then stores the output in x_{new}. For example, if the distance between x and the nearest vertex is more than one $RRTstep$, then the $steer$ returns another sample in the direction of x, which is only one $RRTstep$ away from the nearest vertex. In other words, the function takes one $RRTstep$ toward the new sample and then adds that step to the tree. Line 4 calls the $isCollisionFree$ function in order to check whether the connection between $x_{nearest}$ and x_{new} is collision-free. This function normally checks the validity of the potential edge against all the obstacles of the state space. Therefore, this function has information about all the obstacles in the state space in order to check the validity of the new potential edge. If the link is collision-free, the algorithm continues from Line 5. Otherwise, the algorithm goes to Line 13 to return $Trapped$ so that RRT algorithm, Algorithm 1, ignores the sample, and it needs to have another sample to process. Therefore, it continues with its next iteration to look for another random sample. Line 5 adds x_{new} to the vertex set, and the connection between x_{new} and $x_{nearest}$ will be added to the edge set in Line 6. Line 7 checks whether x_{new} and x are the same. If they are the same, it means that x was not out of the tree reach. Therefore, x has been added to the tree. As a result, the $Extend$ function return $Reached$ as its output (Line 8). In contrast, if x and x_{new} are not the same, it means that x was out of the tree's reach and another vertex in the same direction but nearer to the tree has been added. Therefore, the $Extend$ function return $Advanced$ as its output (Line 10). In other words, this function returns three different outputs due to the status of the new vertex. If x_{rand} is added to the tree, the function output will be $Reached$. If x_{rand} is out of tree's reach and another vertex in the direction of x_{rand} but nearer to the tree is added to the tree, the function's output will be $Advanced$. If, due to the presence of an obstacle, x_{rand} cannot be added to the

tree, the function output will be $Trapped$.

Algorithm 2 Extend Function

```
1:  function Extend(G = (V, E), x)
2:      x_nearest ← Nearest(G, x);
3:      x_new ← Steer(x_nearest, x);
4:      if isCollisionFree(x_nearest, x_new) then
5:          V ← V ⋃ {x_new};
6:          E ← E ⋃ {x_nearest, x_new};
7:          if (x_new = x) then
8:              return Reached;
9:          else
10:             return Advanced;
11:         end if
12:     end if
13:     return Trapped;
14: end function
```

RRT explores the collision-free part of the state space rapidly so that this simple but efficient method is successful for many practical applications. However, RRT may take time to spot a sample in the goal area due to the random sampling process, especially in bug-trap-like problems. Therefore, RRT-Connect (Kuffner & LaValle, 2000) has been proposed to address this problem. It is a bidirectional version of RRT, which grow two trees simultaneously, one from the start location and another one from the goal location.

RRT-Connect is able to find solutions faster than RRT, especially when the goal location is challenging to reach regarding the presence of tight passages that the planner has to pass through them to find solutions. RRT-Connect is incrementally growing two trees simultaneously, one from the start location, while another from the goal configuration. These two trees try aggressively to find a connection. The planner stops exploring the state space when a connection between its two trees established.

Algorithm 3 demonstrates the RRT-Connect approach. Line 1 initializes the first tree of RRT-Connect, which is rooted in the start location. Therefore, its vertex set gets x_{start}

as its first vertex, and the edge set gets null, which means that the start tree has no edge yet. The second tree, which is rooted in the goal area, is initialized in Line 2. Similar to Line 1, the goal tree vertex set get x_{goal} as its first vertex, and the edge set gets null. G_a stands for the first tree, and G_b stands for the second tree (Line 3). Line 4 makes a loop for the iterations of RRT-Connect, which is acting similarly to RRT's. The algorithm gets a random sample, x_{rand}, from the state space in Line 5. After that, RRT-Connect calls the $Extend$ function to add x_{rand} to its first tree, G_a, in Line 6. If the function return $Trapped$, it means that it cannot add x_{rand} to G_a, otherwise it added a new vertex to G_a, which is x_{new}. Once the algorithm comes to Line 7, it means that G_a has a newly added vertex, x_{new}. Therefore, the planner calls the $Connect$ function (Algorithm 4) in order to make a connection between x_{new} and G_b. if the $Connect$ function could add x_{new} to G_b, then both tree has x_{new} as one of their vertices so that they are now connected via x_{new}. As a result, the planner has already found a solution, and it should return its trees (Line 8). But if either the $Connect$ function could not add x_{new} to G_b, or the $Extend$ function could extend G_a toward x_{rand}, the planner needs to go to the next iteration. For the next iteration, the planner swaps its two trees (Line 11). Therefore, in the next iteration, it is the goal tree, which will be extended toward x_{rand}, and the start tree will be trying to be connected to the newly added vertex of the goal tree.

Algorithm 4 outlines the $Connect$ function. It keeps calling $Extend$ function in order to link $Tree_b$ and x_{new} of $Tree_a$. The $Connect$ function will be stopped when the connection is found, or an obstacle blocks the connection. If the $Connect$ function returns $Reached$, it means that the trees are now connected. Therefore, exploring is finished, and the planner will return the trees.

Although RRT and RRT-Connect can work efficiently to find initial solutions, they cannot return optimized solutions due to the lack of optimization process. In order to solve

Algorithm 3 RRT-Connect Algorithm, $RRT - Connect(x_{start},\ x_{goal})$

$V_a \leftarrow \{x_{start}\}; E_a \leftarrow \emptyset;$
$V_b \leftarrow \{x_{goal}\}; E_b \leftarrow \emptyset;$
$G_a \leftarrow (V_a,\ E_a);\ G_b \leftarrow (V_b,\ E_b);$
for $i\ =\ 1\ to\ n$ **do**
    $x_{rand} \leftarrow\ Sample();$
    if $Extend(G_a, x_{rand}) \neq Trapped$ **then**
        if $Connect(G_b, x_{new}) = Reached$ **then**
            return $G_a,\ G_b;$
        end if
    end if
    $Swap(G_a,\ G_b);$
end for
return $G_a,\ G_b;$

Algorithm 4 Connect Function

function $Connect(G,\ x)$
    repeat
        $S \leftarrow Extend(G,\ x);$
    until $S \neq Advanced;$
    return $S;$
end function

this problem, Karaman and Frazzoli introduced RRT* (Karaman & Frazzoli, 2011), which explores the state space similar to RRT, but it optimizes the tree by rewiring its branches to achieve near-optimal solutions. RRT* keeps searching the state space after the first solution has been found to return better paths. RRT* is an almost-surely asymptotically optimal motion planner.

RRT* is incrementally rewiring the tree based on the newly added state to the tree. The newly added states are considered as replacement parents for other existing nearby states in the tree. Algorithm 5 outlines RRT* procedure, which is similar to RRT, but it calls the $Extend^*$ function that is the optimized version of the $Extend$ function.

The $Extend^*$ function includes the rewiring procedure, which tries to optimize the tree vertices near the newly added vertex. Algorithm 6 outlines the $Extend^*$ function. The

Algorithm 5 RRT* Algorithm, $RRT^*(x_{start})$

1: $V \leftarrow \{x_{start}\}; E \leftarrow \emptyset;$
2: $G \leftarrow (V,\ E);$
3: **for** $i\ =\ 1\ to\ n$ **do**
4: $\quad x_{rand} \leftarrow\ Sample();$
5: $\quad Extend^*(G,\ x_{rand});$
6: **end for**
7: **return** G

$Extend^*$ function receives the tree, G, and a random sample from the state space, x_{rand}, which is shown by x in the $Extend^*$ function. It finds the nearest vertex of the tree to x, and stores it in $x_{nearest}$ (Line 2). Line 3 calls the $Steer$ function so as to implement different constraints, and keeps the output in x_{new}. Line 4 checks whether the connection between $x_{nearest}$ and x_{new} is collision-free. If so, then x_{new} can be added to the tree, G (Line 5). In the $Extend$ function, the next step was to add the connection between $x_{nearest}$ and x_{new} as a new edge to the edge set. However, this is different here in the $Extend^*$ function. It saves $x_{nearest}$ in x_{min} (Line 6). Then, $Near$ function returns all near vertices to x_{new}, and they will be stored in X_{near} (Line 7). This function can have two different methodologies, either k-nearest vertices, or vertices within r-disk. The k-nearest is about getting the first k nearest vertices to the random sample, while the r-disk considers the random sample as the center of a ball, which its radius is r, and then returns all vertices within the ball. It depends on the application and the user that which methodology should be selected. Line 8 calculates c_{min}, which is the cost of coming from the root to x_{new} via x_{min}. Afterward, the planner starts to go through all the near vertices, stored in X_{near}, to find a vertex that can offer better cost, and its connection with x_{new} is collision-free. Therefore, the planners makes a loop to go through all collision-free vertices of X_{near} so as to find better parent for x_{new} (Line 9 to Line 14). $Cost(a,\ b)$ function returns the cost of going from vertex a to vertex b. After finding the best possible parent for x_{new}, the planner adds an edge to the

edge set. The new edge connects x_{new} and its parent in the tree (Line 15). In this stage, the tree gets a new vertex so that it is time to rewire the other vertices near the newly added vertex by considering x_{new} as their potential parents. Thus, the planner makes another loop to go through all the near vertices to x_{new} with the aim of comparing their own cost with their cost if they were connected to the tree via x_{new}. Line 17 checks whether the connection between the near vertex, x_{near}, and x_{new} is collision-free, as well as checking whether x_{near} cost is getting better if it is connected to the tree via x_{new} or its previous parent. If x_{new} is a better parent for x_{near} so that the edge of x_{near} and its parent will be removed from the edge set (Line 19), and another edge, in which x_{near} is connected to x_{new} as a child, will be added to the edge set (Line 20).

Finally, the algorithm checks whether it added a new vertex to the tree. If it cannot add any vertex to the tree, it will return $Trapped$, which means that the connection between $x_{nearest}$ and x_{new} is not collision-free (Line 29). If the connection is collision-free, but x_{rand} is out of reach of the tree, and another vertex, which is nearer to the tree and in the direction of x_{rand}, is added to the tree the output will be $Advanced$ (Line 26). Finally, if the x_{rand}, itself is added to the tree so that the algorithm return $Reached$ (Line 24).

RRT* is returning near-optimal solutions. Nonetheless, it still holds the problem of unidirectional searches, which is about finding initial solutions by consuming time in comparison with bidirectional methods.

RRT*-Connect (Klemm et al., 2015) combines RRT-Connect with RRT* to have a bidirectional method which returns near-optimal solutions. Like RRT*, RRT*-Connect keeps searching all over the area in order to return a better solution than the current one. RRT*-Connect is maintaining both trees during their growth like RRT*. It is faster than RRT* in finding solutions. Moreover, it optimizes its solutions while exploring the state space like RRT*.

Algorithm 6 Extend* Function

```
1: function Extend*(G = (V, E), x)
2:     x_nearest ← Nearest(G, x);
3:     x_new ← Steer(x_nearest, x);
4:     if isCollisionFree(x_nearest, x_new) then
5:         V ← V ⋃ {x_new};
6:         x_min ← x_nearest;
7:         X_near ← Near(G, x_new, r_RRT*);
8:         c_min ← Cost(x_nearest, G) + Cost(Line(x_nearest, x_new));
9:         for each x_near ∈ X_near \ x_nearest do
10:            if isCollisionFree(x_near, x_new) & (Cost(x_near, G) + Cost(Line(x_near, x_new)) < c_min) then
11:                x_min ← x_near;
12:                c_min ← Cost(x_near, G) + Cost(Line(x_near, x_new));
13:            end if
14:        end for
15:        E ← E ⋃ {x_min, x_new};
16:        for each x_near ∈ X_near \ x_min do
17:            if isCollisionFree(x_near, x_new) & (Cost(x_new, G) + Cost(Line(x_new, x_near)) <
    Cost(x_near, G)) then
18:                x_parent ← Parent(x_near, G);
19:                E ← E \ {(x_parent, x_near)};
20:                E ← E ⋃ {(x_new, x_near)};
21:            end if
22:        end for
23:        if (x_new = x) then
24:            return Reached;
25:        else
26:            return Advanced;
27:        end if
28:    end if
29:    return Trapped;
30: end function
```

Algorithm 7 presents RRT*-Connect. Line 1, Line 2, and Line 3 initialize the start tree and the goal tree. Each tree has only one vertex, which is its root, and no edges. After initializing the trees, the planner starts to iterate its trees in order to expand them over the collision-free portion of the state space. It does so by creating a loop (Line 4). In each iteration, the planner gets a random sample from the state space and keeps it in x_{rand} (Line 5). It then calls the $Extend^*$ function in order to expand G_a toward x_{rand} (Line 6). If the $Extend^*$ function could add a new vertex to G_a, then it is time for G_b to make a connection between itself and the newly added vertex of G_a. Therefore, Line 7 calls the $Connect^*$ function, which tries to make a connection between the tree and the sample. At the end of each iteration, G_a and G_b will be swapped (Line 9), which means that in the next iteration, another sample, x_{rand}, will be considered as a sample for G_b to be expanded

toward. Then, G_a tries to connect itself to the newly added vertex of G_b. Finally, when the iterations are finished, Line 11 will return the trees.

Algorithm 7 RRT*-Connect Algorithm, $RRT^* - Connect(x_{start},\ x_{goal})$

1: $V_a \leftarrow \{x_{start}\}; E_a \leftarrow \emptyset;$
2: $V_b \leftarrow \{x_{goal}\}; E_b \leftarrow \emptyset;$
3: $G_a \leftarrow (V_a,\ E_a);\ G_b \leftarrow (V_b,\ E_b);$
4: **for** $i\ =\ 1\ to\ n$ **do**
5: $\quad x_{rand} \leftarrow\ Sample();$
6: $\quad$ **if** $Extend^*(G_a, x_{rand}) \neq Trapped$ **then**
7: $\quad\quad Connect^*(G_b, x_{new});$
8: $\quad$ **end if**
9: $\quad Swap(G_a,\ G_b);$
10: **end for**
11: **return** $G_a,\ G_b;$

The $Connect^*$ function, Algorithm 8, is like the $Connect$ function, but the $Connect^*$ function calls the $Extend^*$ function instead of the $Extend$ function. Therefore, it tries to connect the tree to the sample and rewires the tree during the process.

Algorithm 8 Connect* Function

1: **function** $Connect^*(G,\ x)$
2: $\quad$ **repeat**
3: $\quad\quad S \leftarrow Extend^*(G,\ x);$
4: $\quad$ **until** $S \neq Advanced;$
5: $\quad$ **return** $S;$
6: **end function**

Although RRT*-based methods, such as RRT* and RRT*-Connect, obtain near-optimal paths, they are not consistent according to their single-query nature. Moreover, they become expensive in high dimensions (Gammell et al., 2014). In order to minimize the path cost, it is better to look through the states that can achieve the less path cost. However, it is computationally expensive to find the states that can provide better solutions than the first one.

Gammell *et al.* (Gammell et al., 2018, 2014) demonstrated that the effectiveness of the existing approaches diminishes factorially with the dimension of the configuration space. Therefore, they introduced Informed RRT*, which uses informed sampling on RRT* after a first solution is found. Informed sampling goes through an ellipsoidal subset of the configuration space that can provide better solutions. However, Informed RRT* has the problem of other unidirectional tree planners, which is taking time to reach goal configuration, especially when the goal configuration hidden behind narrow passages.

In this chapter, Informed RRT*-Connect is presented, which is a single-query bidirectional planning method for optimal motion planning problems. Informed RRT*-Connect behaves as RRT*-Connect until a first path is found, after which the proposed method only takes samples from the subset of states that may improve the solution. Like other almost-surely asymptotically optimal versions of RRT, Informed RRT*-Connect and RRT*-Connect keep exploring the state space to return near-optimal solutions after the first solution found. However, they are acting differently after a first solution is found. RRT*-Connect look through all over the collision-free part of the state space, while Informed RRT*-Connect search is limited to an ellipsoid subset of the state space which its eccentricity depends on the length of the shortest current solution. Limiting states to a subset gives the ability to the planner to return near-optimal solutions with fewer iterations.

4.2 Background

In this section, the necessary background for the chapter is presented. It first explains the problem definition. Then, the description of the informed set is presented. Afterward, all RRT-based methods that are related to this work are explored.

4.2.1 Problem Definition

This thesis defines the optimal motion planning problem similarly to (Gammell et al., 2014; Karaman & Frazzoli, 2011; Klemm et al., 2015). Let X be the state space, the states that have collisions with obstacles is named $X_{obs} \subset X$. Complement of X_{obs} is $X_{free} = cl\ (\ X_{obs})$, all member states of X_{free} are permissible, where $cl\ (.)$ is a closed set. Let $x_{start} \in X_{free}$ be the start location and $X_{goal} \subset X_{free}$ be the goal configuration. A path defined as a set $\sigma\ [0, 1] \rightarrow X_{free}$ such that $\sigma\ (0) = x_{start}$ and $\sigma\ (1) \in X_{goal}$.

Let $c\ : \Sigma_{X_{free}} \rightarrow \mathbb{R}_{\geq 0}$ be a cost function that assigns a cost value to all collision-free paths. The cost value is the parameter of path optimality. Therefore, the optimal motion planning definition is to search for a path that connects x_{start} to $x_{goal} \in X_{goal}$, while minimizing the cost function. Therefore, the optimized path definition will be defined in Equation 4.1.

$$\sigma^* = \arg_{\sigma \in \Sigma} min\{c\ (\sigma) \mid \sigma\ (0) = x_{start},\ \sigma\ (1) \in X_{goal}, \forall s \in [0, 1], \sigma(s) \in X_{free}\} \tag{4.1}$$

$X_f \subseteq X$ is a subset of the state space which can provide better solution cost than the existing one, c_{best},

$$X_f = \{x \in X \mid f(x) < c_{best}\}. \tag{4.2}$$

Therefore, planners can increase their convergence rate by limiting their search on states that belong to X_f.

However, $f(\cdot)$ in Equation 4.2 is unknown, and it is computationally complicated to be found. Instead, a heuristic function, $\hat{f}(\cdot)$, can be considered as an estimation which

must not overestimate the actual cost of the path (Section 4.2.2). The definition of $\hat{f}(\cdot)$ is similar to Equation 4.2.

4.2.2 Informed Set

The definition of the informed set comes from (Gammell et al., 2014), as a subset of the configuration space that includes states which could improve the optimality of paths. The cost of the path from x_{start} to x_{goal}, $f(x)$, can be divided into two parts. One is the cost of the path from x_{start} to x, $g(x)$, while another is the path cost from x to x_{goal}, $h(x)$. In order to have an estimation of $f(\cdot)$, it is needed to define the estimation of cost-to-come, $\hat{g}(\cdot)$, and the estimation of cost-to-go, $\hat{h}(\cdot)$.

Euclidean distance is the heuristic for problems that are looking for the minimum length of paths. Therefore, the informed subset that can improve the current solution cost, c_{best}, can be defined as

$X_{\hat{f}} = \{x \in X \mid \| x_{start} - x \|_2 + \| x - x_{goal} \|_2 \leq c_{best}\}$, which is the general equation of an n-dimensional prolate hyperspheroid. Figure 4.1 shows an ellipse with x_{start} and x_{goal} as its focal points, the transverse diameter is c_{best}, and the conjugate diameter is $\sqrt{c_{best}^2 - c_{min}^2}$, where c_{min} is the Euclidean distance between x_{start} and x_{goal}.

4.3 Informed RRT*-Connect *(The first proposed method)*

This section presents the proposed method, Informed RRT*-Connect, which is an almost-surely asymptotically optimal dual-tree RRT-based planner. Informed RRT*-Connect acts like RRT*-Connect to explore the configuration space before the first solution is found. Afterward, it limits the search within an ellipsoidal subset to return better solutions than the current one by fewer iterations in comparison to the standard version of RRT*-Connect.

Informed RRT*-Connect first initializes its variables such as its both trees and then starts iterating both of them. It starts exploring the configuration space similarly to

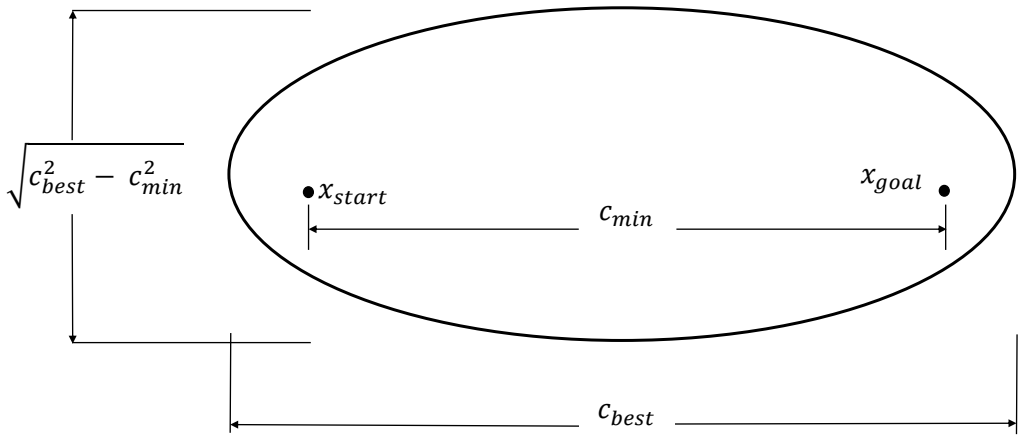


Figure 4.1: The estimated subset, $X_{\hat{f}}$, is an ellipse with x_{start} , and x_{goal}, as its focal points. The ellipse's size is defined based on the minimum cost between x_{start} and x_{goal}, c_{min}, and the cost of the current solution, c_{best}. The ellipse's eccentricity is calculated via c_{min}/c_{best}.

