# MOBILE ROBOT NAVIGATION IN DYNAMIC ENVIRONMENTS

# USING AN IMPROVED RRT* APPROACH

by

Hussein Ali Mohammed

A Thesis Presented to the Faculty of the
American University of Sharjah
College of Engineering
in Partial Fulfillment
of the Requirements
for the Degree of

Master of Science in
Mechatronics Engineering

Sharjah, United Arab Emirates

November 2018

## Approval Signatures

We, the undersigned, approve the Master's Thesis of Hussein Ali Mohammed

Thesis Title: Mobile Robot Navigation in Dynamic Environments Using an Improved RRT* Approach

| **Signature** | **Date of Signature** (dd/mm/yyyy) |
|---|---|
| _____ | _____ |
| Dr. Lotfi Romdhane<br>Professor, Department of Mechanical Engineering<br>Thesis Advisor | |
| _____ | _____ |
| Dr. Mohammad Jaradat<br>Associate Professor, Department of Mechanical Engineering<br>Thesis Co-Advisor | |
| _____ | _____ |
| Dr. Shayok Mukhopadhyay<br>Assistant Professor, Department of Electrical Engineering<br>Thesis Committee Member | |
| _____ | _____ |
| Dr. Usman Tariq<br>Assistant Professor, Department of Electrical Engineering<br>Thesis Committee Member | |
| _____ | _____ |
| Dr. Lotfi Romdhane<br>Director, Mechatronics Engineering Graduate Program | |
| _____ | _____ |
| Dr. Ghaleb Husseini<br>Associate Dean for Graduate Affairs and Research<br>College of Engineering | |
| _____ | _____ |
| Dr. Richard Schoephoerster<br>Dean, College of Engineering | |
| _____ | _____ |
| Dr. Mohamed El-Tarhuni<br>Vice Provost for Graduate Studies | |

**Acknowledgement**

First and foremost, I acknowledge greatly the support of my family, especially my father Ali and my mother Rola, both morally and financially for this would have not been possible without it. I also would like to thank all my friends and colleagues in the Mechatronics department for their support, Engineers Danial, Ali, Mohamad, Amin, Omer, Usman, Koshish, Mirghani, Fares, Wasim, Affan, and last but not least Kent.

And finally, I feel the need to acknowledge my own efforts over the two years spent working on this thesis. It most certainly took a lot of time and effort and it was not easy and I managed to work against great odds to finish this work and bring it up to satisfy my own standards.

*This work is dedicated to my family...*

**Abstract**

A very prominent area in the field of Mechatronics is robot navigation and path planning. This area deals with the problem of autonomously calculating the least cost path in a provided environment, whether it is static or dynamic, and navigating the robot platform though this environment. Over the last decade, since the year 2001, there have been major breakthroughs in this field after LaValle introduced his revolutionary algorithm, the Rapidly-Exploring Random Tree (RRT) approach. Later, in the year 2011, Karaman introduced his novel modification to RRT which he called the Rapidly-Exploring Random Tree Star (RRT*). The main advantage of RRT* is its effectiveness and robustness in finding the path to the target and its probabilistic completeness property which guarantees the best theoretical path if given enough run time. However, RRT* still suffers from long processing times to provide paths with satisfactory quality in terms of cost and smoothness. Having said that, in this thesis we propose an improved version of the RRT* algorithm which will address the issue of long processing times and sub-par path quality. This new method is called Rapidly-Exploring Random Tree Star Normal (RRT*N). The presented method can handle static and dynamic obstacles in 2D and 3D environments. This improved method uses a Gaussian probability distribution to generate new nodes which have a higher probability of being generated along the vector pointing from the starting point to the goal point, which results in a tree centered on the line joining the robot to the target. It is shown that this method can be three times faster in finding the path to the target in static scenarios, and upto 20 times faster in dynamic environments compared to RRT*. Furthermore its rate of achieving satisfactory paths is consistently more than 95% while maintaining similar path quality. For instance in 250 trials of the presented static scenario, RRT*N had an average processing time and average path length of 38.46 seconds and 240.99 units, respectively. Meanwhile, RRT* resulted in 121.58 seconds and 259.11 units. This work is based on an extensive literature review to validate this work's novelty and the simulation and experimental results presented show the robustness of the proposed RRT*N method.

**Keywords:** ***RRT, RRT*, navigation, static, dynamic, path planning, optimal path, mobile robot.***

## Table of Contents

## List of Figures

## List of Tables

# Chapter 1: Introduction and Literature Review

In this chapter, a short introduction will be provided about the robot navigation problem in general and then about the particular problem discussed in this thesis. A discussion of the thesis objectives and the contributions of this research will also be provided. Finally, the organization of this report will be provided.

## 1.1. Overview

In recent years there have been great advancements in both wireless communications as well as computing power, which allowed for great improvements in the autonomous robot navigation field. Indeed, more complex mathematical operations and large quantities of data are processed in significantly shorter periods of time. In the last few years, research started heading towards optimizing the available algorithms for either reduced computational cost, in the form of processing time and memory consumption, or for adaptability to non-holonomic dynamics and by extension providing smoother and lower cost paths. The latter is done through parallel modification being incorporated in the navigation algorithms. However, it is still rare to see this being done to 3D navigation. Therefore, despite robot navigation being a popular research topic, there is still plenty of room for implementing and optimizing the available algorithms in dynamic situations, which will be one of the main points addressed in this research.

## 1.2. Thesis Objectives

The aim of this research is to present a comprehensive review of the available approaches to the path planning problem. Based on this literature review, an improved algorithm of the RRT*, called RRT*N is proposed to significantly reduce the processing time while producing paths of similar cost. This reduction will open the doors for having much shorter navigation times, due to the elimination of initial offline iterations of the algorithm. The needed path is rather generated on the spot, and in turn allows for a great flexibility in the implementation for dynamic scenarios. Further editions of the algorithm will be presented to demonstrate the possibilities of what can be created as alternatives. Further, the resultant paths will be smoothed with a modified moving

average method and a pure pursuit method. The obtained smooth paths are then implemented on the Kobuki platform. The resulting modified algorithm is to be extensively tested in simulation and experimentation to support our claims.

## 1.3. Research Contribution

This work will present a novel and significant modifications to a well-known optimizing algorithm utilized for path planning called Rapidly-Exploring Random Tree Star (RRT*). The modification is comprised of altering the probability distribution with which the growth nodes are generated resulting in a controlled or biased growth pattern. This new growth pattern is based on the normal or Gaussian probability distribution hence the new algorithm is named RRT*N. This makes it unlikely for the tree to grow towards random useless regions that could result in higher path costs, this nets paths that are of more consistent quality which are of similar or better quality to those produced by the classic RRT*. Finally, a sub-algorithm adapted for dynamic situations is also developed alongside a smoothing algorithm to result in a greatly improved version of the classic algorithm both in terms of processing time and path quality, where path quality is measured as the percentage difference from the ideal mathematical path; the lower the percentage difference the better. It is worth mentioning that the ideology behind this work was influenced by the heuristics utilized by the A* algorithm.

## 1.4. Thesis Organization

The remainder of this thesis report is organized as follows: The rest of this chapter will present the problem of robot navigation and how it is broken down into it's main three components. It will then go into the details of the path-planning aspect of this problem and present a comprehensive literature review of the available approaches as it delves into the Rapidly-Exploring Random Tree family of approaches. Chapter 2 will then provide a background on the relevant robot and controller models as well as the details and the specifics of the workings of RRT*. This algorithm is then modified, yielding an improved algorithm, which will be the contribution of this thesis. In Chapter 3, all the obtained results in 2D and 3D and the implementations in static and dynamic

scenarios, will be presented. Finally, the conclusion will sum up the work done and provide a segue into the suggested future work.

## 1.5. Navigation Problem

The meaning of navigation is having all the necessary information to be able to go from one point to another; typically, from a starting point to a goal point. However, more generally, it is the ability to move within an environment safely, while avoiding obstacles or undesirable conditions. Three main things are needed for a successful navigation: knowing the robot's position, being able to plan the robot's path, and to construct a map and understand it [1]. Moreover, a dynamic environment is one where the goal, some or all the obstacles, or both are moving. This is a much more challenging problem compared to a static environment navigation due to the much higher number of variables. Typically, static environment navigation algorithms are modified to be used for dynamic environments. The most interesting one is the predictive algorithm, which marks an object as dynamic if its velocity is above a certain threshold, and it will then attempt to predict how the object moves and instructs the robot to act accordingly [2].

**1.5.1. Mapping and localization.** The maps constructed by the robot utilizing its sensor of choice are stored and are often being streamed back to the user via a wireless communication. This is of great value as the user gets to have real-time view of the robot's performance and this is a vast research field. Indeed, this is a valuable information in the robot navigation problem and the first input used for the generation of efficient and safe paths. In our case the main environmental map will always be provided, namely all large and static known obstacles will already be specified in the map before the path planning procedure starts. However, the detection of dynamic obstacles will be done via the Kinect sensor. Localization is the knowledge of the robot's current location and orientation with respect to its starting position and its environment. This is an equally important information because it is what gives value to the map information by knowing the robot's surrounding at all times as an inference from its position in the provided map. In our case this will be done via Quarc software which utilizes the en-

coder readings filtered through a Kalman filter since the setup is indoors. However, the GPS system can be utilized for outdoor navigation.