RRT*-Connect until a solution is found. After a solution found, the cost of the shortest path is calculated. If the solution cost is getting smaller, the tree will be pruned based on the best-obtained solution cost. Then, the obtained solution cost will be passed to the sampling function to limit the search within the informed subset. After taking a sample, the planner starts to extend $tree_a$ and try to make a connection between the newly added vertex and $tree_b$. Afterward, the two trees will be swapped for the next iteration. If a new solution is found, it will be added to the solution set.

Algorithm 9 outlines the steps of the Informed RRT*-Connect method. The planner first initializes its two trees by giving the start location as the root to the start tree and the goal location to the goal location as its root (Line 1 ~ Line 3). Line 4 initializes the solution set, which keeps all the connection vertices. Thus, it is initialized by getting null at the beginning, because there is no connection between the two trees yet. The next initialization is about the best-obtained path cost, which should be infinity at the beginning of the planning due to having no solution yet. Therefore, the cost of going from the start location to the goal location is infinity (Line 5). After the initialization is finished, Line 6 starts the iterations by making a loop. In each iteration, the best cost, c_{best}, is stored

Algorithm 9 (Informed RRT*-Connect Algorithm, $InformedRRT^* - Connect(x_{start}, x_{goal})$)

1: $V_a \leftarrow \{x_{start}\}; E_a \leftarrow \emptyset;$
2: $V_b \leftarrow \{x_{goal}\}; E_b \leftarrow \emptyset;$
3: $G_a \leftarrow (V_a, E_a);\ G_b \leftarrow (V_b, E_b);$
4: $X_{soln} \leftarrow \emptyset;$
5: $c_{best} \leftarrow \infty;$
6: **for** $i = 1\ to\ n$ **do**
7: $\quad previous\_c_{best} \leftarrow c_{best};$
8: $\quad c_{best} \leftarrow CalculateShortestPathLengh(X_{soln});$
9: $\quad$ **if** $c_{best} < previous\_c_{best}$ **then**
10: $\quad\quad PruneTree(V, E, c_{best});$
11: $\quad$ **end if**
12: $\quad x_{rand} \leftarrow InformedSample(x_{start}, x_{goal}, c_{best});$
13: $\quad$ **if** $Extend^*(G_a, x_{rand}) \neq Trapped$ **then**
14: $\quad\quad Connect^*(G_b, x_{new});$
15: $\quad$ **end if**
16: $\quad Swap(G_a, G_b);$
17: $\quad$ **if** $isSolutionFound(x_{new})$ **then**
18: $\quad\quad X_{soln} \leftarrow X_{soln} \bigcup \{x_{new}\}$
19: $\quad$ **end if**
20: **end for**
21: **return** $G_a, G_b;$

in another variable called $previous\_c_{best}$ (Line 7). Line 8 recalculates c_{best} to update it. It may get better during the previous iteration. Then, c_{best} and $previous\_c_{best}$ are compared together to find out whether a better solution has been found (Line 9). If so, Line 10 will prune the tree, which means that the unnecessary vertices, which could not potentially improve the solution, will be removed from the tree. Afterward, the planner gets an informed sample (Line 12). The $InformedSample$ function provides a random sample from the state space before an initial solution is found, and then, it returns random samples within a subset of the state space. Line 13 calls the $Extend^*$ function in order to extend G_a toward x_{rand}. If G_a gets extended, the planner calls the $Connect^*$ function so as to try to connect G_b to the newly added vertex of G_a (Line 14). Line 16 swaps the trees for the next iteration. At the end of each iteration, the planner checks whether a new

solution is found (Line 17). If so, the connection point, x_{new} will be added to the solution set (Line 18). Finally, Line 21 return both trees.

4.3.1 Graph Pruning

Graph pruning is a method that removes some vertices from the tree in order to make it smaller so that the planning process will be carried out faster. This is important to keep the tree as small as possible. It is because the planner needs to check all tree's vertices so as to for different purposes such as finding an appropriate parent for the new sample. Karaman *et al.* (Karaman et al., 2011) implemented a graph pruning technique on a version of RRT*, which is able to improve paths. They calculate the estimated cost of each vertex as the sum of cost-to-come and estimated cost-to-go. If the estimated cost is higher than the shortest path's length, then the vertex must be removed from the tree; otherwise, the vertex will remain in the tree. However, the cost-to-come of the vertices may be getting smaller due to the rewiring process. As a result, this method may remove vertices that could potentially provide better solutions.

Gammell *et al.* (Gammell et al., 2018) uses another heuristic for pruning the tree. The cost-to-come of a vertex estimated as the Euclidean distance between x_{start} and the vertex. Similarly, the cost-to-go estimated as the Euclidean distance between the vertex and x_{goal}. If the estimated cost is smaller than the shortest path length, then the vertex will not be removed from the tree. In other words, this method keeps only the vertices that either itself or one of its children located inside the informed set.

The $PruneTree$ function removes the vertices of the tree which are not parent of any other vertices and their estimated cost, $\hat{f}(v) = \hat{g}(v) + \hat{h}(v)$, is greater c_{best}. These vertices are not able to provide better solutions than the existing one. In other words, this function removes the leaves of the tree that have the estimated cost more than c_{best}.

Algorithm 10 presents the $PruneTree$ function. The algorithm makes a loop, which

is continued until there no vertex that can be removed from the tree. It first creates a set, V_{prune}, from all vertices that their estimated cost is higher than the best solution cost, c_{best}, and they are not parents of any vertex (Line 2). In other words, the set is included all vertices that are the leaves of the tree, which are not able to be part of a solution that can offer a shorter path to the current one. Therefore, these vertices should be removed from the tree. Line 3 remove all edges, which are the connections between the vertices in V_{prune} and the tree. After cleaning up the edge set, the function removes V_{prune} from the vertex set.

Algorithm 10 $PruneTree(V \subseteq X,\ E \subseteq V \times V,\ c_{best} \in \mathbb{R}_{\geqslant 0})$

1: **do**
2: $V_{prune} \leftarrow \{v \in V \mid \hat{f}(v) > c_{best},\ and\ \forall w \in V,\ (v, w) \notin E\}$;
3: $E \xleftarrow{-} \{(u, v) \in E \mid v \in V_{prune}\}$;
4: $V \xleftarrow{-} V_{prune}$;
5: **while** $V_{prune} \neq \emptyset$;

4.3.2 Direct Sampling of An Ellipsoidal Subset

The idea of direct sampling of an ellipsoidal subset comes from (Gammell et al., 2014). In order to achieve uniformly distributed sampling inside the ellipsoidal subset $X_{ellipse} \sim \mathcal{U}(X_{ellipse})$, it can been transformed uniformly distributed samples from unit n-ball to ellipsoidal subset. $X_{ball} \sim \mathcal{U}(X_{ball})$,

$x_{ellipse} = Lx_{ball} + x_{center}$,

where $X_{center} = (x_{f1} + x_{f2})/2$ is the center of the hyperellipsoid with its two focal points, x_{f1} and x_{f2}, and

$X_{ball} = \{x \in X \mid \| x \|_2 \leq 1\}$ (H. Sun & Farooq, 2002).

This transformation will be calculated by Cholesky decomposition of the hyperellipsoid matrix, $S \in \mathbb{R}^{n \times n}$,

$LL^T \equiv S$,

$$(x - x_{center})^T S(x - x_{center}) = 1,$$

S having eigenvectors corresponding to the axes of the hyperellipsoid, $\{a_i\}$, and eigenvalues corresponding to the squares of its radii, $\{r_i^2\}$. The transformation, L, maintains the uniform distribution in $X_{ellipse}$ (Gammell & Barfoot, 2014).

Therefore, the transformation of $X_{\hat{f}}$ can be achieved just by transverse the radii and axis. The diagonal matrix of the transverse axis is

$$S = diag\{\frac{c_{best}^2}{4}, \frac{c_{best}^2 - c_{\min}^2}{4}, ..., \frac{c_{best}^2 - c_{\min}^2}{4}\}$$

and decomposition

$$L = diag\{\frac{c_{best}}{2}, \frac{\sqrt{c_{best}^2 - c_{\min}^2}}{2}, ..., \frac{\sqrt{c_{best}^2 - c_{\min}^2}}{2}\}$$

where $diag\{.\}$ stands for a diagonal matrix.

In order to rotate the hyperellipsoid to the world frame, the Wahba problem (Wahba, 1965) can solve it. The rotation matrix is calculated by

$$C = U\ diag\{1, ..., 1, det(U)det(V)\}V_T,$$

where $det(.)$ stands for matrix determinant. $U \in \mathbb{R}^{n \times n}$ and $V \in \mathbb{R}^{n \times n}$ are unitary matrices of $U\Sigma V^T \equiv M$ through singular value decomposition. The matrix M is calculated via the outer product of the first column of the identity matrix, 1_1, and the transverse axis on the world frame, a_1,

$$M = a_1 1_1^T,$$

where

$$a_1 = (x_{goal} - x_{start})/\| x_{goal} - x_{start} \|_2 .$$

Thus, the below formula will calculate the states that belong to the informed subset.

$$x_{\hat{f}} = CLx_{ball} + x_{center},$$

Algorithm 11 presents the informed sampling procedure.

$InformedSample$ function is for sampling the configuration space. If the c_{best} is not infinity, it means that a path between x_{start} and x_{goal} has already been found. Therefore,

$InformedSample$ must return samples within the ellipsoid subset. If no path is found, the function does not limit the configuration space and returns a sample over the configuration space.

Algorithm 11 $InformedSample(x_{start},\ x_{goal},\ c_{max})$

if $c_{max} < \infty$ **then**
    $c_{min} \leftarrow \| x_{goal} - x_{start} \|_2$;
    $x_{center} \leftarrow (x_{start} + x_{goal})/2$;
    $\mathbf{C} \leftarrow RotationToWorldFrame(x_{start},\ x_{goal})$;
    $r_1 \leftarrow c_{max}/2$;
    $\{r_i\}_{i=2,\ldots,n} \leftarrow (\sqrt{c_{max}^2 - c_{min}^2})/2$;
    $\mathbf{L} \leftarrow diag\{r_1,\ r_2,\ \ldots,\ r_n\}$;
    $x_{ball} \leftarrow SampleUnitBall$;
    $x_{rand} \leftarrow (\mathbf{C}\mathbf{L}x_{ball} + x_{center}) \bigcap X$;
else
    $x_{rand} \sim \mathcal{U}(X)$;
end if
return x_{rand};

4.3.3 Rewiring Neighborhood

Optimized versions of RRT use some procedures to rewire the near vertices to the newly added vertex in order to optimize the cost of existing vertices. RRT*-Connect is also rewiring the neighbor vertices of new states so that it almost-surely converges asymptotically to the optimum solution. It is needed to define the near vertices to the newly added vertex. There are two types of definitions for the neighborhood in a tree structure: r-disc variant and k-nearest variant.

4.3.3.1 r-disc variant

In this definition, all vertices which are located within a radius, $r^*_{RRT*-Connect}$ will be considered as the neighbors.

$$r^*_{RRT^*-Connect} := \left(2\left(1+\frac{1}{n}\right)\left(\frac{\lambda(X)}{\zeta_n}\right)\left(\frac{\log(|V|)}{|V|}\right)\right)^{\frac{1}{n}} \tag{4.3}$$

where $|\cdot|$ is the cardinality of a set, the Lebesgue measure of an n-dimensional unit ball and the Lebesgue measure of a set are shown by ζ_n and $\lambda(\cdot)$, respectively.

Informed RRT*-Connect searches the state space to find a solution. Then, the search will be limited to a subset of the configuration space so that the rewiring will be a function of the number of vertices in the informed set, $|V \cap X_{\hat{f}}|$, and its measure,

$$\lambda\left(X_{\hat{f}}\right) \leq \min\{\lambda(X), \lambda(X_{subset})\}.$$

Thus, $r^*_{RRT^*-Connect}$ will be updated as

$$r^*_{RRT^*-Connect} \leq \left(2\left(1+\frac{1}{n}\right)\left(\frac{\min\{\lambda(X), \lambda(X_{subset})\}}{\zeta_n}\right)\left(\frac{\log\left(\left|V \cap X_{\hat{f}}\right|\right)}{\left|V \cap X_{\hat{f}}\right|}\right)\right)^{\frac{1}{n}} \tag{4.4}$$

4.3.3.2 k-nearest variant

In this definition, the near neighbor set includes k closest vertices to the new vertex.

$$k^*_{RRT^*-Connect} \coloneqq e\left(1+\frac{1}{n}\right)\log(|V|). \tag{4.5}$$

Similar to r-disc for Informed RRT*-Connect, the $k^*_{RRT^*-Connect}$ will be updated as

$$k^*_{RRT^*-Connect} \coloneqq e\left(1+\frac{1}{n}\right)\log\left(\left|V \cap X_{\hat{f}}\right|\right). \tag{4.6}$$

If the subset contains less number of vertices in comparison to the whole configuration space, then the rewiring neighborhoods of the Informed RRT*-Connect, Equation 4.4 and Equation 4.6 will be smaller than Equation 4.3 and Equation 4.5. Therefore, the planner requires less computational time for the rewiring process. As a result, its performance

will be improved. However, if the distance between x_{start} and x_{goal} be relatively big, then the informed set is not limiting the rewiring neighborhoods considerably so that Informed RRT*-Connect and RRT*-Connect act similarly.

RRT*-Connect is a probabilistic complete and almost-sure asymptotically optimal planner. Informed RRT*-Connect acts exactly like RRT*-Connect before finding an initial solution so that it is probabilistically complete. Moreover, Informed RRT*-Connect maintains a uniform sample distribution over the ellipsoid subset, in which it implements rewiring that satisfies the bound mentioned in (Karaman & Frazzoli, 2011). Therefore, it is also an almost-surely asymptotically optimal motion planner.

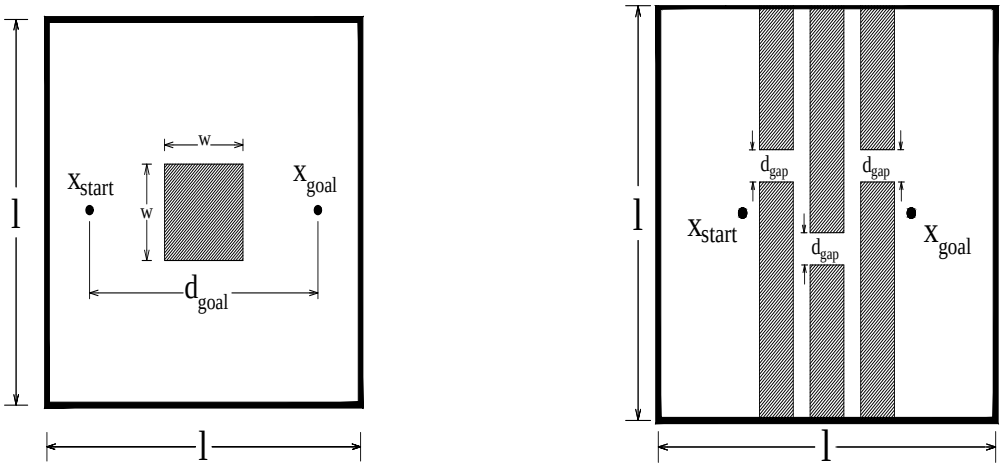
4.4 Simulation

Informed RRT*-Connect was evaluated on simulated problems in $\mathbb{R}^2$, $\mathbb{R}^3$, and $\mathbb{R}^6$ using Open Motion Planning Library (OMPL) (Sucan et al., 2012). OMPL is a motion planning library written in C++, which is integrated with the Robot Operating System (ROS) (Quigley et al., 2009). OMPL App is the front-end for OMPL, which has several rigid bodies and state spaces.

Informed RRT*-Connect has been compared with RRT*-Connect, RRT*, and Informed RRT* on different scenarios in order to demonstrate the significance of the proposed method over the existing planners.

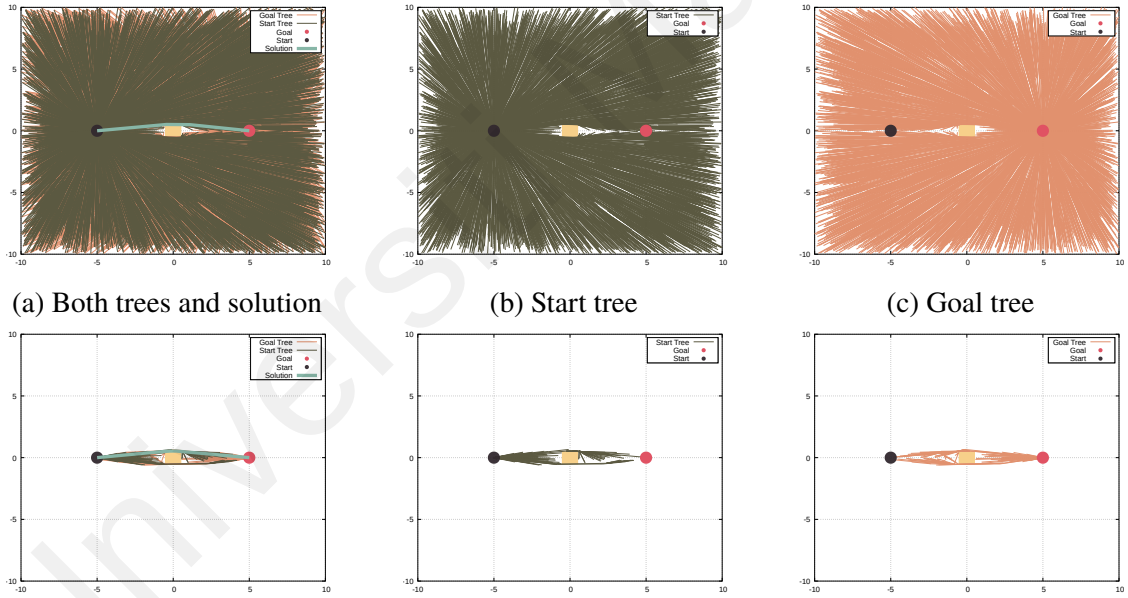
The first two simulations configuration spaces are shown in Figure 4.2. These two simulations are simple which designed for 2 Degree of Freedoms (DoFs) problems. The rest of the simulations are the configuration spaces provided by OMPL App. They are shown in Figure 4.7. *Maze Planar* configuration space, shown in Figure 4.7a, is a 3D problem, while *Home* (Figure 4.7b), and *Apartment Hard* (Figure 4.7c), are 6D problems.

The planners were run 100 times in each scenario in order to get the median of their results.



(a) Single Cube (b) Multiple Narrow Passages

Figure 4.2: Single Cube configuration space, (a), has an obstacle located at the center of the configuration space, and the width of the obstacle is a random variable uniformly distributed over the range [0.25, 0.5]. (b) shows Multiple Narrow Passages configuration space, which only offers planners to pass through three gaps between x_{start} and x_{goal} to produce solutions.



(a) Both trees and solution (b) Start tree (c) Goal tree

(d) Both trees and solution (e) Start tree (f) Goal tree

Figure 4.3: One run of RRT*-Connect and Informed RRT*-Connect in the Single Cube scenario shown when the obstacle width is 0.5, and $1/d_{goal} = 2$. (a), (b), and (c) show RRT*-Connect's trees, while (d), (e), and (f) demonstrate the trees obtained from Informed RRT*-Connect.