**1.5.2. Path-planning.** Path planning is the procedure of computing the trajectory that the robot will follow during its navigation process. Typically, the paths are heuristic meaning they are methodologically generated according to a certain set of rules, or are sampling based meaning that the they are gradually constructed segment by segment as the algorithm process time goes on. Some of these algorithms are single-query meaning that they need the input information once to construct the path while others are multi-query requiring in- out information multiple times. However, there exist other algorithms which operate based on different principles. The path-planning process can be done 'offline' while the robot is moving, while others are capable of such feats. The available approaches and their details will be discussed in the next sections.

**1.5.3. Obstacle avoidance.** Once the information is gathered by the robot and the current and target locations are specified, the algorithm generates a path. Based on the generated path, the robot navigates through its environment. Whenever undesirable situations arise during this navigation process (for instance a potential collision), another protocol comes into play, i.e., obstacle avoidance. This protocol allows the robot to either modify its path to avoid the obstacle or to decide whether it is possible to wait for the obstacle to move away from the path.

## 1.6. Review of Available Approaches

This section will present the available approaches, which are utilized in the path-planning component of the robot navigation problem. We will be going into the details of these approaches and comparing them by presenting their strengths and weaknesses. As a result of carefully studying several survey works such as [3], [4], [5], and [6], the approaches for path planning can be grouped into four main categories: Grid-based algorithms, Geometric algorithms, Artificial potential field algorithms, and Sampling-based algorithms. Brief summaries of these four main categories are found in the following subsections.

**1.6.1. Grid-based algorithms.** Grid based approaches overlay a grid of specified grid size on top of the configuration space and now each grid point is associated with a coordinate that, in turn, represents a specific configuration. Then the motion across the adjacent grid points is checked through collision detection confirming that it takes place within $Z_{free}$. The most popular algorithms from this type are A*, introduced by Hart in [7] and discussed by Zeng in [8], and D*, introduced by Stentz in [9] and discussed by Koenig in [10]. This type of algorithm is usually fast in finding a path when the grid size is large but they fail quickly when they are expected to go through narrow passages. It is also to be noted that such algorithms are not appropriate for higher dimension scenarios because the planner will consume a huge amount of memory and will take a relatively long time to map the full configuration space.

**1.6.2. Geometric algorithms.** Geometric algorithms are ones that use pure geometric properties of the environment along with mathematical methods to generate a path connecting the starting point to the goal point. These methods are generally basic and do not provide any mathematical guarantees to obtain a path quality or a probabilistic completeness. The most popular algorithms of this type are the bug family of algorithms, discussed by Ng in [11] and by Zhu [12], and the visibility graph method discussed by Asano in [13]. These algorithms also belong to the heuristic/deterministic family of algorithms like the grid-based methods meaning they quickly fail when used in higher dimensions.

**1.6.3. Artificial potential field (PF) algorithms.** The potential field approach is where the robot's configuration is treated as a point in a potential field where it is attracted to the goal and repulsed from obstacles, it is very similar in principle to the concept of electrostatic potential fields. The obtained trajectory is the path. The advantage of this approach is that the trajectory is produced with little computation but it is very prone to local minima scenarios and often fails to find a path. Also, they are heuristic, meaning by extension they do not guarantee an optimum path. PF algorithm was introduced by Warren in [14] and discussed by Vadakkepat in [15]

**1.6.4. Sampling-based algorithms.** Sampling-based approaches operate through sampling the environment, as their name implies, through point generation. These points are then utilized to construct segments that, with time, fill the free portion of the configuration space. The obtained paths can be of high quality if they are given enough nodes and time to execute. The most popular sampling-based algorithms are Probabilistic Roadmap (PRM), introduced by Kavraki in [16] and discussed by Geraerts in [17], and Rapidly-Exploring Random Trees (RRT), introduced by LaValle in [18] and discussed along with its improvements for the vast majority by the remainder of the references in this work. Sampling-based approaches are probabilistic in nature meaning that with improvements they can guarantee optimal paths in acceptable processing times.

## 1.7. Rapidly-Exploring Random Trees (RRT)

The Rapidly-Exploring Random Tree algorithm has been gaining significant popularity in the recent years, especially in the field of path and motion planning due to its convergence property. Indeed, the RRT will always find a suitable path, provided it is allowed to run for a long enough time with a sufficient number of nodes. It also provides capabilities that allow for incorporation of holonomic constrains allowing for various real life implementations with articulated robots and rigid bodies. However, it is not probabilistically complete meaning that it does not guarantee the optimum path. Further, the generated path is often irregular unless a large number of nodes is provided. This transitions us towards the more advanced version of this algorithm, called RRT*.

## 1.8. Rapidly-Exploring Random Trees Star (RRT*)

The main goal of navigation and path planning is to find an obstacle free path connecting the starting point and the goal point with a continuous path that has the minimum cost, often being the shortest path. Other criteria for selecting the path could be the smoothness and the dynamic property. Recently, research has been gravitating around RRT*, which is a sampling-based single-query path planning algorithm introduced by Karaman and Frazzoli [19]. The main reasons behind the popularity of this method are its probabilistic completeness, flexibility in application and implementa-

tion, and its low computational cost, when operated in higher dimensions relative to grid-based algorithm [19].

The authors of [19] made a significant improvement compared to the work done by LaValle on the RRT in [18], in terms of path quality and convergence to the optimum path. Indeed, the RRT* algorithm can reconnect nodes to optimize the path cost leading to specific nodes resulting in a path, which is the closest to the best theoretical path. Among all the popular and commonly used and studied sampling-based single-query path planning algorithms, it is found that RRT* is one of the most flexible approach for dynamic environments. It is also suitable for robots with models imposing non-holonomic constraints, as discussed in [20] and [21]. Therefore, following in the footsteps of the work done by Karaman and Frazzoli in [19], several researchers worked to improve the RRT algorithm. Some improvements addressed the lengthy processing times and the issues of a large memory consumption. Meanwhile, other works incorporated dynamic constraints of non-holonomic systems to adapt the algorithm for certain applications.

## 1.9. Improvements on RRT*

Among the available RRT* approaches, the holonomic ones are those where the improvements aim to improve or optimize the algorithm itself. These approaches are trying to intrinsically enhance the process without changing its holonomic constraint property. These approaches usually try to reduce the processing time or memory usage. They sometimes aim at improving the path quality as well.

For example, a modification of the RRT* called Anytime RRT* was discussed in [22] by Karaman et al., which is an online version of RRT*. The proposed algorithm runs only for a fixed amount of time, which is chosen by the user. The goal of this approach is to handle the large computation time usually faced when using the RRT*, especially when looking for high-quality paths. Anytime RRT* provides a quick but unoptimized initial path. The remainder of the permitted time is then used to optimize the generated path.

Likewise, a memory efficient version of RRT* is presented by Adiyatov and Varol in [23] and is called RRT* Fixed Node (RRT*FN). In this version, the algorithm

limits the number of nodes that the tree can contain. This is not to be confused with the number of nodes, which can be generated in the expansion process, but it is in fact the number of nodes constituting the tree itself. This critical piece of information translates to, once this limit is reached, a limitation of the number of nodes as the algorithm can only add a new node to the tree by deleting a previously existing node. The algorithm selects the node to be deleted based on the improvement of the cost, when using the new node in the tree. This procedural check is continuously done on both the local and the global scales, which, at the cost of a reduced path quality, results in considerable saving in memory consumption. Others have taken a different approach to this problem, for instance Gong et. al. They explore in [24] how incorporating the information of the goal node in their modified algorithm, called Target-Oriented RRT*, greatly improves consumed on time and also produces better chances of successfully finding a path to the goal point. The paths they produce are then smoothed using a B-spline.

In contrast, a version of RRT* called Informed RRT*, which focuses on an improved path quality, was presented by Gammell et al. in [25]. As suggested by its name, this version of RRT* is provided with pieces of information that are typically either not known or useful to the classic algorithm; specifically, the path generated from the immediate previous iteration. In other words, a standard RRT* algorithm is executed first to generate an initial path. The generated path is then used as the backbone around which an ellipsoid is formed. This ellipsoid marks the area in which the algorithm will run again. This process is done repeatedly resulting in increasingly smaller ellipsoids. This action results in a path with significantly higher quality at the cost of higher computational time. Also, Qureshi et. al. in [26] improved upon Bidirectional RRT* (BRRT*) to reach even better performances in terms of processing times.