4.4.1 Single Cube

The idea of this simulation comes from (Gammell et al., 2014), which has been designed to examine the impact of informed sampling in relation to the size of the map and distance

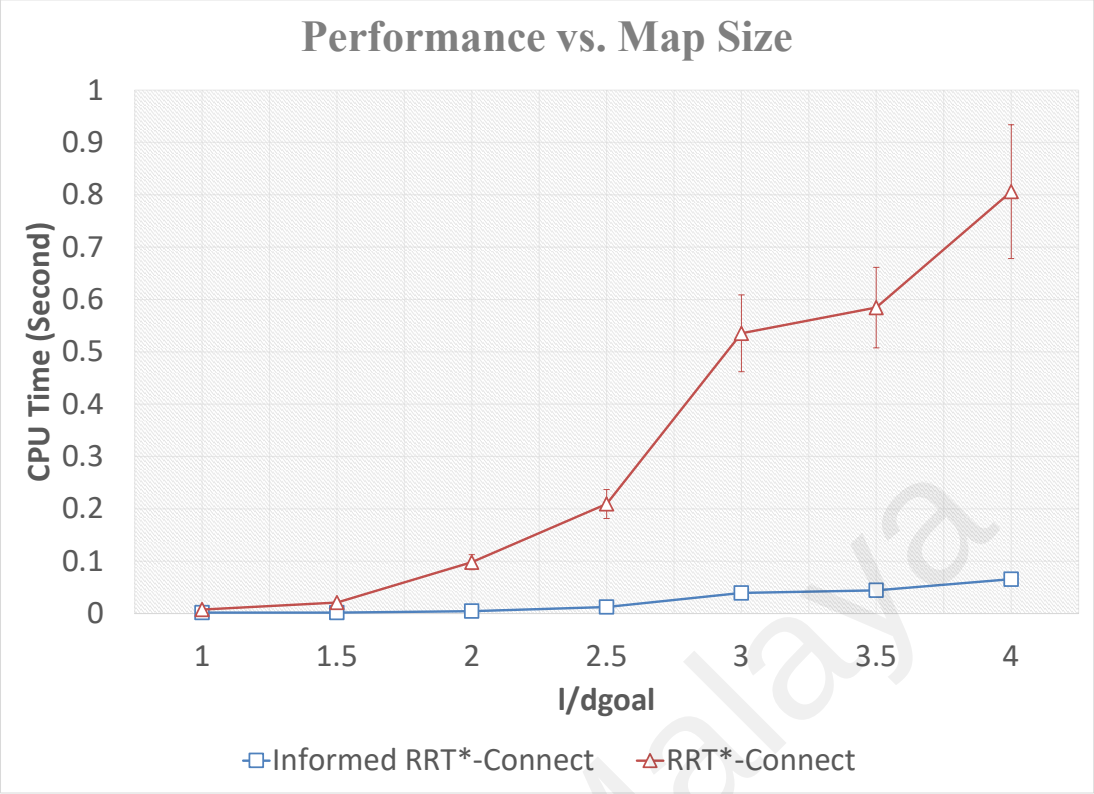


Figure 4.4: The median time required by Informed RRT*-Connect and RRT*-Connect to find solution cost within 2% of the optimal path cost for different amounts of l/d_{goal} in Single Cube scenario obtained from 100 runs. Error bars demonstrate a nonparametric 95% confidence interval for the median time. It can be seen that Informed RRT*-Connect performs best comparatively when the distance between x_{start} and x_{goal}, d_{goal}, is much shorter than the size of the configuration space, l.

between x_{start} and x_{goal}. This simulation includes an obstacle located in the center of the configuration space, which is a square with a randomly selected width between [0.25, 0.5]. The Single Cube map is shown in Figure 4.2a, in which the distance between x_{start} and x_{goal} is shown as d_{goal}, and the length of map is shown as l. The simulation has been carried out for different values of l/d_{goal}. Figure 4.3 demonstrates the trees obtained from RRT*-Connect and Informed RRT*-Connect in this scenario.

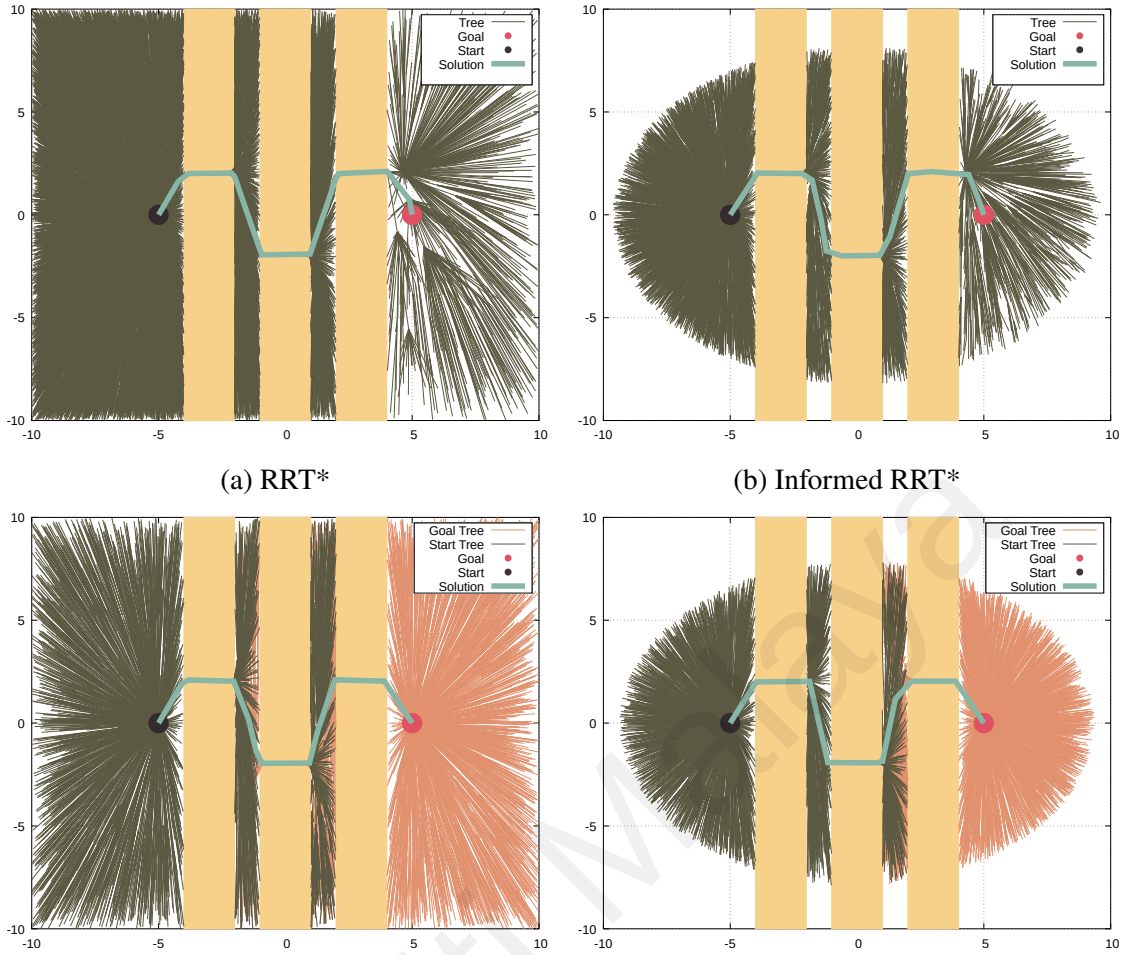
Each planner has solved this scenario for a hundred times, and then the obtained results are compared together. The results obtained from 100 independent runs in Figure 4.4, shows that Informed RRT*-Connect and RRT*-Connect, both found solutions at almost the same time when d_{goal} was equal to l, while at the smallest ratio, 4, Informed RRT*-Connect was about ten times faster than RRT*-Connect. When the subset covers almost all over

the state space, RRT*-Connect and Informed RRT*-Connect have the same amount of space to search. Therefore, they will get an almost similar amount of time to be able to return a near-optimal solution. In contrast, when the subset size is considerably smaller than the state space, Informed RRT*-Connect has a smaller space to search in comparison to RRT*-Connect. As a result, Informed RRT*-Connect is able to return near-optimal solutions faster than RRT*-Connect. The experiment specifications are listed below:

- **Dimension X axis**: (-10, 10)
- **Dimension Y axis**: (-10, 10)
- **start state**: (-2, 0)
- **goal state**: (2, 0)
- **memory_limit**: 100MB
- **runtime_limit**: 20s
- **run_count**: 100

4.4.2 Multiple Narrow Passages

This simulation has been carried out to examine the ability of planners in finding paths in the problems that offer only narrow passages. Figure 4.2b shows the configuration space of Multiple Narrow Passages, in which there are three barriers in the middle of the configuration space. Each barrier has a slight passage with the height that shows as d_{gap}. All these passages have the same height. In order to solve this problem, the planner has to pass through all the three gaps to connect the start location, x_{start}, and the goal location, x_{goal}, together. This simulation has been carried out on the different heights of the passages so as to examine the effect of the passage height on the planners. The gap height in this simulation starts from 2% of the map height, l, and getting smaller until 0.001325% of l. One run of each planner is shown in Figure 4.5.



(a) RRT*

(b) Informed RRT*

(c) RRT*-Connect

(d) Informed RRT*-Connect

Figure 4.5: One example of the trees and solution paths obtained by the planners in Multiple Narrow Passages scenario. (a) and (b) are unidirectional searches. Therefore, their trees density are higher on the left side of the obstacles, while (c) and (d) are bidirectional searches, and their trees' densities are almost the same on both sides of the obstacles.

For each gap height, the planners solved the problem 100 times, and their obtained results are shown in Figure 4.6. It can be seen that the four planners needed approximately the same time to solve the problem when the gap percentage is equal to or bigger than 0.25%. However, they needed a different amount of time to find solutions for smaller gap height. Informed RRT*-Connect took the least time to return solutions among all planners. Informed RRT* was the second-fastest until the gap percentage was bigger than 0.001325% in which RRT*-Connect acted faster than Informed RRT* to return near-optimal solutions. The experiment specifications are listed below:

- **Dimension X axis**: (-10, 10)

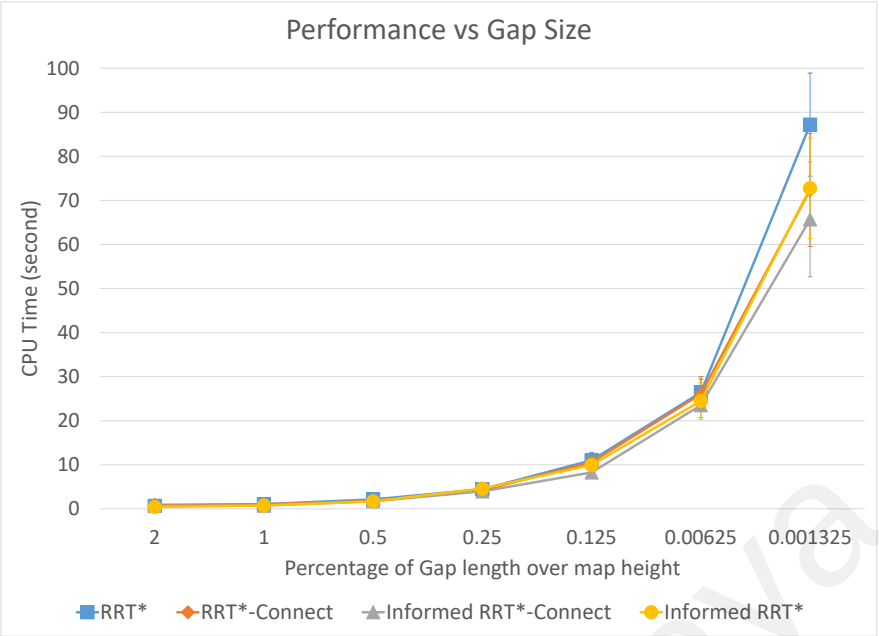
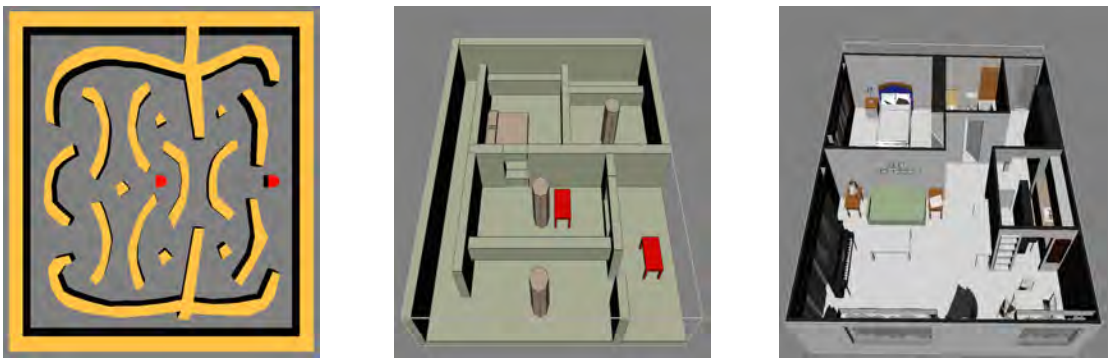


Figure 4.6: The median time required by the planners to find solution cost within 2% of the optimal path cost for different gap sizes in Multiple Narrow Passages scenario obtained from 100 runs. Error bars denote a nonparametric 95% confidence interval for the median time.

- **Dimension Y axis**: (-10, 10)
- **start state**: (-5, 0)
- **goal state**: (5, 0)
- **memory_limit**: 100MB
- **runtime_limit**: 1000s
- **run_count**: 100

4.4.3 OMPL App simulations

OMPL App is a front-end for OMPL, contains several configuration spaces and rigid body robots. They have been designed to use for benchmarking different planners. Some of OMPL's configuration spaces have been used in this simulation, including 3D and 6D problems. The selected configuration spaces that have been tested in this chapter are shown in Figure 4.7.



(a) Maze planar (b) Home (c) Apartment Hard

Figure 4.7: The three OMPL App scenarios which have been tested in the simulation section. (a) is a 3DoFs configuration space, while (b) and (c) are 6DoFs problems. (c) is the most challenging scenario, especially for unidirectional searches. It is because the robot start pose is in the hall, and the goal location is located in the kitchen, which offers narrow doors to pass.

4.4.3.1 *Maze Planar*

Maze Planar, shown in Figure 4.7a, is one of the OMPL App configuration spaces that has been designed for problems with 3DoFs, two real vectors (x-axis and y-axis) and the rotation. The rigid body on the left side is the start pose, the red-colored shape, while the rigid body on the right side is the goal pose. The experiment specifications are listed below:

- **robot**: *car2_planar_robot*
- **environment**: *Maze_planar_env*
- **start state (x, y, yaw)**: (0.01, -0.15, 0.00)
- **goal state (x, y, yaw)**: (41.01, -0.15, 0.00)
- **memory_limit**: 1000MB
- **runtime_limit**: 3s
- **run_count**: 100

4.4.3.2 *Home*

Home is a configuration space that has been designed for problems with 6DoFs (3 coordinate planes (x, y, z) and their rotations (roll, pitch, yaw)). Figure 4.7b shows this

configuration space in which the table on the right side is the start pose, while the table on the left side shows the goal pose. In order to find near-optimal solutions, the planners have to pass through the windows located between the start pose and the goal pose. The experiment specifications are listed below:

- **robot**: *Home_robot*
- **environment**: *Home_env*
- **start state (x, y, z)**: (252.95, -214.95, 46.19)
- **goal state (x, y, z)**: (2.95, -100.00, 46.19)
- **memory_limit**: 1000MB
- **runtime_limit**: 15s
- **run_count**: 100

4.4.3.3 *Apartment Hard*

The last simulation is named "*Apartment Hard*" by OMPL App developers, shown in Figure 4.7c. This scenario has 6 degrees of freedom, including three coordinate planes (x, y, z) and their rotations (roll, pitch, yaw). In this scenario, the start pose is the piano on the left side of the map in the hall, while the goal location is the piano hidden beyond the kitchen walls on the right side of the map. To find near-optimal solutions, the planners need to pass the piano through the kitchen door. The experiment specifications are listed below:

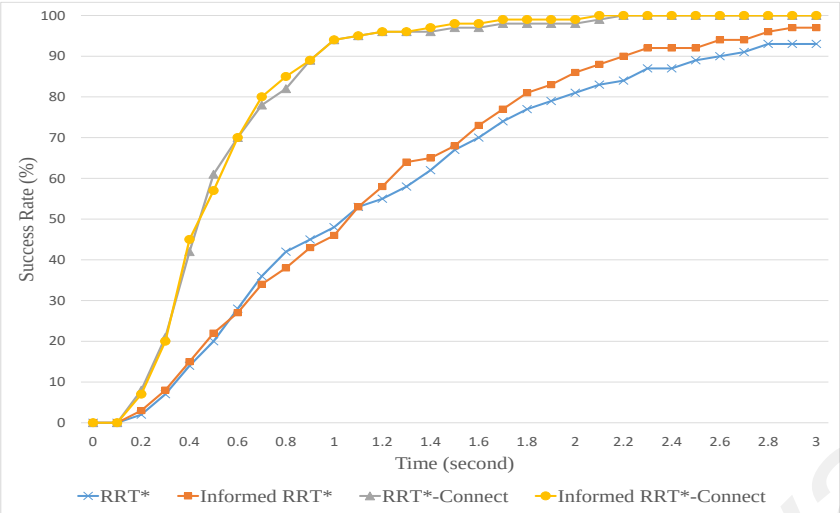
- **robot**: *Apartment_robot*
- **environment**: *Apartment_env*
- **start state (x, y, z)**: (-31.19, -99.85, 36.46)
- **goal state (x, y, z)**: (246.81, -48.85, 36.46)
- **memory_limit**: 1000MB

- **runtime_limit**: 100s
- **run_count**: 100

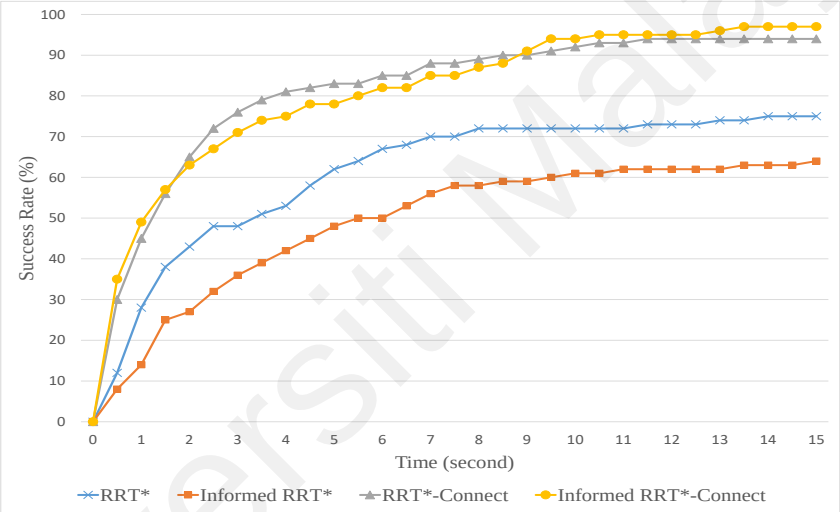
4.5 Discussion

This section presented the evaluation of simulation in this section. First simulation, Single Cube scenario's goal is to show how the informed set acts on different map sizes. Figure 4.4 shows that when the distance between x_{start} and x_{goal} is equal to the configuration space length, l, both planner acts similarly. It is due to the fact that when the distance between x_{start} and x_{goal} is relatively big and the subset almost covers all over the configuration space so that Informed RRT*-Connect search area is not limited to a small portion of the configuration space. Therefore, Informed RRT*-Connect is acting like RRT*-Connect in these kinds of scenarios. In contrast, when the distance of x_{start} and x_{goal} is getting smaller, the time taken for the Informed version is significantly smaller (up to 10 times) than the standard version of RRT*-Connect.

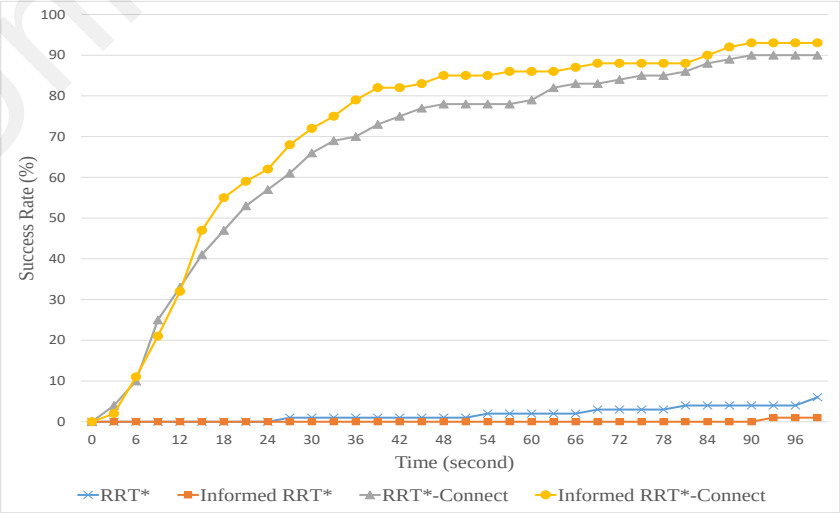
The second simulation, which is about finding paths in a configuration space that only offers narrow passages to connect x_{start} and x_{goal}, shows that dual-tree searches work better than the single-tree methods. It is shown that Informed RRT*-Connect acts better than Informed RRT*. Similarly, RRT*-Connect found paths faster than RRT*. Moreover, when the gap size was only 0.001325% of the map length, the tightest gap, RRT*-Connect worked faster than not only RRT* but also Informed RRT*. It is because RRT*-Connect is a bidirectional RRT-based method, which helps it solve the problems with narrow passages faster than unidirectional RRT-based methods. Informed RRT*-Connect worked as the fastest motion planner in all the gap sizes. It is due to the fact that Informed RRT*-Connect is not only a bidirectional approach but also it uses informed sampling, which let it to have a smaller space to look for near-optimal solutions.



(a) Maze planar

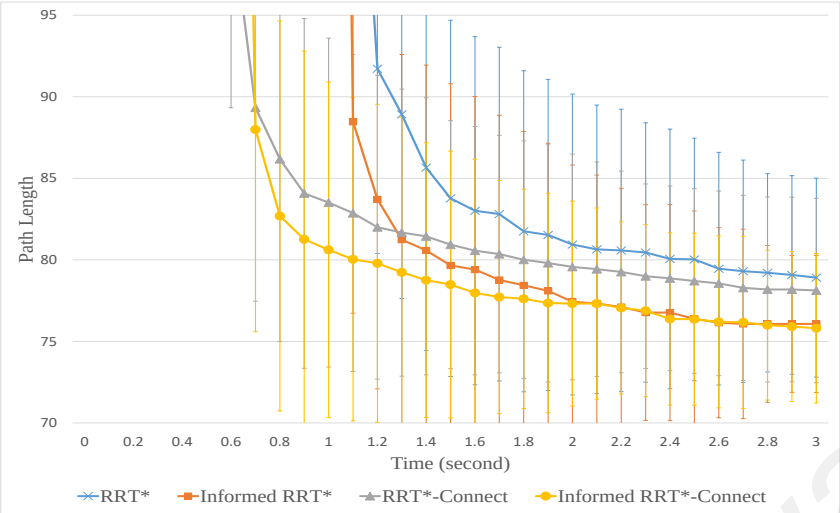


(b) Home

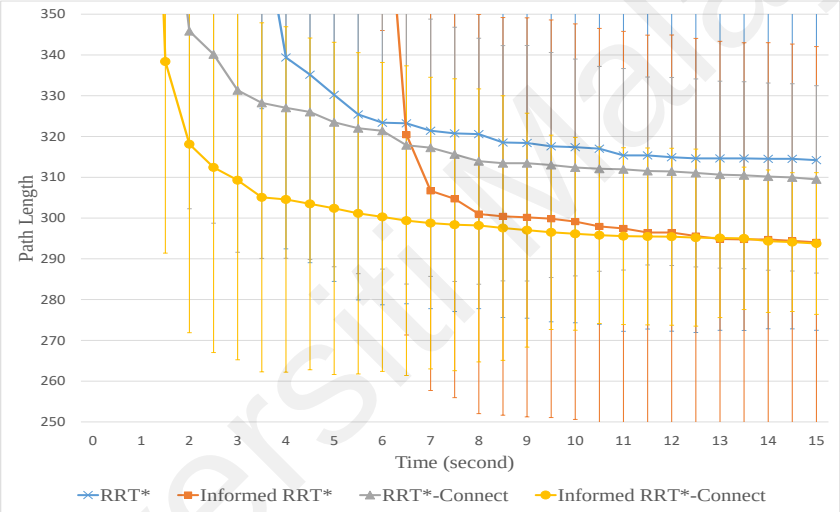


(c) Apartment Hard

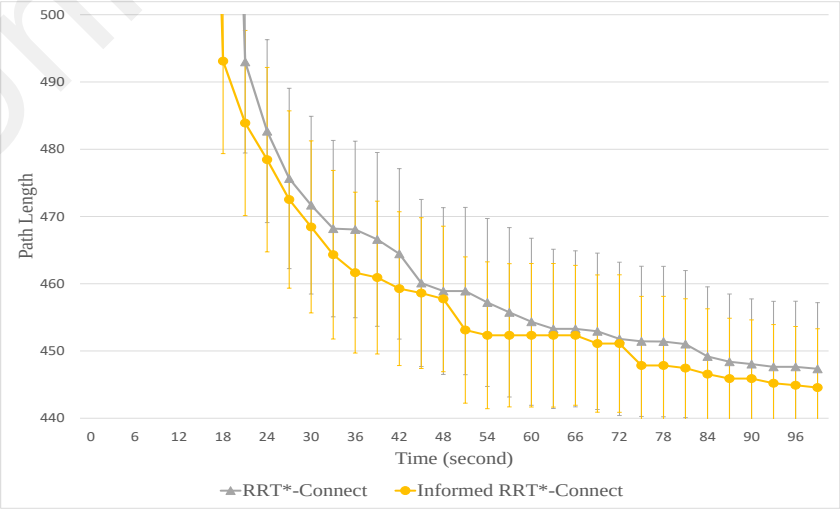
Figure 4.8: The success rate versus time of the four planners on OMPL App configuration spaces.



(a) Maze planar



(b) Home



(c) Apartment Hard

Figure 4.9: The path length versus time of planners for different OMPL App configuration spaces. Error bars denote a nonparametric 95% confidence interval for the median path length.

OMPL Apps configuration spaces' results are shown in Figure 4.8 and Figure 4.9. The obtained results of these simulations are presented in Table 4.1. For the first OMPL App scenario, *Maze Planar*, the planning time was three seconds. The median path cost of Informed RRT*-Connect was 75.8, which is the shortest obtained path among all the planners. The second successful planner in terms of path length is Informed RRT*, which could obtain around 76. RRT* is the least successful planner, which could reach 78.9. In terms of success rate, both bidirectional methods, Informed RRT*-Connect and RRT*-Connect, could achieve total success, 100%, while RRT* and Informed RRT* could achieve 93% and 97%, respectively. In *Home* scenario, the planning time was 15 seconds. Similar to the previous scenario, *Maze Planar*, Informed RRT*-Connect could obtain the best median cost among all the planners, which is approximately 293. The second successful planner was Informed RRT* by offering 294 as its solution cost. RRT* and RRT*-Connect could achieve 314 and 309, respectively. In terms of success rate, the bidirectional methods were acting very better than the unidirectional ones. Informed RRT*-Connect could achieve 97%, while Informed RRT* could only achieve 64%, which makes it the least successful planner in this scenario. For the most challenging scenario, *Apartment Hard*, the time was 100 seconds. The unidirectional RRTs were not successful in these scenarios. It is due to the fact that unidirectional searches cannot find initial solutions as fast as bidirectional searches when the solution is offered via narrow passages. The success rates of RRT* and Informed RRT* were 6% and 1%, respectively. Therefore, the median path length for both of them is infinity. In contrast, Informed RRT*-Connect and RRT*-Connect could achieve 93% and 90%, respectively. In terms of the median of path length, Informed RRT*-Connect could be more successful that RRT*-Connect.

The success rate graphs, as depicted in Figure 4.8, show that the bidirectional searches, RRT*-Connect and Informed RRT*-Connect, are more successful than the unidirectional

Table 4.1: The results of 100 independent runs of each planner in OMPL App scenarios. In *Apartment Hard* scenario, the success rates of RRT* and Informed RRT* are too low so that the *Median Path Length* would be infinity for these two planners. The *Average Path Length* is obtained based on the average path lengths of the runs in which an initial solution is found.

State Space	**Planning Time (s)**	**Planner**	**Median Path Length**	**Average Path Length**	**Success Rate (%)**
Maze Planar	3	RRT*	78.910043	87.47247314	93
		Informed RRT*	76.071604	79.1632481	97
		RRT*-Connect	78.122722	84.72135242	100
		Informed RRT*-Connect	75.807805	80.26087762	100
Home	15	RRT*	314.1974135	433.2467386	75
		Informed RRT*	294.049977	475.1392283	64
		RRT*-Connect	309.479064	339.1374932	94
		Informed RRT*-Connect	293.751639	310.8836259	97
Apartment Hard	100	RRT*	-	589.1352524	6
		Informed RRT*	-	598.4161133	1
		RRT*-Connect	447.3513745	459.9981966	90
		Informed RRT*-Connect	444.5581695	456.1922001	93

searches, RRT* and Informed RRT*. It is the significance of bidirectional planners over the unidirectional planners that are able to find solutions faster than unidirectional planners. It is also noticeable that Informed version of each planner work similar to the standard version in terms of success rate. It is due to the fact that the informed version and the standard version act similarly before the first solution is found.

Informed RRT*-Connect and RRT*-Connect were able to achieve above 90% success in all three simulations, while Informed RRT* and RRT* were only able to achieve approximately 85%, 70%, and 5% in *Maze Planar*, *Home*, and *Apartment Hard* scenarios, respectively. Therefore, bidirectional searches are preferable for motion planning problems in which success in a limited time is essential.

Figure 4.9 demonstrates the path length vs. time in OMPL App configuration spaces achieved by the planners. In Figure 4.9a and Figure 4.9b, it can be seen that Informed versions of the planners produced better results in comparison with the standard versions.

RRT* and Informed RRT* were not able to find solutions in *Apartment Hard* scenario so that for the path length graph (Figure 4.9c), RRT*-Connect, and Informed RRT*-Connect were shown. Informed RRT*-Connect was the most successful planner in terms of success rate and path length.

This chapter presented a new motion planner that combines the ability to quickly finding the first solutions from RRT*-Connect with the capability of quickly returning near-optimal solutions from Informed RRT*. Although RRT*-Connect is a fast path planner in terms of finding solutions, it scans all over the configuration space to return better solutions than the existing one like RRT*, which is not efficient, especially in high-dimensional problems.

The proposed method, Informed RRT*-Connect, not only find its first solutions as fast as RRT*-Connect, which is faster than single-tree based methods but also it returns near-optimal solutions quicker than RRT*-Connect. It is achieved by limiting the configuration space into an ellipsoid subset, which depends on the location of the start configuration, the goal configuration, and the length of the shortest path.

In other words, the proposed method, Informed RRT*-Connect, is a bidirectional asymptotically optimal RRT-based method that uses informed sampling. Informed RRT*-Connect starts exploring state spaces by growing two trees from the start location and the goal location. It expands its two trees toward random samples from all over the state space. After finding a connection between its two trees, it has an initial solution for the given problem. It then limits the state space into one subset of the state space. It gets random samples from the subset instead of all over the state space in order to improve the quality of its solutions faster than methods which do not implement the informed sampling.

It has been successfully demonstrated that the proposed method found first solutions similar to RRT*-Connect in the simulations, and it returns near-optimal solutions faster than the existing planners.

These properties make the proposed method suitable for the motion planning problems in which optimal solutions must be obtained with a limited number of iterations and/or in a limited time slot.

CHAPTER 5: HYBRID RRT

RRT-based methods can be divided into two categories, nonoptimized and optimized. Nonoptimized RRT-based motion planners such as RRT and RRT-Connect have been designed to find initial solutions. They will be stopped soon after finding their initial solutions. On the other hand, optimized RRT-based motion planners like RRT*, Informed RRT*, and Informed RRT*-Connect have been designed to return near-optimal solutions so that they keep working after an initial solution is found in order to optimize their trees.

The optimization process includes rewiring trees to minizine the solution cost. The optimization process will be run in each iteration so that it is a time-consuming process. As a result, the optimization processes make the optimized versions of RRT slower than nonoptimized versions. Consequently, nonoptimized versions of RRT can expand their trees in the state space faster than optimized versions, which helps them find initial solutions faster than optimized versions.

In many scenarios, there is a limited amount of time for motion planning. Therefore, designers mostly sacrifice the path quality and select nonoptimized motion planners to be able to increase the change of having at least one feasible solution at the end of planning time. However, if the solution is found before planning time is over, then the planner cannot benefit from the remaining time to optimize the solution due to the lack of optimization process. This problem limits using optimized versions of RRT to the problem specifics.

In Addition to these two categories, most RRT-based methods are also categorized into two groups, unidirectional and bidirectional methods. Unidirectional methods explore the state space by growing a tree rooted in the start location and expand the tree toward random samples. Bidirectional approaches, on the other hand, grow two trees simultaneously, one is originated from the start location, and another one is originated from the goal

location. These two trees try aggressively to find a link between themselves, which would be a solution to the problem. Having two trees makes bidirectional methods faster than unidirectional searches for finding initial solutions. However, maintaining two trees would be more computationally expensive than single-tree maintenance.

This chapter introduces a single-query semi-bidirectional planning method for optimal motion planning problems called Hybrid RRT, which is not only a combination of unidirectional and bidirectional methods but also is a combination of nonoptimized and optimized versions of RRT. The goal of these combinations is to benefit from the advantages of these types of planners in a single planner.

Hybrid RRT starts exploring state spaces with a nonoptimized dual-tree RRT-based motion planner to be able to find an initial solution as fast as possible. Afterward, it wants to optimize the solution so that it implements an optimized single-tree RRT-based motion planner, which helps it optimize the solution in a short period of time.

Hybrid RRT divides the planning time into three phases. Phase one is to find an initial solution, the second phase is to combine two trees of phase one into one tree, and phase three is to optimize the solution. Hybrid RRT implements a dual-tree search to achieve the first solutions faster than unidirectional methods. After finding the first solution, Hybrid RRT needs to merge its two trees into one. Then, it optimizes the tree to find near-optimal solutions.

In order to get near-optimal solutions fast from the optimization process, Hybrid RRT limits the state space into a subset of the state space like Informed RRT*. Therefore, it needs to combine two trees of phase one into one tree to be able to implement Informed sampling on a single tree.

Hybrid RRT is neither an entirely unidirectional method nor a fully bidirectional one. It is a combination of both groups. Moreover, it is neither a nonoptimized version nor an

optimized version. It uses a nonoptimized search for finding initial solutions, which makes it faster than optimized versions of RRT. Moreover, it implements an optimization process to be able to return near-optimal solutions. Hybrid RRT can find first solutions as fast as RRT-Connect and return the near-optimal solutions as quickly as Informed RRT*.

Section 5.1 reviews the existing works related to Hybrid RRT. Section 5.2 introduces Hybrid RRT method. Section 5.3 provides some simulations that compare Hybrid RRT performance against some of the state-of-the-art motion planners. Section 5.4 evaluates the simulation's results.

5.1 Introduction

There are two types of sampling-based methods, multi-query, and single-query methods. Multi-query planners such as Probabilistic Road Map (PRM) (Kavraki et al., 1996) can solve several problems with different start locations and goal locations in the state space. They first create a roadmap by taking random samples, then connect different locations of the map through the created roadmap.

Single-query planners such as Rapidly-exploring Random Tree (RRT) (LaValle, 1998) do not make a roadmap like multi-query planners. Instead, they construct a tree rooted in the start location and explore the state space by growing the tree toward random samples. Once the goal location is sampled, the exploring process will be stopped.

Unidirectional sampling-based methods, such as RRT, could have a problem to sample the goal area due to their random sampling scheme. RRT-Connect (Kuffner & LaValle, 2000) has been proposed to minimize the time to find solutions. RRT-Connect has two RRT trees, one from the start location and another one from the goal location. Exploring the state space with two trees makes RRT-Connect a faster planner in comparison to RRT, especially when the goal location is challenging to be sampled by using unidirectional searches.

Although RRT-based methods can solve the motion planning problems efficiently, they provide non-optimal solutions (Karaman & Frazzoli, 2011). It is due to the fact that they explore the state space with the random walk so that their outputs would be a sequence of random samples, and they do not have any procedure for optimizing their trees.

RRT* (Karaman & Frazzoli, 2011) implements a rewiring operation, which leads to near-optimal solutions. These types of planners called asymptotically optimal, which means that they will return near-optimal solutions by increasing the number of samples. RRT* does not stop exploring state spaces after a solution is found. It continues exploring the state space with the aim of returning better solutions than the current one. RRT* keeps sampling all over the state space to optimize the current solution so that it is an inefficient way due to its single-query nature (Gammell et al., 2014).

Informed RRT* (Gammell et al., 2018, 2014) solves this problem of RRT* by limiting the search area to a subset of the state space so as to return near-optimal solutions faster than the standard version of RRT*. The subset is a function of the current solution, which means that Informed RRT* cannot limit the state space to a subset before a solution is found. In other words, Informed RRT* acts similarly to RRT* before a solution is found. Therefore, Informed RRT* only expedites the optimization process. It still has the problems of other unidirectional methods, which is spotting a sample in the goal area, especially when the goal area is hidden beyond the narrow passages.

There are some other RRT-based methods, which try to find initial solutions faster than the standard version of RRT* (Gammell et al., 2015; Wang et al., 2019, 2018). *Wang et al.* (Wang et al., 2019, 2018) modified the sampling process to find solutions faster than standard RRT*. However, these methods resulted in nonuniform sample distributions. Batch Informed Trees (BIT*) (Gammell et al., 2015) limits the state space to a subset of it, which including the start location and the goal location in order to return initial solutions

faster. Although BIT* could return first solutions faster than RRT* in many scenarios, it requires more time to find the first solutions in Bug-Trap-like scenarios in comparison to methods that do not limit the exploring area.

5.2 Hybrid RRT *(The second proposed method)*

This section presents the proposed method, Hybrid RRT, which is a semi-dual-tree RRT-based method. Hybrid RRT divides the planning process into three sub-processes: finding-an-initial-solution, combining-two-trees, and optimizing-the-current-solution.

For finding an initial solution, Hybrid RRT implements a dual-tree search to be able to find the first solutions faster than the unidirectional searches. For the optimization process, it applies informed sampling on a single-tree, which helps it return near-optimal solutions more quickly than other methods that do not limit their search area for the optimization process. In other words, it uses a bidirectional search in finding-an-initial-solution sub-process and uses a unidirectional search for optimizing-the-current-solution sub-process. Therefore, it needs to convert the two trees of the first sub-process into one to be able to pass it the third sub-process. Thus, the second sub-process duty is to transform the bidirectional trees into a unidirectional tree.

5.2.1 Hybrid RRT Algorithm

Algorithm 12 outlines the steps of the Hybrid RRT method. The planner first initializes its parameters. Line 1 initializes the start tree so that it takes the start location as its root and null for its edge set. Similarly, Line 2 gives the goal location as the first vertex to the goal tree and an empty set as the edge set. The solution set is set to be null at the beginning of the planning, because there is no solution yet (Line 4). X_{soln} keeps all the vertices located within the goal region. Moreover, the best cost of the obtained path, c_{best}, should be set to infinity before iterations are started (Line 5). In other words, c_{best} keeps

the cost of the shortest path. After initializing the parameter, the planner starts to find an initial solution with a bidirectional nonoptimized RRT. It calls the $FindFirstSolution$ function, Algorithm 13, in order to find its initial solution (Line 6). This function returns two arguments, one is its status, and another one is the connections point. If the function was successful in finding an initial solution, it then returns $Success$ as its status, and x_{cxn} as the connection point of the two trees. In contrast, if it could not find any solutions for the given problem, it returns $Failure$ as its status and $null$ as its connection point. Line 7 checks whether the $FindFirstSolution$ function was successful. If so, the planner starts combining its two trees into one. Line 8 calls the $CombineTwoTrees$ function so as to merge the two trees. After combining two trees, the planner optimizes its solution by a unidirectional asymptotically optimal RRT method (Line 9). Finally, it returns the obtained tree in Line 11.

Algorithm 12 Hybrid RRT Algorithm, $HybridRRT(x_{start},\ x_{goal})$

1: $V_a \leftarrow \{x_{start}\}; E_a \leftarrow \emptyset;$
2: $V_b \leftarrow \{x_{goal}\}; E_b \leftarrow \emptyset;$
3: $G_a \leftarrow (V_a,\ E_a);\ G_b \leftarrow (V_b,\ E_b);$
4: $X_{soln} \leftarrow \emptyset;$
5: $c_{best} \leftarrow \infty;$
6: $[status, x_{cxn}] \leftarrow FindFirstSolution(G_a, G_b);$
7: **if** $status \neq Failure$ **then**
8: $\quad [G_a, c_{best}, X_{soln}] \leftarrow CombineTwoTrees(G_a, G_b, c_{best}, X_{soln}, x_{cxn});$
9: $\quad G_a \leftarrow OptimizeTree(G_a, x_{start}, x_{goal}, c_{best}, X_{soln});$
10: **end if**
11: **return** $G_a;$

5.2.1.1 *FindFirstSolution* Function

The $FindFirstSolution$ function, Algorithm 13, gets two trees (G_a and G_b) as its input arguments (Line 1). Then, the function creates loop to start exploring the state space (Line 2). It gets a random sample, x_{rand}, (Line 3), then it expands G_a toward x_{rand} by

calling the *Extend* function (Line 4). If the *Extend* function could all a new vertex, x_{new}, to G_a, then G_b must try to make a connection to the newly added vertex of G_a. Line 5 calls the *Connect* function so as to make a connection between G_b and x_{new}. If the *Connect* could add x_{new} as a vertex to G_b, it means a connection between the two trees has already been found. Therefore, it is time to return the solution. Thus, the function returns *Success* as its status (Line 7) as well as x_{new} as the connection point (Line 6), x_{cxn}. If neither the *Extend* function nor the *Connect* function is successful, then another iteration must be started. For the next iteration, the trees should be swapped (Line 11). If the number of iterations reaches its maximum and no link between trees is found, the function needs to return *null* as the connection point (Line 13), and *Failure* as its status (Line 14).

Algorithm 13 *FindFirstSolution* Function

```
1:  function FindFirstSolution(G_a, G_b)
2:      for i = 1 to n do
3:          x_rand ← Sample();
4:          if Extend(G_a, x_rand) ≠ Trapped then
5:              if Connect(G_b, x_new) = Reached then
6:                  x_cxn ← x_new;
7:                  status ← Success;
8:                  return [status, x_cxn];
9:              end if
10:         end if
11:         Swap(G_a, G_b);
12:     end for
13:     x_cxn ← null;
14:     status ← Failure;
15:     return [status, x_cxn];
16: end function
```

The *Extend* function gets the tree and the random sample, it then finds the nearest vertex of the tree to the sample. Afterward, it implements the required constraints via the *Steer* function. Finally, it adds the sampled point to the tree if this connection is collision-free.

The $Extend$ function provides three different status, $Reached$, $Trapped$, and $Advanced$. $Reached$ is when the sample is added to the tree, and $Trapped$ is when the sample cannot be added to the tree due to the presence of an obstacle. Finally, if the sample is far from the tree reach and then another vertex in the direction of the sample but nearer to the tree is added to the tree, the function will return $Advanced$ Algorithm 14 outlines the $Extend$ function.