For the case of non-holonomic RRT* approaches, the non-holonomic constraint requires appropriate changes to the path to account for the dynamics of the robot. Examples of such mechanisms are fixed-wing UAVs and car-like robots due to how such robots perform complex motions. The work presented by Webb and Berg in [27] and [28] proposes Adapted RRT*, which is an improved version of the algorithm considering the kinodynamic constraints. Their algorithm proved to be capable of provid-

ing a 10-dimensional state space aerial vehicle and a 5-dimensional state space car-like robot with near-optimum trajectories based on the produced paths.

Another version of the RRT* called Spline RRT* (SRRT*) incorporates splines to significantly improve on the resulting path quality [29]. The authors of this work demonstrated the efficiency of their method in producing 3D trajectories to accommodate for fixed-wing aerial vehicles. Similarly, the work done by Abbadi et al. in [30] also relates to the work in [29], where the aim is to modify the RRT* to produce significantly improved paths, which are realistically applicable to real dynamic situations. Cobano et al. in [31] have demonstrated similar work where they created a version of RRT*, which is optimized for UAVs. This version is called RRT*i, where an initial path is generated by a classical RRT* approach, then a Gaussian distribution and a node rejection technique are utilized to refine this path allowing for more feasible trajectories. Dong et al. presented in [32] their work on adapting RRT* for leader-follower drone applications. The method they proposed is Closed-Loop RRT* (CL-RRT*), where the goal point of their tree is continually changing namely being the leader drone. This is done by running the algorithm with the aid of an LQR-based bottom-level controller. Other work that focused on 3D planning is discussed by Pharpatara et. al. in [33] where they incorporate Potential Field algorithm into RRT* to achieve better convergence to the optimal solution.

Since this work is focusing on dynamic situations, works related to that field will also be discussed. Fulgenzi et. al. in [34] have presented their work in incorporating Gaussian predictive processes that aid in understanding and pre-learning the movements of dynamic objects, hence allowing for successful navigation in dynamic environments. Also, a survey paper on navigation in dynamic environments was considered as reference in this research [35]. Several works further enhanced the work in [19] either to optimize it for processing time, or to make it more suited for operational in 3D like the work done in this research. The aim of this work is to present an improvement on the 3D version of RRT* algorithm in terms of computational time while maintaining the path quality.

Tables 1 and 2 will summarize some of the previously discussed variants of RRT as well as several others by stating their constraints, modes, models, contributions and limitations. The numbers in Tables 1 and 2 refer to the same variant.

Table 1: RRT* variants summary

| # | Variant | Constraints | Mode | Model |
|---|---|---|---|---|
| 1 | RRT* [19] | Holonomic | Offline | Point |
| 2 | Anytime RRT* [22] | Non-holonomic | Offline | Car-like |
| 3 | B-RRT* [36] | Holonomic | Offline | Rigid Body |
| 4 | RRT*FN [23] | Holonomic | Offline | Robotic Arm |
| 5 | RRT*-Smart [37] | Holonomic | Offline | Point |
| 6 | Optimal B-RRT* [38] | Holonomic | Offline | Point |
| 7 | RRT# (RRT ”sharp”) [39] | Holonomic | Offline | Point |
| 8 | Adapted RRT* [28] | Non-holonomic | Offline | Car-like and UAV |
| 9 | SRRT* [29] | Non-holonomic | Offline | UAV |
| 10 | Informed RRT* [25] | Holonomic | Offline | Point |
| 11 | IB-RRT* [26] | Holonomic | Offline | Point |
| 12 | DT-RRT [40] | Non-holonomic | Offline | Car-like |
| 13 | RRT*i [41] | Non-holonomic | Online | UAV |
| 14 | RTR+CS* [42] | Non-holonomic | Offline | Car-like |
| 15 | Mitsubishi RRT* [43] | Non-holonomic | Online | Car-like |
| 16 | CARRT* [44] | Non-holonomic | Online | Humanoid |
| 17 | PRRT* [45] | Non-holonomic | Offline | Car-like |
| 18 | HARRT* [46] | Holonomic | Offline | Point |
| 19 | T-RRT* [47] | Non-holonomic | Offline | UAV |
| 20 | RABIT* [48] | Non-holonomic | Offline | UAV |

Table 2: RRT* variants details

| # | Research Contributions | Limitations |
|---|---|---|
| 1 | Proved asymptotically optimal property for RRT* and introduced near neighbor search and rewiring operations resulting in visibly refined path quality than original RRT. | Tree re-wiring improved tree cost but it also slowed down convergence rate to the optimum path and also has large memory requirements. |
| 2 | Introduced committed trajectories and branch-and-bound adaptation which is a mathematical optimization technique that improved trajectory and computational efficiency by gradually removing nodes from tree that cannot improve the path quality. | Susceptible to overestimating rejection criteria and cause unnecessary node removal during initial expansion of tree when it is not mature resulting in jagged and sub-optimal paths. |

Continued on next page

Continued from previous page

| | | |
|---|---|---|
| 3 | Sample rejection with an admissible heuristic resulted in an improved convergence speed and a path refinement. | It being a bi-directional variant of RRT*, it attempts to connect both trees in each iteration resulted in increased computational requirement, large memory requirements, and jagged and sub-optimal paths. |
| 4 | Forcing a fixed number of nodes in the tree resulted in a memory efficient version of RRT*. | Works only for static known environment and the ate of convergence to optimal path is lower than base RRT*. Also has jagged and sub-optimal paths. |
| 5 | Introduced intelligent sampling and path optimization features that accelerated the convergence rate with improved efficiency with respect to both time and cost. | Dependent upon a heuristic called Biasing Ratio, which is automated and require programmer dependent value for different environments and also has a trade-off between convergence rate and exploration of space. |
| 6 | Increase convergence rate of bidirectional RRT* by introducing multiple heuristics based on different conditions. | Using multiple biased heuristics caused computational overload and they also interfere with the algorithm characteristics and limit its application. |
| 7 | Introduced a global re-planning scheme that keeps promising nodes of the tree to make fast convergence towards optimal path with low cost. | Needs to have real time experiments and applications would be to investigate and improve the approach's efficiency. |
| 8 | Presents a kinodynamic extension of RRT* that ensures asymptotic optimality with controllable linear dynamics in multi-dimension state space. | Still need to present path smoothing steps and real world quad-rotors experiments which would require controller stabilization for final trajectory. |
| 9 | Proposed a spline-based RRT* version using cubic Bezier curve for fixed-wing UAVs and a collision and dynamic feasibility function that checks for constraints during tree expansion resulting in feasible, smooth, and cost-optimal path in 3D simulations. | Need to implement an online planner with real-time application to further investigate and improve the efficiency of the approach. |
| 10 | Proposed a direct sampling technique using ellipsoidal informed subset, which showed improved convergence relative to RRT*. | The heuristic used to shrink the planning problem is highly dependent upon the initial solution cost, making it effective only under certain conditions. |
| 11 | Introduced an intelligent sample insertion heuristic with minimal memory requirements that improved the path quality compared to RRT* and B-RRT*. | Its needs to be further investigated for application in online planning. |

Continued on next page

Continued from previous page

| | | |
|---|---|---|
| 12 | Enabled high speed mobile robot planning by producing practically feasible trajectories that was done by introducing a reconnect-tree scheme which is a modification that maintains child nodes with reduced computational cost instead of using rewire operations, and also reduced the node rejection chances by using kinodynamic conditions. | The reconnect-tree scheme is only beneficial in known environments meaning that the approach needs advancement for dynamic and unknown scenarios. |
| 13 | Produced smoother trajectories as compared to RRT and RRT* with improved path quality using a new local sampling technique hence making them much more practical trajectories for UAVs. | There is a risk of producing unexpected collisions in applications with multi-UAV with increased uncertainty. |
| 14 | Generated feasible paths for car-like robots using local planner that uses circular arcs and straight segments. | Generated paths are not natural and curvature continuous, it is also still in the process of improving computational performance. |
| 15 | Introduced a local re-planning procedure to safely avoid and re-plan due to dynamic obstacles allowing for continuous curvature smooth path for autonomous driving vehicle with the capability of lane management on roads. | Uncertainty of dynamic environment is not managed in the context of autonomous vehicle driving. |
| 16 | Used two trees and cross entropy in RRT* to address the problem of large number of samples, this results in an energy efficient path in high dimensional space. | This approach is limited to address the planning problems of humanoid robots. |
| 17 | Incorporated artificial potential field characteristic in RRT* to address the problem of high memory consumption and slow convergence. | The approach has not been tested or implemented in online planning. |
| 18 | Used RRT* and a homotopic algorithm, presented a human-robot interactive planner. | There is a trade-off between computational efficiency and path quality and the graphical user interface is not very user friendly. |
| 19 | Presented efficient extension of tree in a cost space by integrating transition tests with RRT*. | Needs to have a performance analysis with RRT* in different scenarios to show its best field of implementation. |
| 20 | Combined RRT* with an informed global technique called BIT* to find an optimal path for narrow passages in high dimensions. | Need to further investigate the local optimizer suggested by approach for optimization of path. |

# Chapter 2: Methodology

In this chapter we will be presenting the algorithms, models, and equations, which are needed to provide a clear understanding of this thesis's direction. This chapter focuses on the RRT* method and discusses and presents a step by step explanation of how we arrived at the modification resulting in the RRT*N algorithm, which is this thesis's main contribution.

## 2.1. Background

This section starts by presenting the RRT* algorithm and the required modifications for handling 3D cases. Also, it will contain the dynamic models and the controller designs of the robots to be used for the simulations and the implementation in the 2D case. This will collectively provide a background preview of all the relevant details and information what will then be utilized in constructing our modified approach and testing it.

**2.1.1. Classic RRT* approach details.** In all the available variations of the Rapidly-Exploring Random Tree algorithm, the first and most important variable that needs to be known is the configuration space, $Z$. The set of all the possible transformations that can be applied to the robot is defined as its configuration space. This can also be linked to the dimension of the space in which the robot operates [49]. The occupied region in the configuration space, whether by obstacles or simply an inaccessible or undesirable region is denoted $Z_{obs}$. That region is invalid for navigation hence often when running the algorithm, researchers might set it up such that no nodes will be generated in that region. Although, it does not actually matter since collision checks will be performed to make sure that no branch of the tree enters the region $Z_{obs}$. Meanwhile, the remaining elements in the configuration space, which constitute the free or navigationable space, $Z_{free}$, is the area in which the tree can grow. Then within $Z_{free}$ the starting point, $Z_{init}$, and the goal point, $Z_{goal}$, are defined.

RRT* will construct a tree-like structure, T, that searches for the specified goal point in a pre-known environment. It is worth mentioning that when seen in 3D, the appearance of the produced shape would immediately justify the algorithm's name.

This tree structure will originate from $Z_{init}$ and expands randomly until it finds $Z_{goal}$. The shortest, least cost path is then selected from the tree from the segments available. The algorithm then starts the exploration process by generating a random node within $Z_{free}$, called $Z_{rand}$.

The algorithm then searches for the nearest node in the tree, $Z_{nearest}$. If the distance between $Z_{rand}$ and $Z_{near}$ is within the specified step size, a branch will be created to connect them in the form of a new segment. This is referred to as the expansion or growth process. It is always checked whether the segment is valid by geometrically confirming that it does not result in a collision. However, if the distance between them is larger than the specified step size, the algorithm will generate a new node, which will be exactly one step size away from $Z_{nearest}$ in the direction of $Z_{rand}$. The new node that will be generated and added to the tree as a result is called $Z_{new}$. Next, the algorithm performs a collision check to make sure that $Z_{new}$ is safe. If all the checks indicate that the new node is feasible, the two will be connected. Figure 1 shows the expansion process. Further explanation found in [50] and [51].

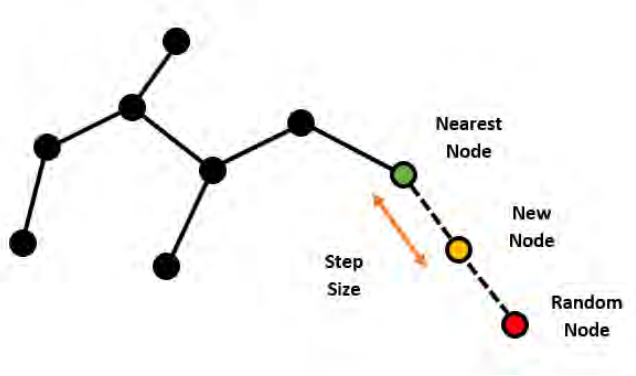


Figure 1: Tree Expansion Process

So far, the procedure explained is the basic working of the classic RRT, there is one more step to for RRT*. The extra step checks the newly added node to the tree to see whether it is possible to reconnect the most recent node to a different parent node such that it would have a lower cost. The parameter used in this rewiring process is defined as the neighborhood radius, $R_n$. The neighborhood radius translates to a circle or a sphere centered at $Z_{new}$, where all the nodes within this region are labeled $Z_{near}$ and

are checked for being a possible lower-cost parents for $Z_{new}$. If such a node is found, it will be labeled $Z_{min}$, and the last connection will be changed such that $Z_{new}$'s parent node is now $Z_{min}$ rather than $Z_{nearest}$. The relation used to find an appropriate initial value for the neighborhood radius is given by the equation below:

$$R_n = \gamma\left(\frac{\log n}{n}\right)^{1/d} \tag{1}$$

Here, $\gamma$ is an appropriate planning constant chosen by the user, n is the dimension of the space (2D or 3D for robot path planning), d is the dimension of the configuration space. It is to be noted that the neighborhood radius is ideally two to three times the magnitude of the step size so that the tree branches can be rewired effectively. The neighborhood radius is 2.5 times the step size in all tests demonstrated in this research. The process described above can be seen in Figures 2 and 3.

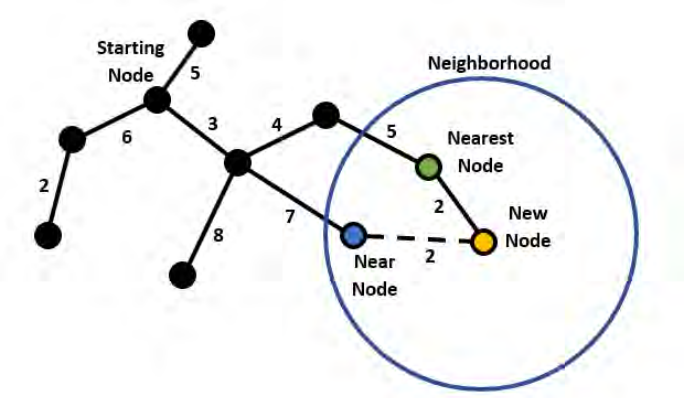


Figure 2: Neighborhood radius and parent node cost check

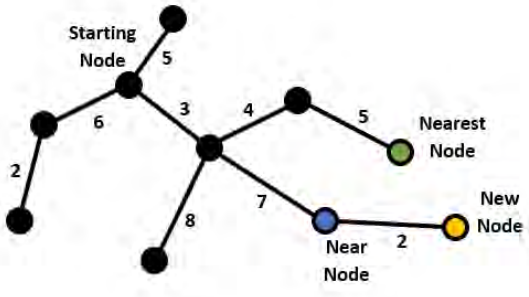


Figure 3: Node reconnected to lower cost parent

As seen in Figures 2 and 3, the neighborhood around the new node ($Z_{new}$) was created and the check for lower cost parent was made. The original parent node, nearest node ($Z_{nearest}$), would result in new node ($Z_{new}$) having a total cost of 14. The check revealed that if it is instead connected to near node ($Z_{near}$) it would have a lower cost of 12. This resulted in the re-connection seen in Figure 3. This process is done whenever a new node is added to the tree to insure the optimum least cost path.

The expansion process goes on as described until the specified number of nodes is used up or the goal region is reached. The process is also explained in more detail in [19]. It is extremely rare for the tree to reach the goal point exactly, hence the introduction of the goal region. This region can be a 2D area or a 3D volume of any shape, for example a square/cube or a circle/sphere with $Z_{goal}$ at its center.

In this work, the goal region is a sphere with a center $Z_{goal}$ and a radius "Vision". The parameter Vison refers to a robot's range of vision. A pseudocode summarizing the process of the classic RRT* is found in Algorithm 1 also a typical trial in a 100x100 free environment is shown in Figure 4.

**Algorithm 1** RRT* Algorithm Pseudocode

```
T initialized through InitializeTree()
T has initial node added by InsertNode(Z_init)
for i = 0:N do
    Z_rand obtained through Sample(i)
    Z_nearest obtained through Nearest(T, Z_rand)
    Z_new obtained through Steer(Z_nearest, Z_rand)
    if ObstacleFree(Z_new) then
        Z_near obtained through Near(T, R_n, Z_new)
        Z_min obtained through ChooseParent(Z_near, Z_nearest, Z_new)
        T updated by InsertNode(T, Z_min, Z_new)
        T rewired by Rewire(T, Z_near, Z_min, Z_new)
    end if
end for
Path is the least cost path extracted from T
```

Algorithm 1 shows in brief the flow of the functions utilized to generate and grow the Rapidly-Exploring Random tree star. This is a standardized manner of presenting this algorithm but the functions are based on the user's implementations.