Algorithm 14 $Extend$ Function

```
1:  function Extend(G = (V, E), x)
2:      x_nearest ← Nearest(G, x);
3:      x_new ← Steer(x_nearest, x);
4:      if isCollisionFree(x_nearest, x_new) then
5:          V ← V ⋃ {x_new};
6:          E ← E ⋃ {x_nearest, x_new};
7:          if (x_new = x) then
8:              return Reached;
9:          else
10:             return Advanced;
11:         end if
12:     end if
13:     return Trapped;
14: end function
```

$Connect$ function duty is to connect two trees. It gets the newly added vertex of G_a, x_{new}, and G_b, and then try to add x_{new} to G_b by calling $Extend$ function repeatedly. It stops calling $Extend$ function when $Extend$ function returns either $Reached$ or $Trapped$. When $Connect$ function receives $Reached$ from $Extend$ function, it means that the connection between the two trees is found. Algorithm 15 presents this procedure.

Algorithm 15 *Connect* Function

```
1: function Connect(G, x)
2:     repeat
3:         S ← Extend(G, x);
4:     until S ≠ Advanced;
5:     return S;
6: end function
```

5.2.1.2 *CombineTwoTrees* Function

After *FindFirstSolution* function finds a path, it is time to merge the two trees into one. Therefore, all vertices and edges of the goal tree must be added to the start tree.

$$E_{startTree} = E_{startTree} \cup E_{goalTree}$$

$$V_{startTree} = V_{startTree} \cup V_{goalTree}$$

where $E_{startTree}$ and $E_{goalTree}$ stand for edges of the start tree and the goal tree, respectively. Similarly, $V_{startTree}$ and $V_{goalTree}$ are the vertices of the start tree and the goal tree, respectively. For simplicity $E_{startTree}$ and $V_{startTree}$ will be shown by E and V.

At this stage, all vertices and edges of the goal tree are added to the start tree. However, some modifications are needed to connect these two trees correctly. The two trees work similarly to a single tree when all of their vertices have a path to the start tree root. The start tree vertices have already had their paths to the root. So, it is only the goal tree vertices, which need to find their paths to the start tree root. In other words, all vertices of the goal tree must be directed to the x_{start} instead of x_{goal}. Therefore, the root of all the goal tree vertices must be changed from x_{goal} to x_{start}. The start tree and the goal tree are connected together via the obtained path. In other words, the path is the only connection between these two trees so that the path is the starting stage of merging these two trees.

In problem definition section of Chapter 4, Section 4.2.1, path defined as $\sigma[0,\ 1] \rightarrow$

X_{free} such that $\sigma(0) = x_{start}$ and $\sigma(1) \in X_{goal}$. The $FindFirstSolution$ function implements a bidirectional search to find initial solutions. Therefore, the path has two different parts; one part is from the start tree, while another is from the goal tree. Let x_{cxn} be the connection vertex that is in both trees. Therefore, the path defined as $\sigma = \sigma_{start} \cup \sigma_{goal}$ such that

$\sigma_{start}[0,\ 1] \rightarrow X_{free} \mid \sigma_{start}(0) = x_{start},\ \sigma_{start}(1) = x_{cxn},$

and

$\sigma_{goal}[0,\ 1] \rightarrow X_{free} \mid \sigma_{goal}(0) = x_{goal},\ \sigma_{goal}(1) = x_{cxn}.$

Let $n + 1$ be the number of vertices in σ_{start}, and $m + 1$ be the number of vertices in σ_{goal}. So, σ_{start} can be defined such

$$\sigma_{start} = \sigma_{start}(0) \cup \sigma_{start}(\frac{1}{n}) \cup \sigma_{start}(\frac{2}{n}) \cup \ldots \cup \sigma_{start}(\frac{n-1}{n})$$

$$\cup\sigma_{start}(1) = \bigcup_{i=0}^{n} \sigma_{start}(\frac{i}{n})$$

Similarly, σ_{goal} can be defined such

$$\sigma_{goal} = \sigma_{goal}(0) \cup \sigma_{goal}(\frac{1}{m}) \cup \sigma_{goal}(\frac{2}{m}) \cup \ldots \cup \sigma_{goal}(\frac{m-1}{m})$$

$$\cup\sigma_{goal}(1) = \bigcup_{i=0}^{m} \sigma_{goal}(\frac{i}{m})$$

σ_{start} is already part of the start tree, while σ_{goal} needs to be modified to be part of the start tree. The first step is to remove the edge of x_{cxn} and its parent in the goal tree and add it to the start tree. In other words, the rule of child and parent must be exchanged, x_{cxn} must be parent of its parent in the goal tree, $\sigma_{goal}(\frac{m-1}{m})$.

Therefore, the edge in which $\sigma_{goal}(\frac{m-1}{m})$ was the parent and x_{cxn} was the child must be

removed from E. Let show an edge with its two vertices as $(x_{parent},\ x_{child})$. So,

$$E \leftarrow\ E \backslash (\sigma_{goal}(\frac{m-1}{m}),\ x_{cxn})$$

Then, the new edge in which x_{cxn} is the parent of $\sigma_{goal}(\frac{m-1}{m})$ must be added to E.

$$E \leftarrow\ E \cup (x_{cxn},\ \sigma_{goal}(\frac{m-1}{m}))$$

As a result, the two trees connection is no longer x_{cxn}. They are now connected via the previous parent of x_{cxn}, $\sigma_{goal}(\frac{m-1}{m})$. In other words, x_{cxn} gets one edge closer to x_{goal}. This process must be continued until x_{goal} added to the start tree. It means that all the edges of σ_{goal} must be removed from E.

$$E \leftarrow E \backslash (\bigcup_{i=m}^{0} (\sigma_{goal}(\frac{i-1}{m}),\ \sigma_{goal}(\frac{i}{m})))$$

Instead of the removed edges, the reversed version of them must be added to E.

$$E \leftarrow E \bigcup_{i=m}^{0} (\sigma_{goal}(\frac{i}{m}),\ \sigma_{goal}(\frac{i-1}{m}))$$

After changing the directions of all edges of σ_{goal}, the vertices of the path have their way back to x_{start}, which means that they are connected to the start tree correctly.

All vertices of the goal tree can be categorized into three groups from the viewpoint of σ_{goal}, the first group is the vertices located on the path, the second group is the vertices that pass from at least one of the path vertices to reach their root, x_{goal}, and finally, the third group is the vertices that in their way to x_{goal}, they do not pass from any of the path vertices. The third group is connected to x_{goal} via other branches of the goal tree than the

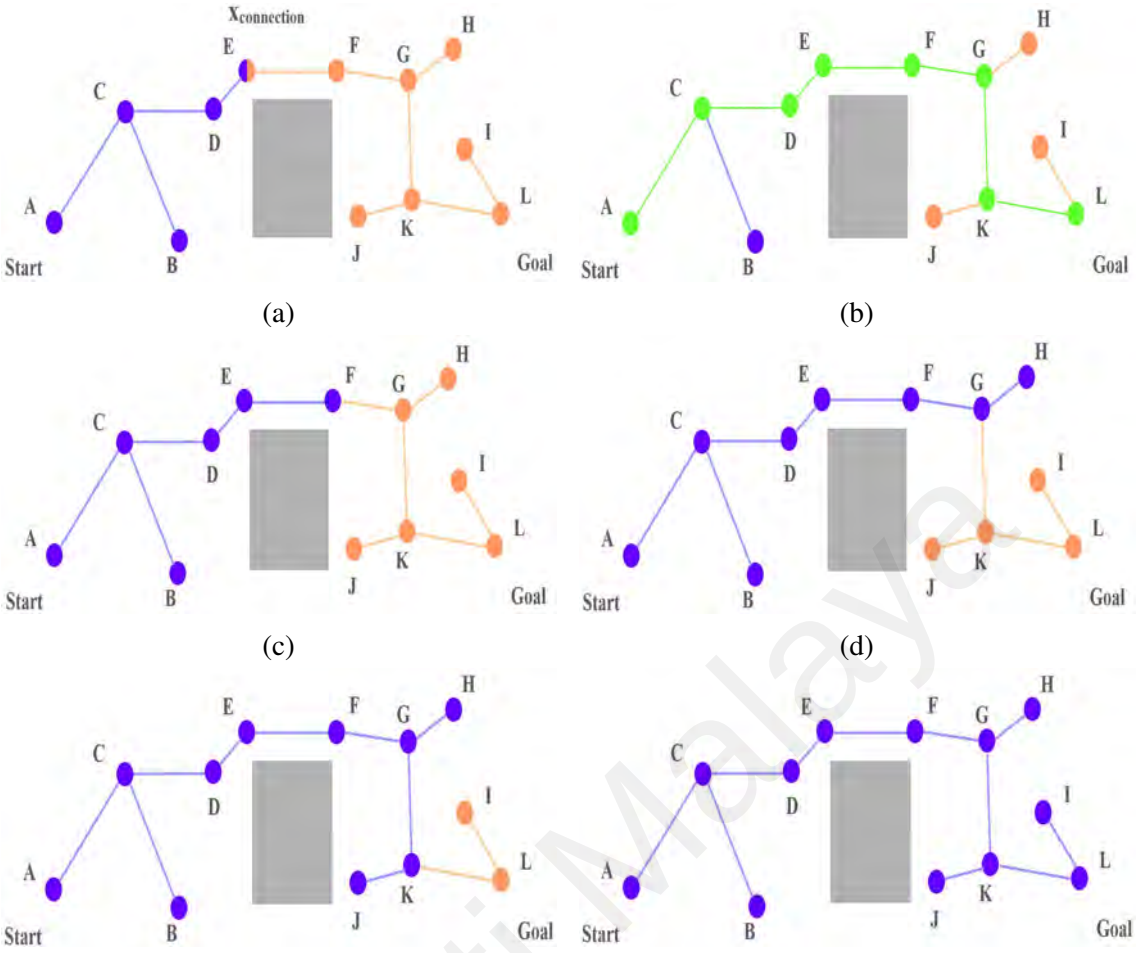


Figure 5.1: An example of combining two trees together.

path.

By reversing the path edges in the goal tree part, the vertices of the first group are now connected to the start tree correctly. The second group, which are connected to their roots via at least one of the path vertices, they are now connected to x_{start} instead of x_{goal}. It is due to the fact that when they are going back to their root, they need to pass via at least one of the path vertices; once they reached the path vertices, they will be directed to x_{start}. The third group, which are connected to x_{goal} without passing from any of the path vertices, they are also connected to the x_{start}. When they start going back to their root, they will reach x_{goal}, which is now a part of the start tree. Therefore, all the vertices of the goal tree are connected to the start tree correctly.

Figure 5.1 shows an example of merging two trees by implementing the presented

methodology. Figure 5.1a shows the two trees before start merging them. The start tree highlighted by blue, and the goal tree highlighted by orange. The vertex E is connection vertex, x_{cxn}. The path between vertex A, as x_{start}, and L as x_{goal}, is shown in Figure 5.1b, in which the vertices and the edges of the path are highlighted by green color.

The first step is to remove the edge of x_{cxn}, E, and its parent, F, in the goal tree, and add another edge in which x_{cxn}, E, will be the parent of F (Figure 5.1c).

Similar to the first step, all other path edges that belong to the goal tree must be removed from the edge set, and their new versions in which the rule of child and parent are swapped must be added. Therefore, the next change is to remove G from its parent, K, and connect it as a child to F (Figure 5.1d). By replacing the parent of G, vertex H is now connected to the start tree correctly, because, in its way back to the root, it comes to vertex G so that it will be directed to x_{start}, vertex A, instead of going to x_{goal}, vertex L.

Similarly, K must be disconnected from L and then connected to G as one of its children. By doing so, J will be connected to the start tree, too (Figure 5.1e).

The final step is to add L, x_{goal}, as a child to K (Figure 5.1f). As a result, all other vertices that are connected to x_{goal}, such as I, have their path back to x_{start}, vertex A.

The $CombineTwoTrees$ function is presented in Algorithm 16. At the first, the function creates a union of both trees into one (Line 2, Line 3, and Line 4). The function saves the goal tree parent of the connection point in $child$ (Line 5), and stores the connection point itself in $newParent$ (Line 6). It then creates a loop, which its condition is to be in run until $child$ is pointing to $null$ (Line 7). During this loop, the child will be removed from its old parent in the goal tree, $oldParent$, and then it should be connected to its new parent, $newParent$. So, in Line 8, the child's parent is stored in $oldParent$. Then, the edge, which connects the $oldParent$ and the $child$ is removed from the edge set (Line 9). Instead, another edge, which is connecting the $child$ to its new parent, $newParent$, will be added

Algorithm 16 $CombineTwoTrees$ Function

1: **function** $CombineTwoTrees(G_a,\ G_b,\ c_{best},\ X_{soln},\ x_{cxn})$
2: $E \leftarrow E_a \bigcup E_b$;
3: $V \leftarrow V_a \bigcup V_b$;
4: $G = (V,\ E)$;
5: $child = x_{cxn}.ParentInGoalTree$;
6: $newParent = x_{cxn}$;
7: **while** $isNotNull(child)$ **do**
8: $oldParent = child.parent$;
9: $E \leftarrow E \setminus \{(oldParent,\ child)\}$;
10: $E \leftarrow E \bigcup \{(newParent,\ child)\}$;
11: $newParent = child$;
12: $child = oldParent$;
13: **end while**
14: $X_{soln} \leftarrow \{v \in V \mid v \in X_{goal}\}$;
15: $c_{best} \leftarrow min_{x_{soln}} \in X_{soln}\{Cost(x_{soln})\}$;
16: **return** $[G,\ c_{best},\ X_{soln}]$;
17: **end function**

to the edge set (Line 10). So, the $child$ is now connected to the start tree properly. For the next iteration, the $child$ will be considered as the parent for its $oldParent$. Therefore, in Line 11, the $child$ will be stored in the $newParent$, and in Line 12, the $oldParent$ will be stored in the $child$. It loop continues until the root of the goal tree is added to the start tree properly. Afterward, the function updates the solution set by adding all vertices, which are located within the goal location (Line 14), and it also updates the best cost by checking the shortest path cost (Line 15). Finally, it returns the tree, the solution set, and the best cost.

5.2.1.3 *OptimizeTree* Function

In this stage, the planner has found a solution by using the dual-tree search and then merge the two trees into one. Therefore, it is time to optimize the solution. There are several methods for optimizing the current solution of RRT-based methods. Among them, informed sampling has shown a significant impact by limiting the state space to one of its subsets to make exploring area smaller, which helps the planner return near-optimal

solutions faster than other methods.

Algorithm 17 $OptimizeTree$ Function

```
1: function OptimizeTree(G, x_start, x_goal, c_best, X_soln)
2:     for i = 1 to n do
3:         x_rand ← InformedSample(x_start, x_goal, c_best);
4:         if Extend*(G, x_rand) ≠ Trapped then
5:             if x_new ∈ X_goal then
6:                 X_soln ← X_soln ⋃ {x_new}
7:             end if
8:             previous_c_best ← c_best;
9:             c_best ← min_{x_soln ∈ X_soln} {Cost(x_soln)};
10:            if c_best < previous_c_best then
11:                PruneTree(V, E, c_best);
12:            end if
13:        end if
14:    end for
15:    return G;
16: end function
```

Algorithm 17 outlines the steps of the $OptimizeTree$ function, which uses Informed sampling to optimize the tree. This function gets the tree, the start location, x_{start}, the goal location, x_{goal}, the best solution cost, c_{best}, and the solution set, X_{Soln} (Line 1). Line 2 creates a loop for a limited number of iterations. This loop can have several stoppage criteria, such as a limited number of iterations, a limited amount of time, or a targeted solution cost. The $InformedSample$ function is called to return a sample within the subset, and then the returned value will be stored in the x_{rand} (Line 3). Line 4 calls the $Extend^*$ function and pass the tree and the x_{rand} to it so as to extend the tree toward the x_{rand}. If the $Extend^*$ function returns $Trapped$, then the x_{rand} cannot be added to the tree so that the $OptimizeTree$ function should get another sample to be able to expend the tree. Otherwise, the tree gets new vertex, x_{new}. In Line 5, the newly added vertex, x_{new}, will be checked to find out whether it is located within the goal location. If so, the x_{new} is added to the solution set, X_{soln} (Line 6). Before updating the c_{best}, its previous

value needs to be stored to be able to be compared with the updated one (Line 8). Line 9 recalculates the c_{best}. Afterward, the previous c_{best}, $previous\_c_{best}$, and the updated one, c_{best}, are compared together (Line 10). If its value gets smaller, then the function prunes the tree to keep the tree as small as possible (Line 11). Finally, when the iterations are finished, the function returns the tree, G (Line 15).

The $InformedSample$ function gets x_{start}, x_{goal}, and c_{best} and then returns a sample within the informed set. This sampling process is outlined in Algorithm 18. Line 2 calculates the minimum possible path between the start location, x_{start}, and the goal location, x_{goal}, which is a straight line between these two configurations. In the next line, Line 3, the center point of the subset, x_{center}, is calculated. Line 4 obtained the rotation matrix, $\mathbf{C}$. The function calculates the transformation matrix, $\mathbf{L}$ (Line 5 ~ Line 7). Afterward, a random sample within a unit ball needs to be taken, x_{ball} (Line 8). Then, the x_{ball} needs to be transformed to the world frame and stored in x_{rand} (Line 9). Finally, the function returns the x_{rand} (Line 10).

Algorithm 18 $InformedSample$ Function

1: **function** $InformedSample(x_{start},\ x_{goal},\ c_{best})$
2: $\quad c_{min} \leftarrow \| x_{goal} - x_{start} \|_2$;
3: $\quad x_{center} \leftarrow (x_{start} + x_{goal})/2$;
4: $\quad \mathbf{C} \leftarrow RotationToWorldFrame(x_{start},\ x_{goal})$;
5: $\quad r_1 \leftarrow c_{best}/2$;
6: $\quad \{r_i\}_{i=2,\ldots,n} \leftarrow (\sqrt{c_{max}^2 - c_{min}^2})/2$;
7: $\quad \mathbf{L} \leftarrow diag\{r_1,\ r_2,\ \ldots,\ r_n\}$;
8: $\quad x_{ball} \leftarrow SampleUnitBall$;
9: $\quad x_{rand} \leftarrow (\mathbf{C}\mathbf{L}x_{ball} + x_{center}) \bigcap X$;
10: $\quad$ **return** x_{rand};
11: **end function**

The $Extend^*$ function is the optimized version of the $Extend$ function. It includes the rewiring process. The optimization process first tries to connect the x_{new} to a vertex that offers the shortest path to the root. It then considers x_{new} as a potential parent for

its near vertices. In other words, the parents' costs of the near vertices are compared to the x_{new} cost. If the x_{new} offers a shorter path to the root, then the parents of near vertices will be changed to the x_{new}. The output of this function has three different options, $Reached$, $Advanced$, and $Trapped$. $Reached$ is when the x_{rand} has been added to the tree. $Advanced$ will be the output when the function could not add the x_{rand} to the tree, but it added another vertex to the tree, which is in the direction of the x_{rand} but nearer to the tree. $Trapped$ will be returned by the function whenever neither the x_{rand} nor any other vertices could be added to the tree due to the presence of an obstacle. Algorithm 19 outlines the $Extend^*$ function.

Algorithm 19 Extend* Function

```
1: function Extend*(G = (V, E), x)
2:     x_nearest ← Nearest(G, x);
3:     x_new ← Steer(x_nearest, x);
4:     if isCollisionFree(x_nearest, x_new) then
5:         V ← V ∪ {x_new};
6:         x_min ← x_nearest;
7:         X_near ← Near(G, x_new, r_RRT*);
8:         c_min ← Cost(x_nearest, G) + Cost(Line(x_nearest, x_new));
9:         for each x_near ∈ X_near \ x_nearest do
10:            if isCollisionFree(x_near, x_new) & (Cost(x_near, G) + Cost(Line(x_near, x_new)) < c_min) then
11:                x_min ← x_near;
12:                c_min ← Cost(x_near, G) + Cost(Line(x_near, x_new));
13:            end if
14:        end for
15:        E ← E ∪ {x_min, x_new};
16:        for each x_near ∈ X_near \ x_min do
17:            if isCollisionFree(x_near, x_new) & (Cost(x_new, G) + Cost(Line(x_new, x_near)) < Cost(x_near, G)) then
18:                x_parent ← Parent(x_near, G);
19:                E ← E \ {(x_parent, x_near)};
20:                E ← E ∪ {(x_new, x_near)};
21:            end if
22:        end for
23:        if (x_new = x) then
24:            return Reached;
25:        else
26:            return Advanced;
27:        end if
28:    end if
29:    return Trapped;
30: end function
```

$PruneTree$ is removing the vertices of the tree which are not able to provide better solutions than the current one. This function removes the leaves of the tree that have the

estimated cost, $\hat{f}(v) = \hat{g}(v) + \hat{h}(v)$, more than the current shortest path cost, c_{best}. This procedure is presented in Algorithm 20.

Algorithm 20 *PruneTree* Function

1: **function** $PruneTree(V \subseteq X,\ E \subseteq V \times V,\ c_{best} \in \mathbb{R}_{\geqslant 0})$
2: **do**
3: $V_{prune} \leftarrow \{v \in V \mid \hat{f}(v) > c_{best},\ and\ \forall w \in V,\ (v, w) \notin E\}$;
4: $E \overset{-}{\leftarrow} \{(u, v) \in E \mid v \in V_{prune}\}$;
5: $V \overset{-}{\leftarrow} V_{prune}$;
6: **while** $V_{prune} \neq \emptyset$;
7: **end function**

5.3 Simulation

Hybrid RRT was compared to other RRT-based methods on simulated problems in $\mathbb{R}^3$ and $\mathbb{R}^6$ using Open Motion Planning Library (OMPL) (Sucan et al., 2012).