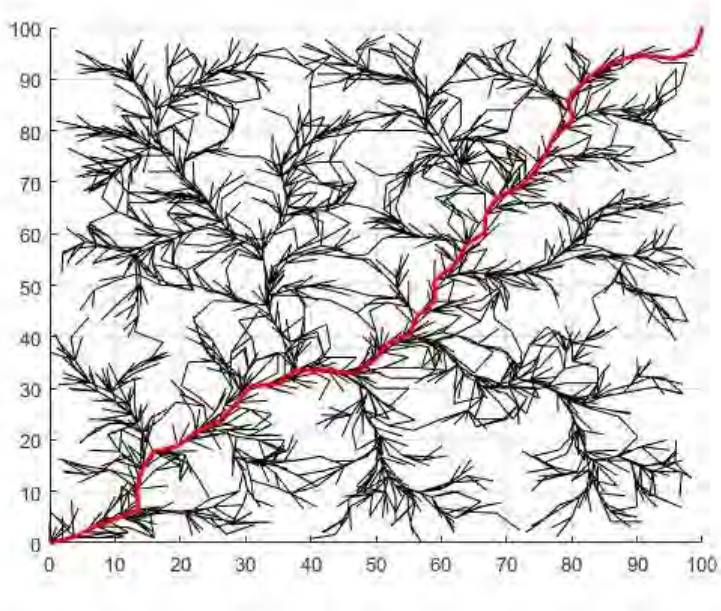


Figure 4: RRT* typical 2D trial in free environment

**2.1.2. Mobile robot model and controller.** The derivation of the mobile robot's kinematics is based on the assumption that it is made up of a rigid frame and rigid wheels and that it is moving on a horizontal plane. The robot frames and parameters are shown in Figure 5 and Table 3.

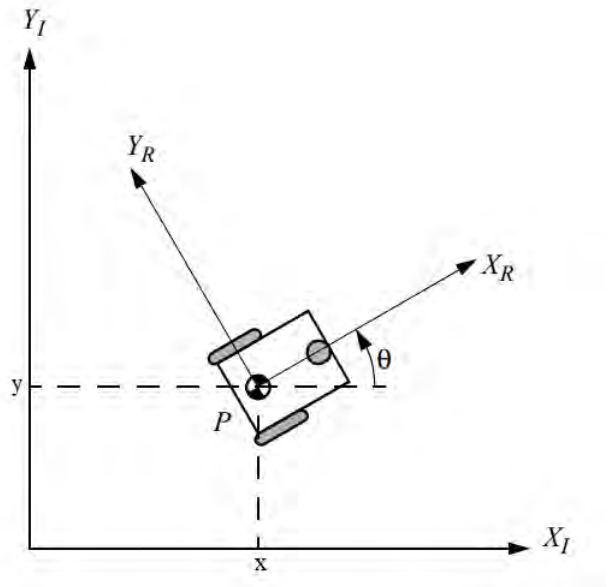


Figure 5: Global and local reference frames [52]

Further background information is provided in [52]. We establish a relationship between the robot's inertial frame (local frame) and the global environmental frame to specify the position of the robot. The global inertial frame axes are $X_I$ and $Y_I$ and the robot's local frame axes are $X_R$ and $Y_R$.

Table 3: Robot model parameter descriptions

| Parameter | Description |
|---|---|
| P | Robot local pose vector |
| G | Robot global pose vector |
| x | x-position of the robot with respect to the inertial frame |
| y | y-position of the robot with respect to the inertial frame |
| r | Robot wheel radius |
| $\theta$ | Orientation of the robot with respect to the inertial frame |
| $\phi_R$ | Robot right wheel angle |
| $\phi_L$ | Robot left wheel angle |
| $v_R$ | Robot right wheel speed |
| $v_L$ | Robot left wheel speed |
| $\omega_R$ | Robot right wheel rotational speed |
| $\omega_L$ | Robot left wheel rotational speed |
| l | Distance between each wheel and center of robot |
| R | Orthogonal rotation matrix |

The robot that will be used in the tests is shown in Figure 6.

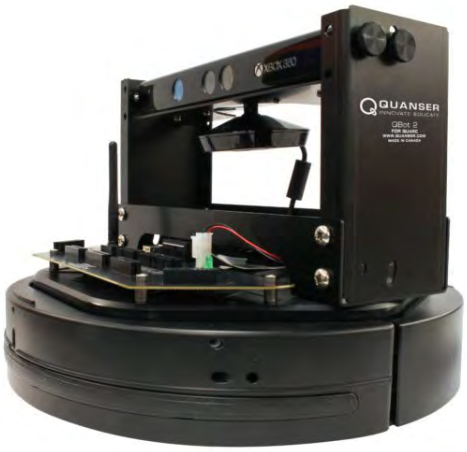


Figure 6: Mobile robot used for the model parameters [53]

And the values that will be used in the 2D simulation and implementation are in Table 4.

Table 4: Robot model parameter values

| **Parameter** | **Value** |
|---|---|
| r | 0.06 |
| L | 0.15 |

First, we define the pose vector as follows:

$$G = \begin{pmatrix} x \\ y \\ \theta \end{pmatrix} \tag{2}$$

Then we define the rotation matrix as follows:

$$R = \begin{pmatrix} c\theta & s\theta & 0 \\ -s\theta & c\theta & 0 \\ 0 & 0 & 1 \end{pmatrix} \tag{3}$$

We relate the robot's local pose to the global pose:

$$\dot{P} = R\dot{G} \tag{4}$$

We find the speed of the robot's right and left wheels:

$$v_R = \frac{r\dot{\phi}_R}{2}, \quad v_L = \frac{r\dot{\phi}_L}{2} \tag{5}$$

We find the rotational speed of the robot's right and left wheels:

$$\omega_R = \frac{r\dot{\phi}_R}{2l}, \quad \omega_L = -\frac{r\dot{\phi}_L}{2l} \tag{6}$$

Finally, we find the robot's velocity vector in the global frame:

$$\dot{G} = R^{-1} \begin{pmatrix} v_R + v_L \\ 0 \\ \omega_R + \omega_L \end{pmatrix} \tag{7}$$

Having the robot pose through integrating the $\dot{G}$ vector allows us to perform position control on the robot. Also, the vector itself allows us to record the robot's linear and rotational speeds. We utilized a basic PI controller method for our case where we aim to maintain a constant velocity while varying the rotational speed in accordance with the provided waypoints, the details of controller used are obtained from [53] and are discussed in Table 5.

Table 5: Robot controller parameter descriptions

| **Parameter** | **Description** |
|---|---|
| v | Robot speed |
| $\delta$ | Error vector |
| $e_v$ | Error of speed |
| $e_\theta$ | Error of orientation |
| $K_p$ | Proportional gain |
| $K_i$ | Integrator gain |

And the gain values that will be used in the 3D simulation are shown in Table 6.

Table 6: Robot controller parameter values

| **Parameter** | **Value** |
|---|---|
| $K_p$ | 10 |
| $K_i$ | 0.5 |

The error vector is defined as follows:

$$\delta = \begin{pmatrix} e_v \\ e_\theta \end{pmatrix} \tag{8}$$

We compute the desired speed:

$$v = K_p \int_0^T e_v + K_i \int_0^T \int_0^T \dot{e_v} dt \tag{9}$$

We compute the desired rotational speed:

$$\dot{\theta} = K_p e_\theta + K_i \int_0^T \dot{e_\theta} dt \tag{10}$$

The above robot model and controller are constructed in MATLAB for the simulation part and then they were made in SIMULINK utilizing Quarc blocks to communicate the information to the Qbot2 for the implementation part.

**2.1.3. Drone model and controller.** The outlines used in [54] were useful in clarifying several aspects of the way and the order the different tasks have to be made. However, the model used in [54] was not clear, hence [55] was used for that as a reference. Figure 7 summarizes how the block diagram of the Quadrotor's controller needs to be structured. This only depicts the control in the x-direction and the pitch angle but the same is done for the other parameters. It shows how the inner loop is the angle control or the stabilizer and that is the faster loop. The outer loop controls the position and is the slower one. An accurate dynamic model of the system is needed, which was the objective of the work presented in [55].

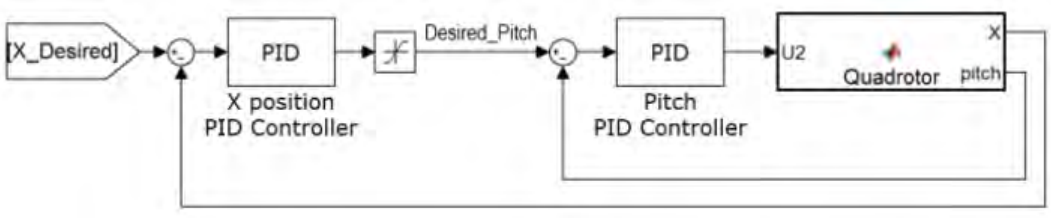


Figure 7: Quadrotor control loop for x-direction and pitch [55]

As mentioned before, the work in [55] was greatly helpful in guiding us in constructing this model. To begin the modeling, we first define our working frames:

the inertial frame is the ground having the gravity pointing in the negative z-direction while the body frame is the drone itself, which is defined by its orientation and has the axes of the propellers pointing in the positive z-direction and its arms pointing in the x and y-directions. Tables 7 and 8 provide all the parameters used in the system model derivation and the values used.

Table 7: Drone model parameter descriptions

| **Parameter** | **Description** |
|---|---|
| P | Drone position vector |
| x | x-position of the drone with respect to the inertial frame |
| y | y-position of the drone with respect to the inertial frame |
| z | z-position of the drone with respect to the inertial frame |
| A | Drone orientation vector |
| $\phi$ | Roll angle of the drone |
| $\theta$ | Pitch angle of the drone |
| $\psi$ | Yaw angle of the drone |
| $\omega$ | Angular velocity vector |
| R | Rotation matrix relating body frame to inertial frame |
| $T_i$ | Individual propeller thrust force |
| $u_i$ | PWM input signal to the individual propeller |
| K | Thrust function relating propeller thrust force to PWM input |
| $T_B$ | Total body thrust force vector |
| $F_d$ | Drag force vector |
| $k_d$ | Coefficient of drag |
| $\tau_B$ | Torque vector |
| L | Distance between center of drone and center of propeller |
| b | Coefficient of proportionality between PWM input and propeller torque |
| m | Drone's mass |
| G | Gravitational acceleration vector |
| g | Gravitational acceleration |
| I | Inertia matrix |
| $I_{xx}$ | Drone's moment of inertia about its x-axis |
| $I_{yy}$ | Drone's moment of inertia about its y-axis |
| $I_{zz}$ | Drone's moment of inertia about its z-axis |

Figure 8 demonstrates the drone's local or body frame as well as the global or inertial frame discussed previously. These frames must be correctly defined and understood properly to successfully relate the location, orientation, and motion of the drone to the ground station.

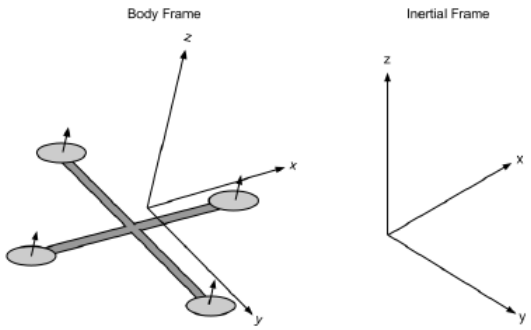


Figure 8: Body and Inertial frames of the drone [55]

And the values that will be used in the 3D simulation are in Table 8.

Table 8: Drone model parameter values

| **Parameter** | **Value** |
|---|---|
| m | 1.79 |
| b | 4 |
| $k_d$ | 25 |
| L | 0.2 |
| $I_{xx}$ | 0.03 |
| $I_{yy}$ | 0.03 |
| $I_{zz}$ | 0.04 |

The values presented in Table 8 are particular to the Quanser Qball2 drone. The drone is shown in Figure 9.

Figure 9: Drone used for the model parameters [53]

First, we define the position and orientation vectors as follows:

$$P = \begin{pmatrix} x \\ y \\ z \end{pmatrix} \tag{11}$$

$$A = \begin{pmatrix} \phi \\ \theta \\ \psi \end{pmatrix} \tag{12}$$

Following that, the angular velocity matrix is defined as follows:

$$\omega = \begin{pmatrix} 1 & 0 & -s_\theta \\ 0 & c_\phi & c_\theta s_\phi \\ 0 & -s_\phi & c_\theta c_\phi \end{pmatrix} \dot{A} \tag{13}$$

Then, we define the rotation matrix that relates the body and inertial frames: (refer to Figure 8 for clearer understanding)

$$R = \begin{pmatrix} c_\phi c_\psi - c_\theta s_\phi s_\psi & -c_\psi s_\phi - c_\phi c_\theta s_\psi & s_\theta s_\psi \\ c_\phi c_\psi s_\theta + c_\phi s_\psi & c_\phi c_\theta c_\psi - s_\phi s_\psi & -c_\psi s_\theta \\ s_\phi s_\theta & c_\phi s_\theta & c_\theta \end{pmatrix} \tag{14}$$

In the next step, the thrust provided by each propeller is obtained from our own drone's model:

$$T_i(s) = \frac{1800}{s^2 + 15s} u_i(s) \tag{15}$$

In time domain this relation is as follows:

$$T_i(t) = (1 - e^{-15t}) 120 u_i(t) \tag{16}$$

This can be simplified to look like this:

$$T_i = Ku_i \tag{17}$$

And the total thrust is the sum of the thrust provided by the four propellers:

$$T_B = \begin{pmatrix} 0 \\ 0 \\ \sum_{i=1}^{4} T_i \end{pmatrix} \tag{18}$$

The drag force vector is then defined as:

$$F_D = -k_d \begin{pmatrix} \dot{x} \\ \dot{y} \\ \dot{z} \end{pmatrix} \tag{19}$$

As for the angular thrust, we also used our own model and the vector is given as below:

$$\tau_B = \begin{pmatrix} LK(u_1 - u_3) \\ LK(u_2 - u_4) \\ b(u_1 - u_2 + u_3 - u_4) \end{pmatrix} \tag{20}$$

The next step is to formulate the equations of motion. Newton's laws yield the following:

$$m\ddot{P} = mG + RT_B + F_D \tag{21}$$

So the linear acceleration vector is defined as below:

$$\ddot{P} = G + \frac{RT_B}{m} + \frac{F_D}{m} \tag{22}$$

Where G is defined as the gravity vector:

$$G = -k_d \begin{pmatrix} 0 \\ 0 \\ -g \end{pmatrix} \tag{23}$$

Then, Euler's equations yield the following:

$$I\dot{\omega} + \omega \times (I\omega) = \tau_B \tag{24}$$

So the rotational acceleration vector is defined as below:

$$\dot{\omega} = I^{-1}\Big(\tau_B - \omega \times (I\omega)\Big) \tag{25}$$

Where I is the inertia matrix defined as follows:

$$\omega = \begin{pmatrix} I_{xx} & 0 & 0 \\ 0 & I_{yy} & 0 \\ 0 & 0 & I_{zz} \end{pmatrix} \tag{26}$$

The parameters used are defined in Table 9 and their values are listed in Table 10.

Table 9: Drone controller parameter descriptions

| Parameter | Description |
|---|---|
| $\delta$ | Error vector |
| $e_\phi$ | Error of angle in roll direction |
| $e_\theta$ | Error of angle in pitch direction |
| $e_\psi$ | Error of angle in yaw direction |
| $K_p$ | Proportional gain |
| $K_d$ | Derivative gain |
| $K_i$ | Integrator gain |

All the parameters and equations were then implemented under MATLAB. Also, two functions were created, the first one computes the linear acceleration while the other one computes the angular acceleration. After having completed the dynamic model of the drone, we proceed to the design the stabilizer, or the inner loop controller for the drone. In this case, we opted to follow one of the methods used by [55] and construct a PID controller.

Table 10: Drone controller parameter values

| **Parameter** | **Value** |
|---|---|
| $K_p$ | 0.7 |
| $K_d$ | 0.3 |
| $K_i$ | 0.1 |

The error vector is defined as follows:

$$\delta = \begin{pmatrix} e_\phi \\ e_\theta \\ e_\psi \end{pmatrix} \tag{27}$$

Where each error element is shown below:

$$e_\phi = K_d\dot{\phi} + K_p\int_0^T \dot{\phi}dt + K_i\int_0^T\int_0^T \dot{\phi}dtdt \tag{28}$$

$$e_\theta = K_d\dot{\theta} + K_p\int_0^T \dot{\theta}dt + K_i\int_0^T\int_0^T \dot{\theta}dtdt \tag{29}$$

$$e_\psi = K_d\dot{\psi} + K_p\int_0^T \dot{\psi}dt + K_i\int_0^T\int_0^T \dot{\psi}dtdt \tag{30}$$

The error elements are then utilized to obtain the controlled PWM input signals to the individual propellers. The functions to determine the required input to the propellers were derived and are shown below:

$$u_1 = \frac{mg}{4Kc_\phi c_\theta} - \frac{e_\psi I_{zz}}{4b} - \frac{e_\theta I_{xx}}{2KL} \tag{31}$$

$$u_2 = \frac{mg}{4Kc_\phi c_\theta} + \frac{e_\psi I_{zz}}{4b} - \frac{e_\theta I_{yy}}{2KL} \tag{32}$$

$$u_3 = \frac{mg}{4Kc_\phi c_\theta} + \frac{e_\psi I_{zz}}{4b} - \frac{e_\theta I_{xx}}{2KL} \tag{33}$$

$$u_4 = \frac{mg}{4Kc_\phi c_\theta} + \frac{e_\psi I_{zz}}{4b} + \frac{e_\theta I_{yy}}{2KL} \tag{34}$$

These equations along with the parameter values provided were then put into another MATLAB function that takes in the current state and return the required propeller PWM inputs.

## 2.2. Proposed Approach

In this thesis, it is proposed that the methodology of RRT* that is presented in [19] be enhanced in being more flexible to the environment and having reduced processing times. This improvement is to be done by incorporating a Gaussian distribution into the process of generating nodes. Our modified form of RRT* will be referred to as Rapidly-Exploring Random Trees Star Normal (RRT*N) where Normal refers to the normal or Gaussian prob- ability distribution. The aim is also to give it more capabilities to handle 3D static and dynamic environments. For the improved RRT*N, the information about the obstacles is fed into the algorithm, allowing for better paths even in dynamic environment. This modification will be explained in more details later.