Four different state spaces are selected for the simulation, which are the built-in state spaces of the OMPL App. These four state spaces are shown in Figure 5.2 and Figure 5.3. The planners were compared together in these state spaces based on their ability to find initial solutions, and their ability to return near-optimal solutions. The planners were run 100 times in each scenario.

Simulation carried out on four different OMPL App scenarios, *BugTrap_planar* , *Maze_planar*, *Home*, and *Twistycool*. *BugTrap_planar* (Figure 5.2), and *Maze_planar*

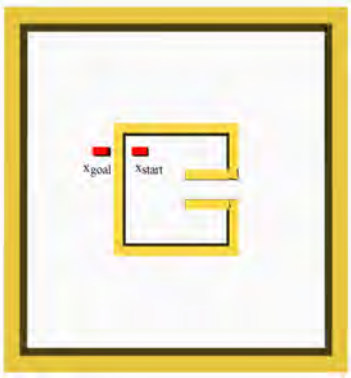
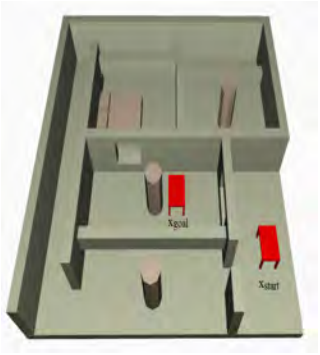
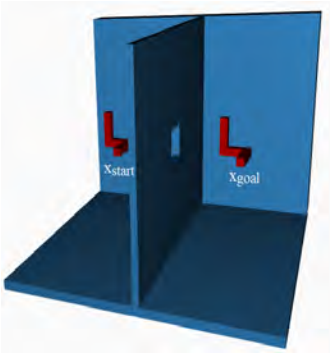


Figure 5.2: The OMPL App *BugTrap_planar* state space. It includes two rigid bodies representing the start location, x_{start}, and the goal location, x_{goal}.

(a) *Maze_planar*



(b) Home



(c) *Twistycool*

Figure 5.3: The OMPL state spaces used for the simulation. The rigid bodies are highlighted by red color. Each state space has two rigid bodies, which are representing the start location, x_{start}, and the goal location, x_{goal}.

(Figure 5.3a) are 3D state spaces, while *Home* (Figure 5.3b), and *Twistycool* (Figure 5.3c) are 6D state spaces.

Simulations are divided into two categories: the ability to find initial solutions and the ability to return near-optimal solutions.

5.3.1 Find Initial Solutions

In this test, Hybrid RRT has been compared to RRT, RRT-Connect, RRT*, Informed RRT*, and BIT*. In order to test the ability of planners in terms of finding initial solutions, each planner has unlimited time to find an initial solution. All planners found the first solutions for 100 times.

5.3.1.1 *BugTrap_planar*

BugTrap_planar, shown in Figure 5.2, is an OMPL App state space with 3 Degree of Freedoms (DoFs), including one rotation and two real vectors (x-axis and y-axis). This simulation is designed to compare the ability of different planners in terms of finding initial solutions.

As can be seen in Table 5.1, Hybrid RRT could return initial solutions with nearly the same amount of time as RRT-Connect, and faster than other planners. RRT is the

Table 5.1: The obtained results of the planners in *BugTrap_planar* scenario. Each planner was run for 100 times, and the average time needed to find the initial solutions are presented.

Planner	**Average Time (Second)**
RRT	0.411341305
RRT-Connect	0.404057516
Hybrid RRT	0.393124017
RRT*	0.994565494
Informed RRT*	0.640909516
BIT*	0.528384085

third-fastest planner in terms of finding an initial solution. In contrast, RRT* is the slowest planner, which could find initial solutions after approximately one second. It can be seen that the nonoptimized RRTs were faster than the asymptotically optimal RRT. The reason is that the asymptotically optimal versions of RRT run their rewiring procedure in each iteration so that it takes time and does not let them expand as fast as nonoptimized versions. Moreover, among the asymptotically optimal versions, BIT* is the fastest in finding an initial solution, which is due to the batches it takes to find initial solutions.

5.3.2 Find near-optimal Solutions

In this simulation, Hybrid RRT has been compared to RRT* and Informed RRT* in terms of finding near-optimal solutions in a limited time. In each scenario, the planners had limited time for finding solutions and optimizing them.

5.3.2.1 *Maze_planar*

Maze_planar (Figure 5.3a), is an OMPL App state space with 3 Degree of Freedoms (DoFs), including one rotation and two real vectors (x-axis and y-axis). The planners had 5 seconds to solve this problem in each run. The experiment specifications are listed below:

- **robot**: *car2_planar_robot*
- **environment**: *Maze_planar_env*
- **start state (x, y, yaw)**: (0.01, -0.15, 0.00)

- **goal state (x, y, yaw)**: (41.01, -0.15, 0.00)
- **memory_limit**: 1000MB
- **runtime_limit**: 5s
- **run_count**: 100

5.3.2.2 *Home*

Home is 6DoFs problems (3 coordinate planes (x, y, z) and their rotations (roll, pitch, yaw)) Figure 5.3b. In order to return near-optimal solutions, the planners must pass through the window located between x_{start} and x_{goal}. Ten seconds had been given to planners to solve this problem in each run.The experiment specifications are listed below:

- **robot**: *Home_robot*
- **environment**: *Home_env*
- **start state (x, y, z)**: (252.95, -214.95, 46.19)
- **goal state (x, y, z)**: (2.95, -100.00, 46.19)
- **memory_limit**: 1000MB
- **runtime_limit**: 10s
- **run_count**: 100

5.3.2.3 *Twistycool*

Twistycool is a 6DoFs problem (Figure 5.3c), which is difficult to be solved due to offering only a small passage to connect x_{start} to x_{goal}. There is a wall in the middle of the map, and it has only a small window, which is the only passage through the wall so that the planners need to find it to solve the problem. Due to the difficulty of this scenario, each planner had 100 seconds to solve the problem in each run. The experiment specifications are listed below:

- **robot**: *Twistycool_robot*

- **environment**: *Twistycool_env*
- **start state (x, y, z)**: (270.0, 160.0, -200.0)
- **goal state (x, y, z)**: (270.0, 160.0, -400.0)
- **memory_limit**: 1000MB
- **runtime_limit**: 100s
- **run_count**: 100

5.3.3 The proposed methods comparison

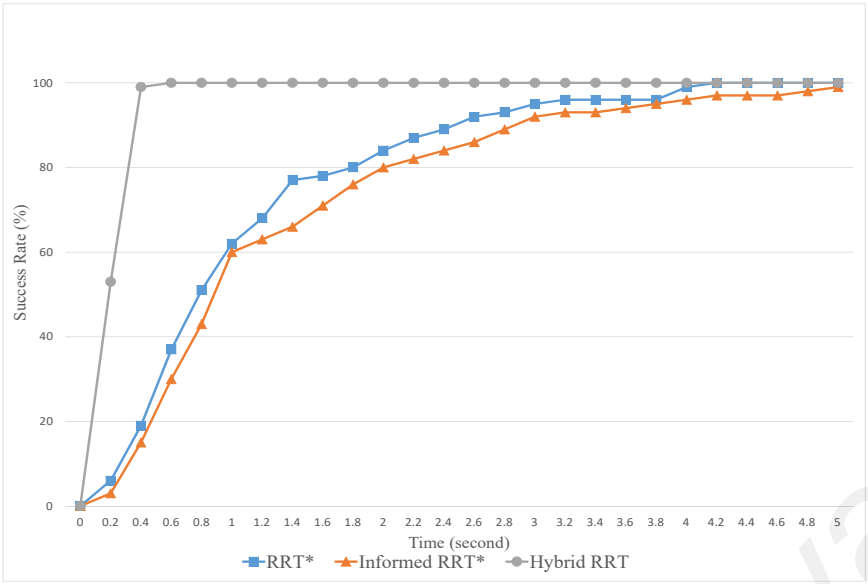
In this section the proposed methods, Informed RRT*-Connect and Hybrid RRT, were compared together in the OMPL app scenarios (Figure 5.3).

5.4 Discussion

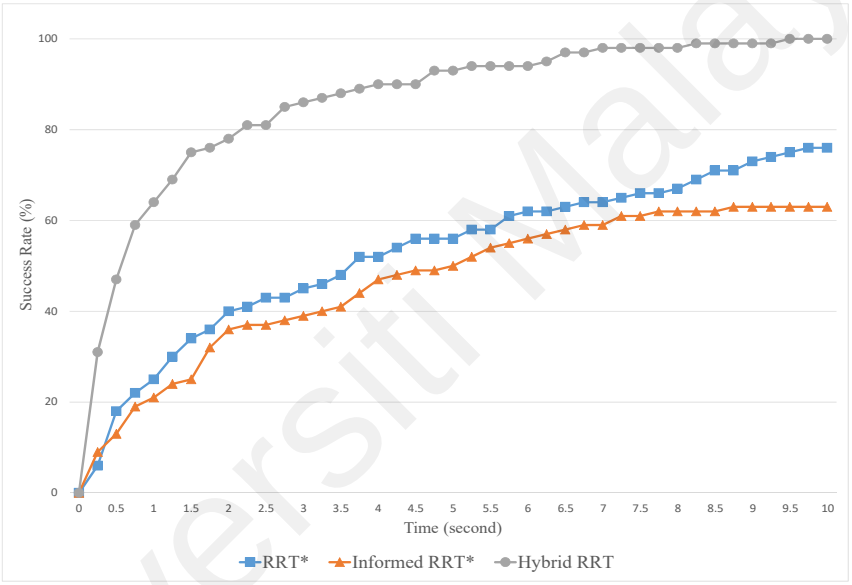
The evaluation of the simulation is represented in this section, which includes several parts, finding initial solution (Section 5.4.1), comparison of Hybrid RRT with the existing methods (Section 5.4.2), comparison of the proposed methods together (Section 5.4.3), and summary (Section 5.4.4).

5.4.1 Finding initial solutions

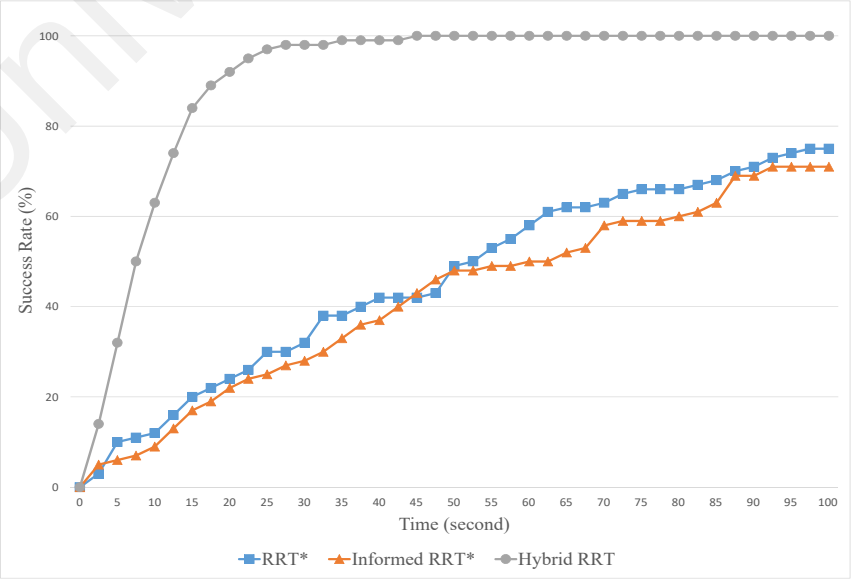
In the first simulation, finding initial solutions, it can be seen that the optimized methods (RRT*, Informed RRT*, and BIT*) are slower than nonoptimized versions. It is due to the fact that the optimized versions try to rewire their trees from the early stage of planning so that this process takes time and does not let them expand their trees as fast as nonoptimized versions. Hybrid RRT postpones the rewiring trees until it finds an initial solution. Then, it starts optimizing its solutions. This ability makes Hybrid RRT a fast planner in terms of finding initial solutions.



(a) Maze planar

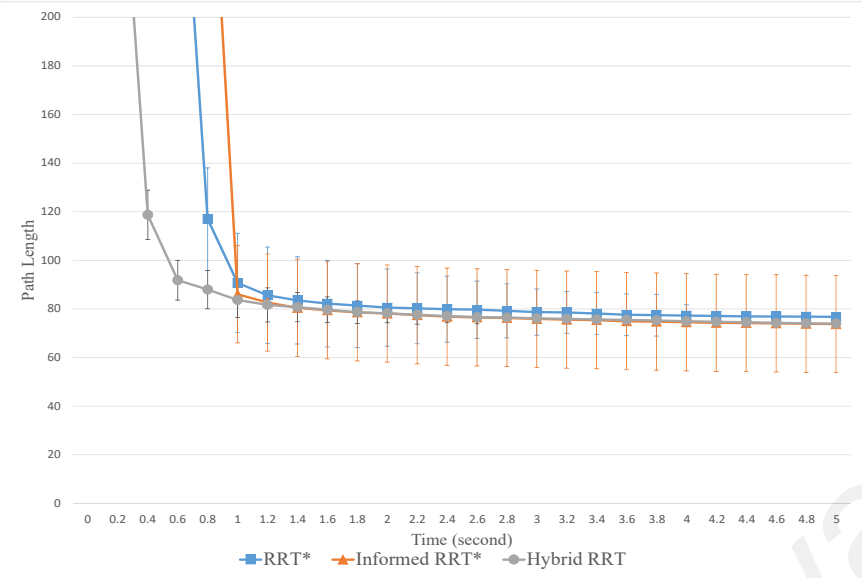


(b) Home

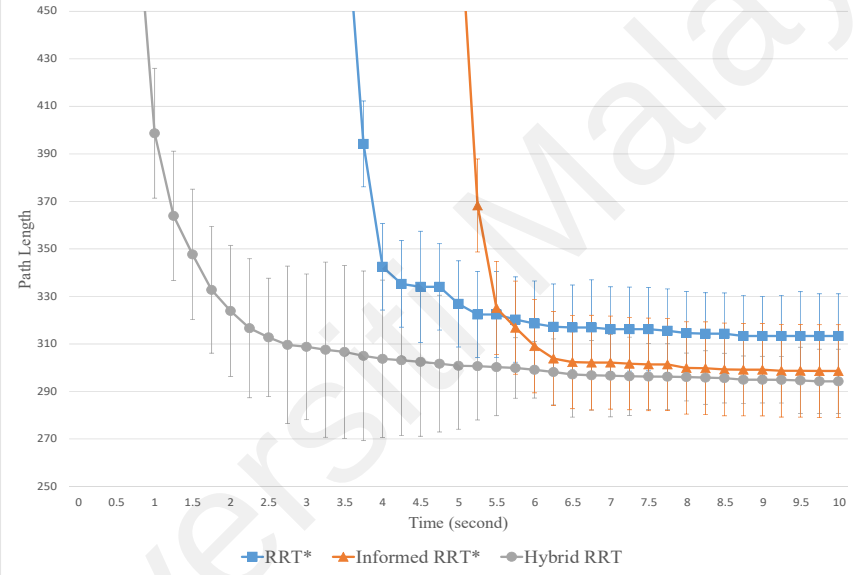


(c) Twistycool

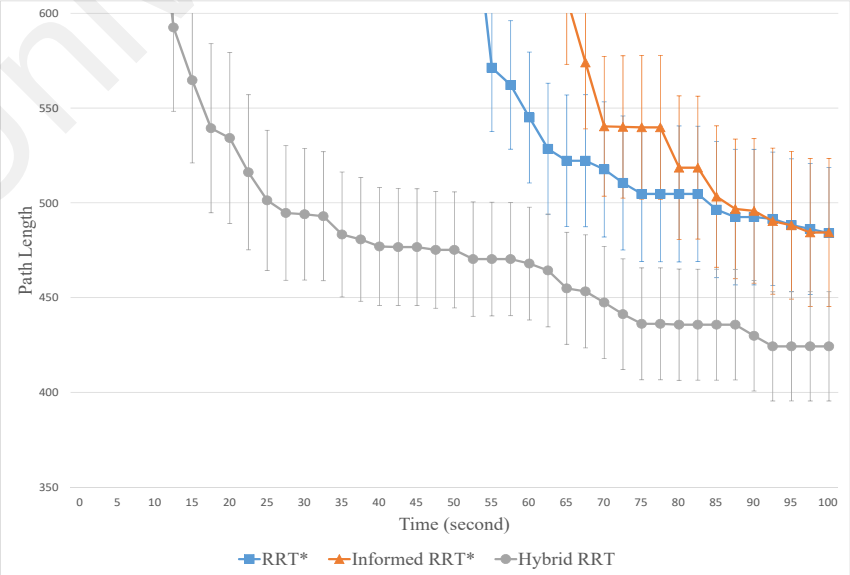
Figure 5.4: The rate of success of the three planners versus time on all the scenarios.



(a) Maze planar

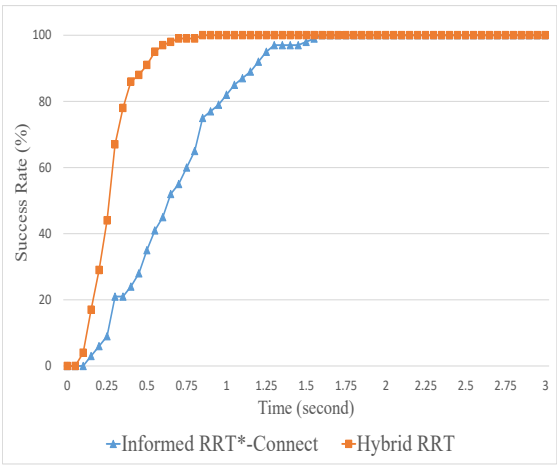


(b) Home

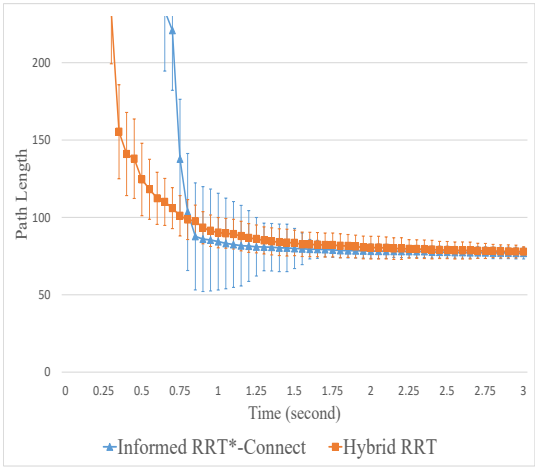


(c) *Twistycool*

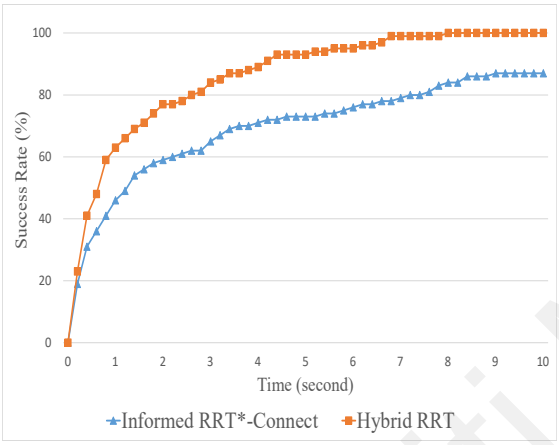
Figure 5.5: The solution cost versus time of the three planners on all the scenarios. Error bars represent a nonparametric 95% confidence interval for median solution cost.



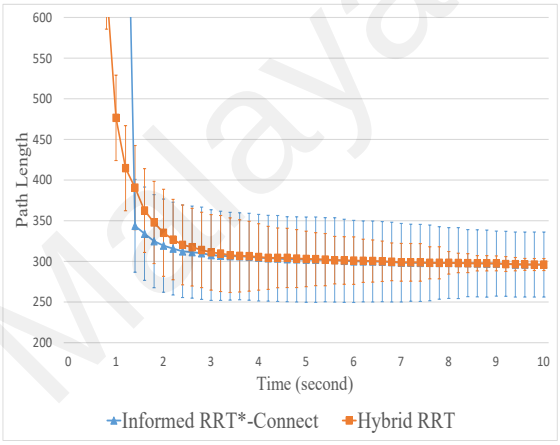
(a) Maze planar success rate



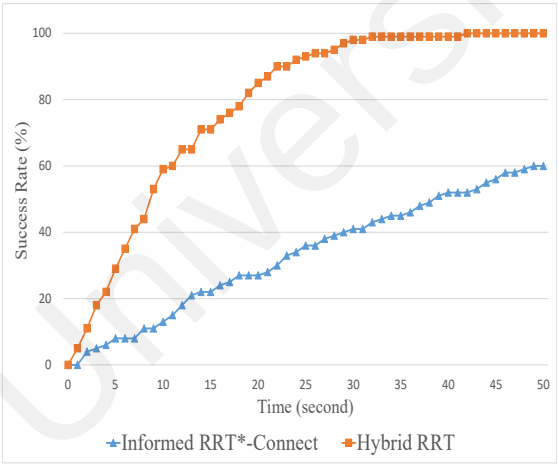
(b) Maze planar path length



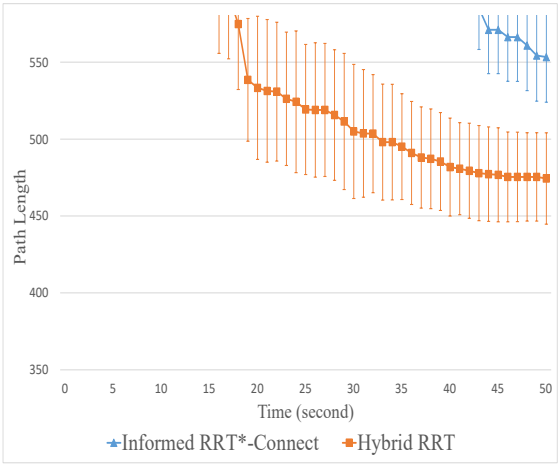
(c) Home success rate



(d) Home path length



(e) *Twistycool* success rate



(f) *Twistycool* path length

Figure 5.6: Comparison results of the proposed methods together in terms of success rate and path length. Error bars in path length graphs represent a nonparametric 95% confidence interval for median solution cost.

5.4.2 Hybrid RRT comparison with the existing methods

The next three simulations were carried out to compare the ability of RRT*, Informed RRT*, and Hybrid RRT in terms of returning near-optimal solutions. The success rates over time of planners in three scenarios are shown in Figure 5.4, and the median of path lengths are shown Figure 5.5.

Table 5.2 presents the obtained results of these simulations.

For the first OMPL App scenario, *Maze Planar*, the planning time was five seconds. The median path cost of Hybrid RRT and Informed RRT* were almost the same, while RRT* achieved longer paths in this scenario so that RRT* is the least successful planner in terms of path length. In terms of success rate, both Hybrid RRT and RRT* could achieve total success, 100%, while Informed RRT* could achieve 99%. In *Home* scenario, the planning time was 10 seconds. In spite of the previous scenario, *Maze Planar*, Hybrid RRT and Informed RRT* did not obtain approximately the same result. Hybrid RRT could return 294.2 as the solution cost, while Informed RRT* and RRT* could return 298.6 and 313.3, respectively. In terms of success rate, Hybrid RRT was able to achieve the total success, 100%, while Informed RRT* could only achieve 63%, which makes it the least successful planner in this scenario. For the most challenging scenario, the *Twistycool*, the time was 100 seconds. Informed RRT* and RRT* were able to return almost the same value as their median path length, which is about 484. On the other hand, Hybrid RRT could return a better median path length, 424, which is nearly 60 units less than the other planners' obtained path cost. The success rates of RRT* and Informed RRT* were 75% and 71%, respectively. In contrast, Hybrid RRT could achieve total success, 100%, in this scenario. Hybrid RRT could achieve total success in all the scenarios and offer shorter paths compared with the other planners. Using nonoptimized bidirectional search is why Hybrid RRT could outperform the other methods in the success rate. In other words,

having two trees with no procedure for optimization helps Hybrid RRT achieve initial solutions faster than other methods.

Table 5.2: The results of 100 independent runs of each planner in OMPL App scenarios.

State Space	Planning Time(s)	Planner	Median Path Length	Average Path Length	Success Rate (%)
Maze Planar	5	RRT*	76.7167445	76.61962043	100
		Informed RRT*	73.765986	77.67313142	99
		Hybrid RRT	73.946069	74.00716048	100
Home	10	RRT*	313.3311305	362.1013499	76
		Informed RRT*	298.605655	370.8753614	63
		Hybrid RRT	294.2446595	308.4424628	100
Twistycool	100	RRT*	484.0112635	472.0144996	75
		Informed RRT*	484.355887	463.4621787	71
		Hybrid RRT	424.2882035	359.3157852	100

In all the scenarios, Hybrid RRT could achieve 100% success rate, while RRT* and Informed RRT* could only achieve complete success in *Maze_planar*, which is a 3DoF problem.

Although all the planners could achieve 100% success rate in *Maze_planar*, they reached it by consuming different amounts of time. Hybrid RRT reached 100% after only 0.4s, while RRT* and Informed RRT* achieved it after approximately 4s. Therefore, Hybrid RRT acted ten times faster than other planners in *Maze_planar* scenario. In terms of path length, Hybrid RRT could achieve near-optimal solutions faster and RRT* and Informed RRT*.

In *Home* scenario, Hybrid RRT could reach 100% after about 8s, while RRT* and Informed RRT* could only achieve approximately 75% and 65% after 10s, respectively. Moreover, Hybrid RRT was the fastest planner in terms of optimizing the solution. Hybrid RRT obtained solution cost of 310 before 3s, while Informed RRT* achieved it after 6s, and RRT* could not reach this level at the end of the planning time frame.

Twistycool is the most challenging problem to be solved among all the simulated scenarios. Hybrid RRT was able to reach total success after around 45s, while RRT* and

Informed RRT* were not able to reach 100%. They could achieve nearly 80% after 100s. RRT* and Informed RRT* could optimize their solution to approximately 470 after 100s, which has been obtained by Hybrid RRT after only 35s.

5.4.3 Informed RRT*-Connect versus Hybrid RRT

The proposed methods, Informed RRT*-Connect and Hybrid RRT, were compared together in OMPL App scenarios. Their obtained results have been shown in Figure 5.6. It can be seen that Hybrid RRT was more successful than Informed RRT*-Connect in terms of success rate. It is because that Hybrid RRT does not have any procedure of optimizing its trees before finding initial solutions, which help it to be faster other versions of RRT that keep rewiring their trees from the early stage of planning, such as Informed RRT*-Connect.

In the first two scenarios, *Maze Planar* (Figure 5.3a) and *Home* (Figure 5.3b), Informed RRT*-Connect and Hybrid RRT were acting almost similarly. Informed RRT*-Connect could return near-optimal solution slightly faster than Hybrid RRT, which is due to rewiring its trees from the beginning of the planning. However, in *Twistycool* scenario (Figure 5.3c), Hybrid RRT could return near-optimal solutions faster than Informed RRT*-Connect. It is because that Hybrid RRT could find its initial solution approximately around 18s, while Informed RRT*-Connect could find its initial solution after about 40s. It means that Hybrid RRT was about two times faster than Informed RRT*-Connect, and then it has enough time to optimize its solution.

Table 5.3 presents the obtained results of these simulations. Both planners had three seconds to solve the first scenario, *Maze Planar*. Their obtained results are approximately the same in both the path length and the success rate. Although both planners were 100% successful in this scenario, they achieved it with different amounts of time. Hybrid RRT could get the total success around 0.75 seconds, while Informed RRT*-Connect could

achieve after around 1.5 seconds (Figure 5.6a). In contrast to the success rate, in path length, it is Informed RRT*-Connect that could get near-optimal solution slightly faster than Hybrid RRT (Figure 5.6b). In the second OMPL scenario, *Home*, both planners could achieve 296.07 as their path length. Although they could return the same path length, they were different in terms of success rate. Hybrid RRT could obtain total success, 100%, while Informed RRT*-Connect could achieve 87% in this scenario. Like the previous scenario, Informed RRT*-Connect could reach the near-optimal solution slightly faster than Hybrid RRT (Figure 5.6d). In the most challenging scenario, the *Twistycool*, the proposed methods had different results in both path length and success rate. The median path length for Hybrid RRT was 474, while it was 553 for Informed RRT*-Connect. In terms of success rate, Hybrid RRT could again achieve 100%, while Informed RRT*-Connect could obtain 61%.

Table 5.3: The results of 100 independent runs of each proposed method in OMPL App scenarios.

State Space	**Planning Time (s)**	**Planner**	**Median Path Length**	**Average Path Length**	**Success Rate (%)**
Maze Planar	3	Informed RRT*-Connect	76.8861285	79.66400145	100
		Hybrid RRT	77.850624	81.36165075	100
Home	10	Informed RRT*-Connect	296.0728605	376.6897815	87
		Hybrid RRT	296.0712065	305.4543328	100
Twistycool	50	Informed RRT*-Connect	553.34607	485.9692609	61
		Hybrid RRT	474.415751	434.2218751	100

5.4.4 Summary

This chapter presented a new path planner, Hybrid RRT, which combines the abilities of bidirectional RRTs, unidirectional RRTs, nonoptimized RRTs, and asymptotically optimal RRTs to be able to surpass them. Hybrid RRT implements a bidirectional search to find an initial solution faster than unidirectional methods. Then, it merges its two trees into

one so as to optimize it via implementing informed sampling for a single tree. In other words, Hybrid RRT divides planning time into three different phases. Phase one is about finding initial solutions. Nonoptimized RRTs normally obtain initial solutions faster than asymptotically optimal versions of RRT. Moreover, bidirectional RRTs are faster than unidirectional RRTs in terms of finding initial solutions. Therefore, Hybrid RRT uses a bidirectional nonoptimized RRT to find initial solutions. Although having two trees help planners to find initial solutions faster, they may take more time to optimize their both trees in comparison with unidirectional RRTs. As a result, Hybrid RRT implements a unidirectional asymptotically RRT, which uses informed sampling. This helps Hybrid RRT return near-optimal solutions faster. Thus, the second phase is to combine the two trees of phase one into one tree. Afterward, Hybrid RRT has a single tree, which has an initial solution. In phase three, Hybrid RRT limits the search area into one of the state space subsets, which its size is defined based on the length of the initial solution. It then takes samples within the subset in order to improve the solution quality.

The simulations show that Hybrid RRT outperforms RRT* and Informed RRT* in terms of the success rate as well as optimization time. It has a higher success rate, and it could return near-optimal solutions faster. Moreover, Hybrid RRT can return initial solutions faster than Informed RRT*-Connect, and it can also return near-optimal solutions faster Informed RRT*-Connect in scenarios in which finding initial solutions is challenging. These abilities make Hybrid RRT suitable for various motion planning problems.

CHAPTER 6: DISCUSSION AND CONCLUSION

Autonomous robots are working in unknown and uncontrolled environments. Therefore, they need to have a motion planner to guide them to reach their desired locations. Motion planners must find high-quality and collision-free paths in a reasonable amount of time. Accomplishing this task is challenging due to the problem domain. Most environments are continuous so that planners discretize them to have a finite number of samples. Mostly, planners are either graph-based or sampling-based to discretize their environments.

Approximating an environment by a graph is challenging due to the relationship between resolution and search performance. If the resolution is selected too high, then finding solutions for problems will take a long time. On the other hand, if the resolution is selected too low, then the solution will be found faster, but it is low-quality.

Although there are several limitations to graph-based methods, they are efficient in their searches, such as A*. This planner uses heuristics intending to prioritize the search area based on the potential solution quality.

Sampling-based methods do not make a graph from state spaces before start planning. They make an anytime approximation from the search space, which helps them avoid any resolution decision before they start planning. Moreover, sampling-based methods can be run indefinitely until they find suitable paths.

There are two types of sampling-based methods: Multi-query and single-query methods. Multi-query planners such as Probabilistic Road Map (PRM) (Kavraki et al., 1996) can solve several problems with different start locations and goal locations in the state space. They first create a roadmap by taking random samples, then connecting different locations through the created roadmap. Therefore, planning time for multi-query planners is divided into two phases, the learning phase and the query phase. In the learning phase, the planner

creates its roadmap from the state space by taking random samples and connecting them together. In the query phase, the planner receives several the start and the goal locations and then try to connect these locations to the roadmap and then find a path between each pair of the start and the goal locations.

Single-query planners such as Rapidly-exploring Random Tree (RRT) (LaValle, 1998) do not make a roadmap like multi-query planners. Instead, unidirectional RRTs construct a tree rooted in the start location and explore the state space by growing the tree toward random samples. The exploring process will be stopped once the goal location is sampled. They do not consume time for any learning phases. In other words, they keep searching the state space until either an appropriate result is found or the planning time is over. RRT-based planners are probability complete, which means that they have a unity probability of finding a solution, if one exists, with an infinite number of samples.

In spite of the benefits of unidirectional RRTs, they need more time to find initial solutions in comparison with bidirectional RRTs. Bidirectional RRT-based methods, such as RRT-Connect (Kuffner & LaValle, 2000), implements two trees from the start location and the goal location. These two trees try aggressively to make a connection between themselves. Using two trees makes RRT-Connect a faster motion planner compared to RRT in terms of finding initial solutions.

Although RRT-based methods can solve the motion planning problems efficiently, they provide nonoptimal solutions (Karaman & Frazzoli, 2011). It is due to the fact that they explore the state space with the random walk so that their outputs would be a sequence of random samples, and they do not have any procedure for optimizing their trees.

RRT* (Karaman & Frazzoli, 2011) implements a rewiring procedure, which leads to near-optimal solutions. RRT*-based motion planners are called almost-surely asymptotically optimal, which means that they will return near-optimal solutions by increasing the number

of samples. RRT* does not stop exploring state spaces after a solution is found. It continues exploring the state space with the aim of returning better solutions than the current one. RRT* keeps sampling all over the state space to optimize the current solution so that it is an inefficient way due to its single-query nature (Gammell et al., 2014). In other words, it is better to distribute the samples over a region of the environment that has more probability of improving the solution quality.

Gammell *et al.* (Gammell et al., 2014) combined the heuristics of graph-based with the random sampling of sampling-based to propose a subset of the state space in which better solutions can be found. They proposed Informed RRT* (Gammell et al., 2018, 2014), which works like RRT* before finding an initial solution. Afterward, it limits the search area to an ellipsoidal subset of the state space.

Although Informed RRT* can return near-optimal solutions faster than the standard version of RRT*, it only expedites the optimization process. It still has the problems of other unidirectional methods, which is spotting a sample in the goal area, especially when the goal area is hidden beyond the narrow passages.

Chapter 4 proposed a new motion planner, Informed RRT*-Connect. This planner is a bidirectional RRT-based motion planner. It starts exploring the environment by implementing two trees, one from the start location, another from the goal location. These two trees are optimizing themselves like RRT*. After an initial solution is found, Informed RRT*-Connect search area will be limited to an ellipsoidal subset of the state space which its eccentricity depends on the length of the shortest solution. Limiting state spaces to subsets gives the ability to the planner to return near-optimal solutions with fewer iterations. Moreover, it is a dual-tree motion planner so that it can find first solutions faster than unidirectional searches, such as Informed RRT* so that it can reach the subset of the state space faster than Informed RRT*.

The comparison has been carried out in terms of success rate and path length. Informed RRT*-Connect has been tested in various scenarios. In the easiest scenario, which is shown in Figure 4.7a, all planners were successful in finding initial solutions. In this scenario, informed RRT*-Connect could 100% successful, while the second most successful was Informed RRT* by 97%. In the most challenging scenario (Figure 4.9c), Informed RRT*-Connect success rate was 93%, while Informed RRT* was only 1%. Therefore, Informed RRT*-Connect is a better anytime planner than the existing methods. This planner has been published:

- Mashayekhi, Reza, et al. "Informed RRT*-Connect: An Asymptotically Optimal Single-Query Path Planning Method." IEEE Access 8 (2020): 19842-19852.

Chapter 5 proposed another motion planner that is a semi-bidirectional RRT-based planner, Hybrid RRT. Almost all RRT-based methods can be divided into two categories, nonoptimized versions, and optimized versions. Nonoptimized versions such as RRT and RRT-Connect are used to find initial solutions. On the other hand, optimized versions like RRT* and Informed RRT* have been designed to return near-optimal solutions so that they keep optimizing their trees from the beginning of the planning until the end. However, it makes them slower than nonoptimized versions of RRT in terms of finding initial solutions. In Addition to these categories, most RRT-based methods are also categorized into two groups: unidirectional and bidirectional.

Hybrid RRT divides the planning time into three phases. Phase one is to find an initial solution, the second phase is to combine two trees of phase one into one tree, and phase three is to optimize the solution. Hybrid RRT implements a bidirectional nonoptimal RRT to achieve the first solutions faster than unidirectional methods. After finding the first solution, Hybrid RRT needs to merge its two trees into one. Then, it optimizes the tree to

find near-optimal solutions.

In order to achieve fast results out from the optimization process, Hybrid RRT limits the state space into a subset of the state space like Informed RRT*. Therefore, it needs to combine two trees of phase one into one tree to be able to implement Informed sampling on a single tree.

Hybrid RRT is neither an entirely unidirectional method nor a fully bidirectional one. It is a combination of both groups. Moreover, it is neither a nonoptimized version nor an optimized version. It uses a nonoptimized search for finding initial solutions, which make it faster than the optimized versions of RRT. Moreover, it implements an optimization process to be able to return near-optimal solutions. Hybrid RRT can find first solutions as fast as RRT-Connect and returns the near-optimal solutions as quickly as Informed RRT*.

The comparison has been carried out in terms of success rate and path length. Hybrid RRT has been tested in various scenarios. The easiest scenario (Figure 5.3a), in which all planners were almost 100% successful. Although all planners were successful in this scenario, they achieved complete success with different amounts of time. Hybrid RRT could reach complete success after only 0.4s, while the second-fastest planner, RRT*, could achieve complete success approximately after 4s. It means that Hybrid RRT was ten times faster than the second-fastest planner. In all other scenarios, Hybrid RRT was the fastest in terms of success rate.

In terms of path length, the most challenging scenario was Figure 5.3c in which RRT* and Informed RRT* could achieve 475 at the end of planning time, 100s. In comparison, Hybrid RRT could achieve this path length after only 35s. Hybrid RRT was about three times faster than the second-fastest planner in terms of path length. This planner has been published:

- Mashayekhi, Reza, et al. "Hybrid RRT: A Semi-Dual-Tree RRT-Based Motion

Planner." IEEE Access 8 (2020): 18658-18668.

The proposed methods were compared with each other in OMPL App scenarios (Figure 5.3). In the first scenario, *Maze Planar* (Figure 5.3a), and the second scenario, *Home* (Figure 5.3b), Hybrid RRT was more successful in terms of success rate. It achieved 100% in both scenarios, while Informed RRT*-Connect could only achieve 100% in the first scenario. Although both planners could achieve total success in the first scenario, Hybrid RRT achieved almost two times faster than Informed RRT*-Connect. On the other hand, in terms of path length, Informed RRT*-Connect acted slightly faster than Hybrid RRT to reach a near-optimal solution. Informed RRT*-Connect keeps rewiring its two trees from the beginning of the planning time, while Hybrid RRT starts rewiring after it finds an initial solution. It is the reason that Informed RRT*-Connect could offer slightly shorter paths in the first two benchmarking scenarios. Finally, in the most challenging scenario, *Twistycool* (Figure 5.3c), Hybrid RRT could surpass Informed RRT*-Connect is both path length and success rate.

This thesis proposed two new single-query almost-surely asymptotically optimal motion planners that outperform the state-of-the-art motion planners.

REFERENCES

Aguilar, W. G., Morales, S., Ruiz, H., & Abad, V. (2017). Rrt* gl based optimal path planning for real-time navigation of uavs. In *International work-conference on artificial neural networks* (pp. 585–595).

Aguinaga, I., Borro, D., & Matey, L. (2008). Parallel RRT-based path planning for selective disassembly planning. *The International Journal of Advanced Manufacturing Technology*, *36*(11-12), 1221–1233.

Aine, S., & Likhachev, M. (2016). Truncated incremental search. *Artificial Intelligence*, *234*, 49–77.

Aine, S., Swaminathan, S., Narayanan, V., Hwang, V., & Likhachev, M. (2016). Multi-heuristic a. *The International Journal of Robotics Research*, *35*(1-3), 224–243.

Akgun, B., & Stilman, M. (2011). Sampling heuristics for optimal motion planning in high dimensions. In *2011 ieee/rsj international conference on intelligent robots and systems* (pp. 2640–2645).

Amato, N. M., & Wu, Y. (1996). A randomized roadmap method for path and manipulation planning. In *Proceedings of ieee international conference on robotics and automation* (Vol. 1, pp. 113–120).

Arslan, O., & Tsiotras, P. (2013). Use of relaxation methods in sampling-based algorithms for optimal motion planning. In *2013 ieee international conference on robotics and automation* (pp. 2421–2428).

Ayawli, B. B. K., Mei, X., Shen, M., Appiah, A. Y., & Kyeremeh, F. (2019). Optimized rrt-a* path planning method for mobile robots in partially known environment. *Information Technology and Control*, *48*(2), 179–194.

Bellman, R. (1954). *The theory of dynamic programming* (Tech. Rep.). Rand corp santa monica ca.

Bellman, R. (1958). On a routing problem. *Quarterly of applied mathematics*, *16*(1), 87–90.

Bellmann, R. (1957). Dynamic programming princeton university press. *Princeton, NJ*.

Bertsekas, D. (1975). Convergence of discretization procedures in dynamic programming. *IEEE Transactions on Automatic Control*, *20*(3), 415–419.

Bohlin, R., & Kavraki, L. E. (2000). 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, pp. 521–528).

Boor, V., Overmars, M. H., & Van Der Stappen, A. F. (1999). The Gaussian sampling strategy for probabilistic roadmap planners. In *Proceedings 1999 ieee international conference on robotics and automation (cat. no. 99ch36288c)* (Vol. 2, pp. 1018–1023).

Branicky, M. S., LaValle, S. M., Olson, K., & Yang, L. (2001). Quasi-randomized path planning. In *Proceedings 2001 icra. ieee international conference on robotics and automation (cat. no. 01ch37164)* (Vol. 2, pp. 1481–1487).