**2.2.1. Novel modification.** Our novel modified algorithm, RRT*N, produces paths of similar quality to the classic RRT* algorithm in less time using a new technique to generate the nodes. The idea is to change the way the nodes are generated from being uniformly random to a user-controlled normal distribution function that effectively determines the density distribution of the nodes in an imaginary cone pointing in direction of the goal. An imaginary vector is created that connects $Z_{init}$ and $Z_{goal}$ and nodes are

only generated within a specific distance from that line. The line is represented by a vector $L$. The distance from the line is based on a given standard deviation parameter, which can be based on the magnitude of the distance between $Z_{init}$ and $Z_{goal}$ or the largest projected width of the obstacles in the direction normal to $L$.

This limitation on the space where to generate the nodes, results in concentrated nodes being generated along the direction of $Z_{goal}$ hence prioritizing branches growing towards the goal region, resulting in greatly improved processing times with paths of similar lengths and quality. The algorithm main difference is how the nodes are generated. However, that small change has huge effect on the outcome. The points that would have been directly fed to the algorithm in a classic RRT* are instead manipulated first in RRT*N before they are used. The points are limited to be created along the line $L$ and are then deviated from it based on the specified Gaussian distribution. The most important criteria for this approach to be significantly superior to the classic algorithm, there has to be connectivity to the goal point through a forward pointing cone. The Gaussian probability distribution law is shown in Equation 35 and Figure 10.

$$P(c) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(c-\mu)^2}{2\sigma^2}}, \quad c = x_n, y_n \tag{35}$$

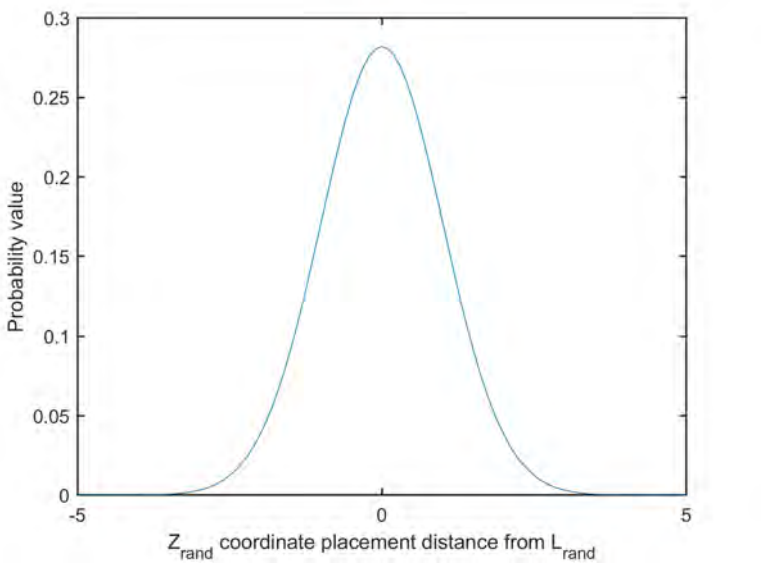


Figure 10: Gaussian distribution

Figure 11 demonstrates the typical node distribution produced by RRT* (left) and RRT*N (right). This gives a clearer idea of how the incorporated Gaussian distribution will impact the final tree. The higher density of nodes along the direction of the target in RRT*N translate to faster path generation.

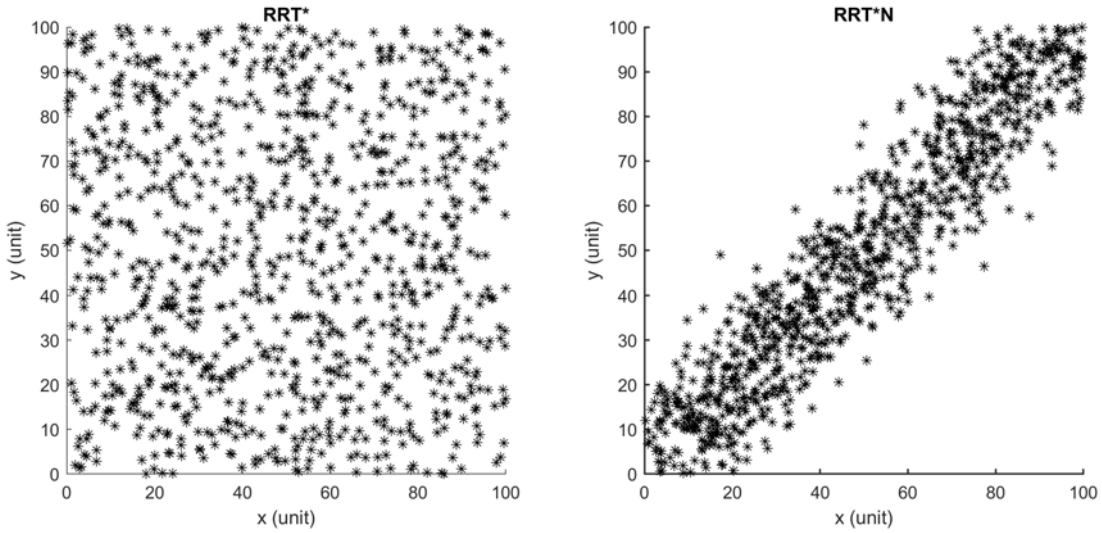


Figure 11: Node distribution

Where $\mu$ is the mean which is taken as zero in all our trials and $\sigma$ is the desired standard deviation. The variable $\sigma$ is varied manually in the static tests, whereas in the dynamic tests, $\sigma$ is automatically varied to avoid being stuck in local minima. The magnitude of $\sigma$ affects how far from the line $L$ the nodes are generated. A small value of $\sigma$ generates nodes close to the line $L$, whereas a high value of $\sigma$ makes the RRT*N to behave like the classical RRT*.

The way the equation above works is that the node returned by "Sample", now called $L_{rand}$, will have all its components ($x_n$, $y_n$, and possibly $z_n$) fed into it and then it will calculate the probability distribution and outputs the location of $Z_{rand}$. However, the large distinction in this work is that the way $\sigma$ changes is based on a function of the environment and obstacle properties. This is critical as the value of $\sigma$ is directly related to the performance of RRT*N. Selected to be too small and it is susceptible to local minimum issues and selected to be too large and the processing time will suffer. The

pseudocode of the RRT*N is shown in Algorithm 2. Also in Figure 12 we demonstrate how a typical run of RRT*N looks like in a 100x100 free environment.

**Algorithm 2** RRT*N Algorithm Pseudocode

$T$ initialized through *InitializeTree()*
$T$ has initial node added by *InsertNode*($Z_{init}$)
$L$ generated through *Norm*($Z_{init}$, $Z_{goal}$)
**for** i = 0:N **do**
  $L_{rand}$ obtained through *Sample(i)*
  $Z_{rand}$ obtained through *Gaussian*($L_{rand}$)
  $Z_{nearest}$ obtained through *Nearest*($T$, $Z_{rand}$)
  $Z_{new}$ obtained through *Steer*($Z_{nearest}$, $Z_{rand}$)
  **if** *ObstacleFree*($Z_{new}$) **then**
    $Z_{near}$ obtained through *Near*($T$,$R_n$, $Z_{new}$)
    $Z_{min}$ obtained through *ChooseParent*($Z_{near}$, $Z_{nearest}$, $Z_{new}$)
    $T$ updated by *InsertNode*($T$, $Z_{min}$, $Z_{new}$)
    $T$ rewired by *Rewire*($T$, $Z_{near}$, $Z_{min}$, $Z_{new}$)
  **end if**
**end for**
*Path* is the least cost path extracted from $T$

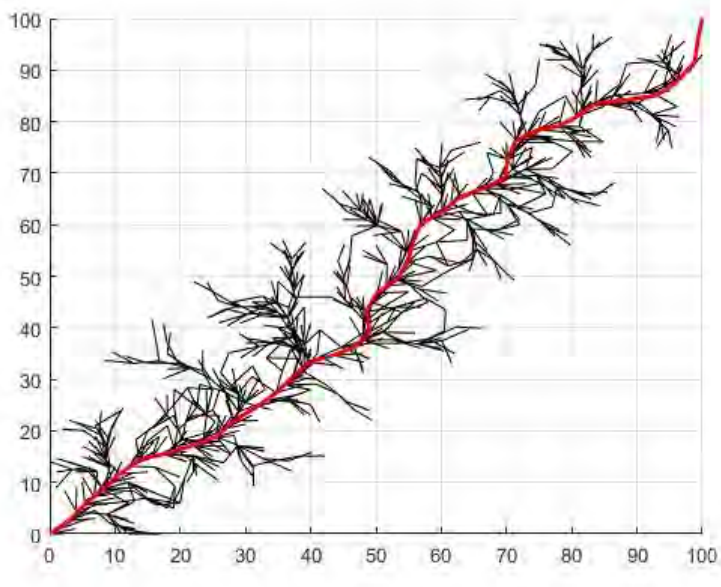


Figure 12: RRT*N typical 2D trial in free environment

**2.2.2. Extension to 3D.** To implement the RRT* technique in a 3D environment, we need to define the configuration space to be in 3D and consequently the obstacles need to be defined as 3D shapes. Also, the nodes need to have an additional dimension in which they can be generated. the performance of the classic algorithm and its improved version presented in this work in static and dynamic situations will be shown in a later section. For now, Figures 13 and 14 show how a typical 1000 node trial looks like in 3D using RRT* and RRT*N respectively. All tests had a step size of 3 and a neighborhood radius of 7.5. It is to be noted that RRT* ran out of nodes before reaching the goal unlike the other two algorithms.