Burfoot, D., Pineau, J., & Dudek, G. (2006). RRT-Plan: A Randomized Algorithm for STRIPS Planning. In *Icaps* (pp. 362–365).

Burns, B., & Brock, O. (2007). Single-query motion planning with utility-guided random trees. In *Proceedings 2007 ieee international conference on robotics and automation* (pp. 3307–3312).

Chen, P. C., & Hwang, Y. K. (1998). SANDROS: a dynamic graph search algorithm for motion planning. *IEEE Transactions on Robotics and Automation*, *14*(3), 390–403.

Chitta, S., Sucan, I., & Cousins, S. (2012). Moveit![ros topics]. *IEEE Robotics & Automation Magazine*, *19*(1), 18–19.

Choudhury, S., Gammell, J. D., Barfoot, T. D., Srinivasa, S. S., & Scherer, S. (2016). Regionally accelerated batch informed trees (rabit*): A framework to integrate local information into optimal path planning. In *2016 ieee international conference on robotics and automation (icra)* (pp. 4207–4214).

Cohen, B., Phillips, M., & Likhachev, M. (2015). Planning single-arm manipulations with n-arm robots. In *Eighth annual symposium on combinatorial search.*

Devaurs, D., Siméon, T., & Cortés, J. (2015). Optimal path planning in complex cost spaces with sampling-based algorithms. *IEEE Transactions on Automation Science and Engineering*, *13*(2), 415–424.

Diankov, R. (2010). Automated construction of robotic manipulation programs.

Diankov, R., & Kuffner, J. (2007). Randomized statistical path planning. In *2007 ieee/rsj international conference on intelligent robots and systems* (pp. 1–6).

Dijkstra, E. W., et al. (1959). A note on two problems in connexion with graphs. *Numerische mathematik*, *1*(1), 269–271.

Donald, B., Xavier, P., Canny, J., & Reif, J. (1993). Kinodynamic motion planning. *Journal of the ACM (JACM)*, *40*(5), 1048–1066.

Ellekilde, L.-P., & Petersen, H. G. (2013). Motion planning efficient trajectories for industrial bin-picking. *The International Journal of Robotics Research*, *32*(9-10), 991–1004.

Estrada, A. V., Lien, J.-M., & Amato, N. M. (2006). Vizmo++: a visualization, authoring, and educational tool for motion planning. In *Proceedings 2006 ieee international conference on robotics and automation, 2006. icra 2006.* (pp. 727–732).

Ferguson, D., & Stentz, A. (2005). The delayed d* algorithm for efficient path replanning. In *Proceedings of the 2005 ieee international conference on robotics and automation* (pp. 2045–2050).

Floyd, R. W. (1962). Algorithm 97: shortest path. *Communications of the ACM*, *5*(6), 345.

Ford Jr, L. R. (1956). *Network flow theory* (Tech. Rep.). Rand Corp Santa Monica Ca.

Fragkopoulos, C., & Graeser, A. (2010). A RRT based path planning algorithm for Rehabilitation robots. In *Isr 2010 (41st international symposium on robotics) and robotik 2010 (6th german conference on robotics)* (pp. 1–8).

Fulgenzi, C., Tay, C., Spalanzani, A., & Laugier, C. (2008). Probabilistic navigation in dynamic environment using rapidly-exploring random trees and gaussian processes.

In *2008 ieee/rsj international conference on intelligent robots and systems* (pp. 1056–1062).

Gammell, J. D., & Barfoot, T. D. (2014). The probability density function of a transformation-based hyperellipsoid sampling technique. *arXiv preprint arXiv:1404.1347*.

Gammell, J. D., Barfoot, T. D., & Srinivasa, S. S. (2018). Informed sampling for asymptotically optimal path planning. *IEEE Transactions on Robotics*, *34*(4), 966–984.

Gammell, J. D., Srinivasa, S. S., & Barfoot, T. D. (2014). 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* (pp. 2997–3004).

Gammell, J. D., Srinivasa, S. S., & Barfoot, T. D. (2015). Batch informed trees (BIT*): Sampling-based optimal planning via the heuristically guided search of implicit random geometric graphs. In *2015 ieee international conference on robotics and automation (icra)* (pp. 3067–3074).

González, D., Pérez, J., Milanés, V., & Nashashibi, F. (2015). A review of motion planning techniques for automated vehicles. *IEEE Transactions on Intelligent Transportation Systems*, *17*(4), 1135–1145.

Guibas, L. J., Holleman, C., & Kavraki, L. E. (1999). A probabilistic roadmap planner for flexible objects with a workspace medial-axis-based sampling approach. In *Proceedings 1999 ieee/rsj international conference on intelligent robots and systems. human and environment friendly robots with high intelligence and emotional quotients (cat. no. 99ch36289)* (Vol. 1, pp. 254–259).

Hart, P. E., Nilsson, N. J., & Raphael, B. (1968). A formal basis for the heuristic determination of minimum cost paths. *IEEE transactions on Systems Science and Cybernetics*, *4*(2), 100–107.

Holleman, C., & Kavraki, L. E. (2000). A framework for using the workspace medial axis in prm planners. In *Proceedings 2000 icra. millennium conference. ieee international conference on robotics and automation. symposia proceedings (cat. no. 00ch37065)* (Vol. 2, pp. 1408–1413).

Hsu, D., & Sun, Z. (2004). Adaptively combining multiple sampling strategies for probabilistic roadmap planning. In *Ieee conference on robotics, automation and mechatronics, 2004.* (Vol. 2, pp. 774–779).

Ingerman, P. Z. (1962). Algorithm 141: path matrix. *Communications of the ACM*, *5*(11), 556.

Islam, F., Narayanan, V., & Likhachev, M. (2015). Dynamic multi-heuristic a. In *2015 ieee international conference on robotics and automation (icra)* (pp. 2376–2382).

Jaillet, L., Cortés, J., & Siméon, T. (2008). Transition-based RRT for path planning in continuous cost spaces. In *2008 ieee/rsj international conference on intelligent robots and systems* (pp. 2145–2150).

Jaillet, L., Yershova, A., La Valle, S. M., & Siméon, T. (2005). Adaptive tuning of the sampling domain for dynamic-domain RRTs. In *2005 ieee/rsj international conference on intelligent robots and systems* (pp. 2851–2856).

Karaman, S., & Frazzoli, E. (2011). Sampling-based algorithms for optimal motion planning. *The international journal of robotics research*, *30*(7), 846–894.

Karaman, S., Walter, M. R., Perez, A., Frazzoli, E., & Teller, S. (2011). Anytime motion planning using the RRT. In *2011 ieee international conference on robotics and automation* (pp. 1478–1483).

Kavraki, L. E., Kolountzakis, M. N., & Latombe, J.-C. (1998). Analysis of probabilistic roadmaps for path planning. *IEEE Transactions on Robotics and Automation*, *14*(1), 166–171.

Kavraki, L. E., Svestka, P., Latombe, J.-C., & Overmars, M. H. (1996). Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE transactions on Robotics and Automation*, *12*(4), 566–580.

Kim, D., Lee, J., & Yoon, S.-e. (2014). Cloud RRT*: Sampling cloud based RRT*. In *2014 ieee international conference on robotics and automation (icra)* (pp. 2519–2526).

Klemm, S., Oberländer, J., Hermann, A., Roennau, A., Schamm, T., Zollner, J. M., & Dillmann, R. (2015). RRT*-Connect: Faster, asymptotically optimal motion planning. In *2015 ieee international conference on robotics and biomimetics*

(robio) (pp. 1670–1677).

Koenig, S., & Likhachev, M. (2005). Fast replanning for navigation in unknown terrain. *IEEE Transactions on Robotics*, *21*(3), 354–363.

Koenig, S., Likhachev, M., & Furcy, D. (2004). Lifelong planning A*. *Artificial Intelligence*, *155*(1-2), 93–146.

Kuffner, J. J., & LaValle, S. M. (2000). 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, pp. 995–1001).

Kuwata, Y., Fiore, G. A., Teo, J., Frazzoli, E., & How, J. P. (2008). Motion planning for urban driving using RRT. In *2008 ieee/rsj international conference on intelligent robots and systems* (pp. 1681–1686).

Kuwata, Y., Teo, J., Fiore, G., Karaman, S., Frazzoli, E., & How, J. P. (2009). Real-time motion planning with applications to autonomous urban driving. *IEEE Transactions on control systems technology*, *17*(5), 1105–1118.

Lan, X., & Di Cairano, S. (2015). Continuous curvature path planning for semi-autonomous vehicle maneuvers using RRT. In *2015 european control conference (ecc)* (pp. 2360–2365).

Larsen, E., Gottschalk, S., Lin, M. C., & Manocha, D. (2000). Fast distance queries with rectangular swept sphere volumes. In *Proceedings 2000 icra. millennium conference. ieee international conference on robotics and automation. symposia proceedings (cat. no. 00ch37065)* (Vol. 4, pp. 3719–3726).

Larson, R. (1967). A survey of dynamic programming computational procedures. *IEEE Transactions on Automatic Control*, *12*(6), 767–774.

LaValle, S. M. (1998). Rapidly-exploring random trees: A new tool for path planning.

LaValle, S. M. (2006). *Planning algorithms*. Cambridge university press.

LaValle, S. M., & Kuffner Jr, J. J. (2001). Randomized kinodynamic planning. *The*

international journal of robotics research, *20*(5), 378–400.

Lee, J., Pippin, C., & Balch, T. (2008). Cost based planning with RRT in outdoor environments. In *2008 ieee/rsj international conference on intelligent robots and systems* (pp. 684–689).

Li, Y., Littlefield, Z., & Bekris, K. E. (2016). Asymptotically optimal sampling-based kinodynamic planning. *The International Journal of Robotics Research*, *35*(5), 528–564.

Likhachev, M., Ferguson, D., Gordon, G., Stentz, A., & Thrun, S. (2008). Anytime search in dynamic graphs.

Likhachev, M., Ferguson, D. I., Gordon, G. J., Stentz, A., & Thrun, S. (2005). Anytime Dynamic A*: An Anytime, Replanning Algorithm. In *Icaps* (Vol. 5, pp. 262–271).

Likhachev, M., Gordon, G. J., & Thrun, S. (2004). ARA*: Anytime A* with provable bounds on sub-optimality. In *Advances in neural information processing systems* (pp. 767–774).

Liu, H., Zhang, X., Wen, J., Wang, R., & Chen, X. (2019). Goal-biased Bidirectional RRT based on Curve-smoothing. *IFAC-PapersOnLine*, *52*(24), 255–260.

Lozano-Pérez, T., & Wesley, M. A. (1979). An algorithm for planning collision-free paths among polyhedral obstacles. *Communications of the ACM*, *22*(10), 560–570.

Martin, S. R., Wright, S. E., & Sheppard, J. W. (2007). Offline and online evolutionary bi-directional RRT algorithms for efficient re-planning in dynamic environments. In *2007 ieee international conference on automation science and engineering* (pp. 1131–1136).

Mashayekhi, R., Idris, M. Y. I., Anisi, M. H., & Ahmedy, I. (2020). Hybrid RRT: A Semi-Dual-Tree RRT-Based Motion Planner. *IEEE Access*, *8*, 18658–18668.

Mashayekhi, R., Idris, M. Y. I., Anisi, M. H., Ahmedy, I., & Ali, I. (2020). Informed RRT*-Connect: An Asymptotically Optimal Single-Query Path Planning Method. *IEEE Access*, *8*, 19842–19852.

Melchior, N. A., & Simmons, R. (2007). Particle RRT for path planning with uncertainty. In *Proceedings 2007 ieee international conference on robotics and automation* (pp. 1617–1624).

Moore, E. F. (1959). The shortest path through a maze. In *Proc. int. symp. switching theory, 1959* (pp. 285–292).

Nasir, J., Islam, F., Malik, U., Ayaz, Y., Hasan, O., Khan, M., & Muhammad, M. S. (2013). RRT*-SMART: A rapid convergence implementation of RRT. *International Journal of Advanced Robotic Systems*, *10*(7), 299.

Olsen, A. L., & Petersen, H. G. (2007). Motion planning for gantry mounted manipulators: A ship-welding application example. In *Proceedings 2007 ieee international conference on robotics and automation* (pp. 4782–4786).

Paden, B., Čáp, M., Yong, S. Z., Yershov, D., & Frazzoli, E. (2016). A survey of motion planning and control techniques for self-driving urban vehicles. *IEEE Transactions on intelligent vehicles*, *1*(1), 33–55.

Pan, J., Chitta, S., & Manocha, D. (2012). Fcl: A general purpose library for collision and proximity queries. In *2012 ieee international conference on robotics and automation* (pp. 3859–3866).

Pan, J., Zhang, L., & Manocha, D. (2010). Retraction-based RRT planner for articulated models. In *2010 ieee international conference on robotics and automation* (pp. 2529–2536).

Pepy, R., & Lambert, A. (2006). Safe path planning in an uncertain-configuration space using RRT. In *2006 ieee/rsj international conference on intelligent robots and systems* (pp. 5376–5381).

Perez, A., Karaman, S., Shkolnik, A., Frazzoli, E., Teller, S., & Walter, M. R. (2011). Asymptotically-optimal path planning for manipulation using incremental sampling-based algorithms. In *2011 ieee/rsj international conference on intelligent robots and systems* (pp. 4307–4313).

Persson, S. M., & Sharf, I. (2014). Sampling-based A* algorithm for robot path-planning. *The International Journal of Robotics Research*, *33*(13), 1683–1708.

Plaku, E., Bekris, K. E., & Kavraki, L. E. (2007). Oops for motion planning: An online, open-source, programming system. In *Proceedings 2007 ieee international conference on robotics and automation* (pp. 3711–3716).

Pohl, I. (1970). First results on the effect of error in heuristic search. *Machine Intelligence*, *5*, 219–236.

Pohl, I. (1973). The avoidance of (relative) catastrophe, heuristic competence, genuine dynamic weighting and computational issues in heuristic problem solving. In *Proceedings of the 3rd international joint conference on artificial intelligence* (pp. 12–17).

Quigley, M., Conley, K., Gerkey, B., Faust, J., Foote, T., Leibs, J., ... Ng, A. Y. (2009). ROS: an open-source Robot Operating System. In *Icra workshop on open source software* (Vol. 3, p. 5).

Qureshi, A. H., & Ayaz, Y. (2015). Intelligent bidirectional rapidly-exploring random trees for optimal motion planning in complex cluttered environments. *Robotics and Autonomous Systems*, *68*, 1–11.

Qureshi, A. H., & Ayaz, Y. (2016). Potential functions based sampling heuristic for optimal path planning. *Autonomous Robots*, *40*(6), 1079–1093.

Rodriguez, S., Tang, X., Lien, J.-M., & Amato, N. M. (2006). An obstacle-based rapidly-exploring random tree. In *Proceedings 2006 ieee international conference on robotics and automation, 2006. icra 2006.* (pp. 895–900).

Roy, B. (1959). Transitivité et connexité. *Comptes Rendus Hebdomadaires Des Seances De L Academie Des Sciences*, *249*(2), 216–218.

Sallaberger, C. S., & D'Eleuterio, G. M. (1995). Optimal robotic path planning using dynamic programming and randomization. *Acta Astronautica*, *35*(2-3), 143–156.

Salzman, O., & Halperin, D. (2016). Asymptotically near-optimal RRT for fast, high-quality motion planning. *IEEE Transactions on Robotics*, *32*(3), 473–483.

Srinivasa, S. S., Berenson, D., Cakmak, M., Collet, A., Dogar, M. R., Dragan, A. D., ... others (2012). Herb 2.0: Lessons learned from developing a mobile manipulator for the home. *Proceedings of the IEEE*, *100*(8), 2410–2428.

Srinivasa, S. S., Ferguson, D., Helfrich, C. J., Berenson, D., Collet, A., Diankov, R., . . . Weghe, M. V. (2010). HERB: a home exploring robotic butler. *Autonomous Robots*, *28*(1), 5.

Stentz, A. (1997). Optimal and efficient path planning for partially known environments. In *Intelligent unmanned ground vehicles* (pp. 203–220). Springer.

Stentz, A., et al. (1995). The focussed dˆ* algorithm for real-time replanning. In *Ijcai* (Vol. 95, pp. 1652–1659).

Sucan, I. A., Moll, M., & Kavraki, L. E. (2012). The open motion planning library. *IEEE Robotics & Automation Magazine*, *19*(4), 72–82.

Sudhakara, P., Ganapathy, V., & Sundaran, K. (2017). Optimal trajectory planning based on bidirectional spline-RRT* for wheeled mobile robot. In *2017 third international conference on sensing, signal processing and security (icsss)* (pp. 65–68).

Sun, H., & Farooq, M. (2002). Note on the generation of random points uniformly distributed in hyper-ellipsoids. In *Proceedings of the fifth international conference on information fusion. fusion 2002.(ieee cat. no. 02ex5997)* (Vol. 1, pp. 489–496).

Sun, Z., Hsu, D., Jiang, T., Kurniawati, H., & Reif, J. H. (2005). Narrow passage sampling for probabilistic roadmap planning. *IEEE Transactions on Robotics*, *21*(6), 1105–1115.

Teniente, E. H., & Andrade-Cetto, J. (2013). HRA*: Hybrid randomized path planning for complex 3D environments. In *2013 ieee/rsj international conference on intelligent robots and systems* (pp. 1766–1771).

Thomas, S., Morales, M., Tang, X., & Amato, N. M. (2007). Biasing samplers to improve motion planning performance. In *Proceedings 2007 ieee international conference on robotics and automation* (pp. 1625–1630).

Tong, B., Liu, Q., & Dai, C. (2019). A rrt* fn based path replanning algorithm. In *2019 ieee 4th advanced information technology, electronic and automation control conference (iaeac)* (Vol. 1, pp. 1435–1445).

Urmson, C., & Simmons, R. (2003). Approaches for heuristically biasing RRT growth. In *Proceedings 2003 ieee/rsj international conference on intelligent robots and*

systems (iros 2003)(cat. no. 03ch37453) (Vol. 2, pp. 1178–1183).

Voronoi, G. (1908). Nouvelles applications des paramètres continus à la théorie des formes quadratiques. Deuxième mémoire. Recherches sur les parallélloèdres primitifs. *Journal für die reine und angewandte Mathematik*, *134*, 198–287.

Wahba, G. (1965). A least squares estimate of satellite attitude. *SIAM review*, *7*(3), 409–409.

Wang, J., Chi, W., Li, C., Wang, C., & Meng, M. Q.-H. (2020). Neural rrt*: Learning-based optimal path planning. *IEEE Transactions on Automation Science and Engineering*, *17*(4), 1748–1758.

Wang, J., Chi, W., Shao, M., & Meng, M. Q.-H. (2019). Finding a High-Quality Initial Solution for the RRTs Algorithms in 2D Environments. *Robotica*, *37*(10), 1677–1694.

Wang, J., Li, C. X.-T., Chi, W., & Meng, M. Q.-H. (2018). Tropistic RRT*: An Efficient Planning Algorithm via Adaptive Restricted Sampling Space. In *2018 ieee international conference on information and automation (icia)* (pp. 1639–1646).

Warshall, S. (1962). A theorem on boolean matrices. *Journal of the ACM (JACM)*, *9*(1), 11–12.

Webb, D. J., & Van Den Berg, J. (2013). Kinodynamic RRT*: Asymptotically optimal motion planning for robots with linear dynamics. In *2013 ieee international conference on robotics and automation* (pp. 5054–5061).

Wilmarth, S. A., Amato, N. M., & Stiller, P. F. (1999). Maprm: A probabilistic roadmap planner with sampling on the medial axis of the free space. In *Proceedings 1999 ieee international conference on robotics and automation (cat. no. 99ch36288c)* (Vol. 2, pp. 1024–1031).

Wohlsen, M. (2014). Amazon reveals the robots at the heart of its epic cyber monday operation. *Last updated on*, *12*(01), 2014.

Xinyu, W., Xiaojuan, L., Yong, G., Jiadong, S., & Rui, W. (2019). Bidirectional Potential Guided RRT* for Motion Planning. *IEEE Access*, *7*, 95034–95045.

Yeh, H.-Y., Thomas, S., Eppstein, D., & Amato, N. M. (2012). UOBPRM: A uniformly distributed obstacle-based PRM. In *2012 ieee/rsj international conference on intelligent robots and systems* (pp. 2655–2662).

Yershova, A., Jaillet, L., Siméon, T., & LaValle, S. M. (2005). Dynamic-domain RRTs: Efficient exploration by controlling the sampling domain. In *Proceedings of the 2005 ieee international conference on robotics and automation* (pp. 3856–3861).

Yoon, J., & Crane, C. D. (2011). Path planning for Unmanned Ground Vehicle in urban parking area. In *2011 11th international conference on control, automation and systems* (pp. 887–892).

Zammit, C., & Van Kampen, E.-J. (2018). Comparison between A* and RRT algorithms for UAV path planning. In *2018 aiaa guidance, navigation, and control conference* (p. 1846).

Zucker, M., Ratliff, N., Dragan, A. D., Pivtoraiko, M., Klingensmith, M., Dellin, C. M., ... Srinivasa, S. S. (2013). Chomp: Covariant hamiltonian optimization for motion planning. *The International Journal of Robotics Research*, *32*(9-10), 1164–1193.