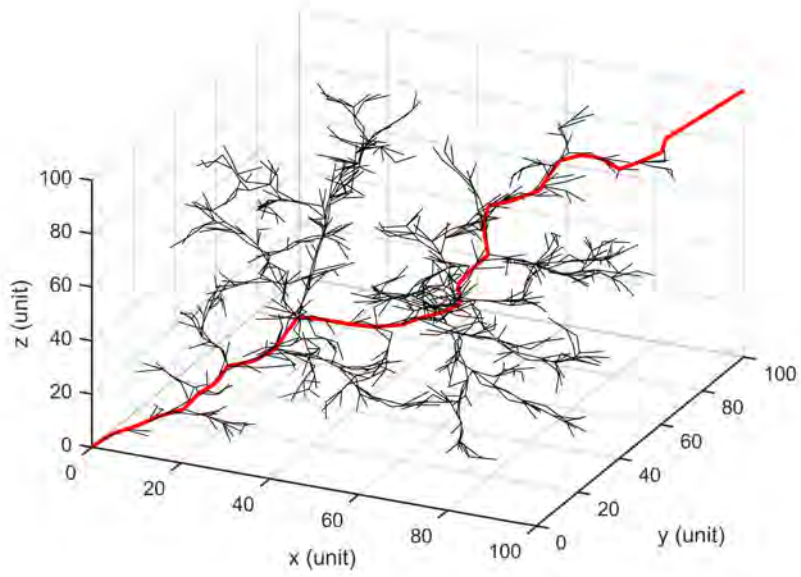


Figure 13: RRT* typical 1000 node trial in free environment

Figures 13 and 14 only demonstrate the output acquired for a specific experiment with the two algorithms. The parameters used are as follows: 100x100x100 empty map with the starting point being (0, 0, 0) and the goal point is (100, 100, 100). Vision used is 5 units, for RRT*N the $\sigma$ used was 5% of the magnitude of $L$. This method could be useful in works related to what is discussed in [56]. This thesis will address simulations and testing of RRT*N simulations in 3D static and dynamic scenarios. A specific environment with fixed obstacles will be used for the comparison of the

RRT* and the proposed RRT*N. The environment used is constructed from five static obstacles shaped as spheres, which are blocking a direct path towards the goal but still allowing for a comfortable maneuvering in between them.

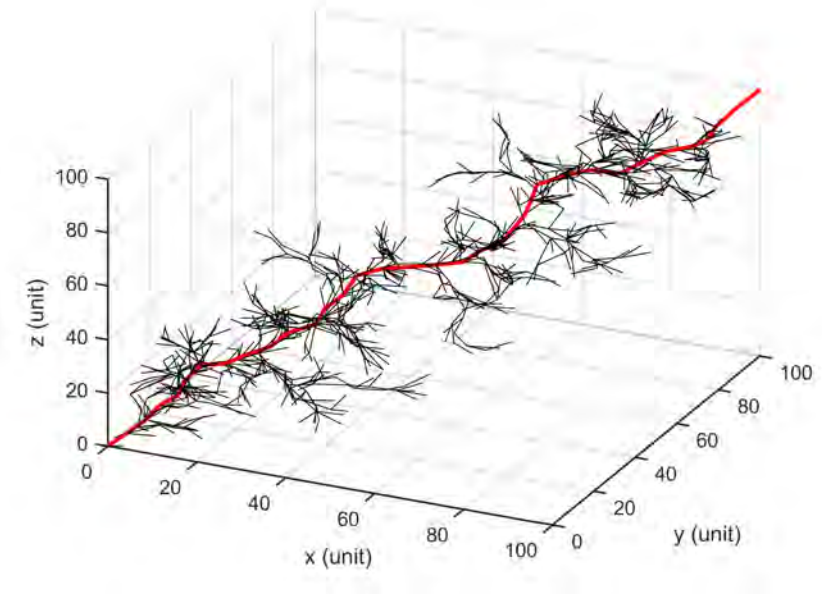


Figure 14: RRT*N typical 1000 node trial in free environment

All five obstacles are spherical with radius of 15 units. The location of their centers in cartesian coordinates are the following: (67, 37, 32), (47, 77, 22), (37, 57, 67), (32, 32, 17), and (77, 77, 47). Again, like before, the environment is 100x100x100 units with the starting point being (0, 0, 0) and the goal point is (100, 100, 100). RRT* and RRT*N were each executed 100 times each with a node limit of 10000 in each trial. All the parameters are the same as previously. A new assessment parameter will be introduced, which represents the percentage of trials in which the tree can reach the target within a "Vision" distance of the goal region before running out of nodes. This parameter will be called success rate.

**2.2.3. Adapting algorithm to dynamic obstacles.** So far, only static situations were discussed. To extend the RRT*N to dynamic scenarios further modifications are needed. The modification consists of detecting the nodes having either a conflict with a moving obstacle or are at risk of conflicting. This check is made after the initial

tree is constructed and a path is obtained. Any collisions/conflicts associated with that path are then rectified by eliminating the appropriate nodes and re-initializing the tree at a node at an appropriate offset from the conflict and setting the new goal point also at an appropriate offset from the other side of the conflict.

This can be visualized at cutting off an undesirable portion of the tree and creating it with the new information to stitch back the missing part of the path. Meaning that the nodes that are within a predefined range of any newly detected dynamic obstacle are immediately discarded and a new tree is grown to connect the two severed ends of the original tree. The severed ends are now the new starting and ending points. This process is done whenever the robot detects a new object within its vision range . Once that occurs, a new sub algorithm is executed to deal with the situation. As mentioned before, a similar approach to what is presented in [33] is adopted to approach this problem. A pseudocode of the new algorithm is shown in Algorithm 3.

**Algorithm 3** RRT*N Dynamic Obstacle Sub-Algorithm Pseudocode

**while** *DynamicObstacleCollision(Obs, P)* **do**
    Identify and reject invalid nodes from *Path*
    Select new starting and goal points
    Re-execute Algorithm 2 to obtain $T_{dynamic}$
    $Path_{dynamic}$ is the least cost path extracted from $T_{dynamic}$
    *Path* is updated to incorporate $Path_{dynamic}$
**end while**

Algorithm 3 will be in action as soon as the initial RRT*N is finished processing and provides an initial path for the environment with known static obstacles. The situation in which the test will be done is a free environment where a dynamic obstacle will appear once the robot has it within its vision range. An example of what is produced is demonstrated in 2D in Figures 15 and 16 and in 3D in Figures 17 and 18. It is important to note that this process of cutting and stitching will be executed in a loop, meaning that whenever there is a consequent collision after a restitching process, it will be executed again.

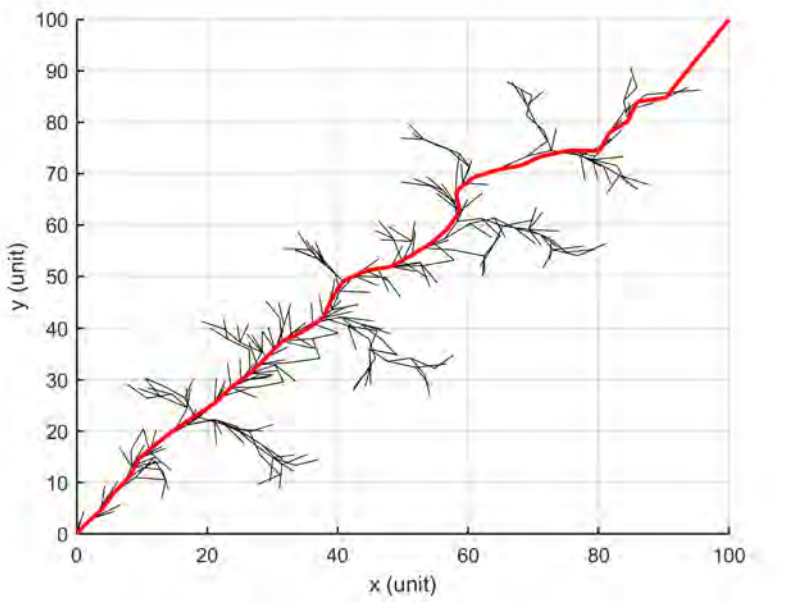


Figure 15: Initial path generated by RRT*N in 2D environment before encountering the dynamic obstacle

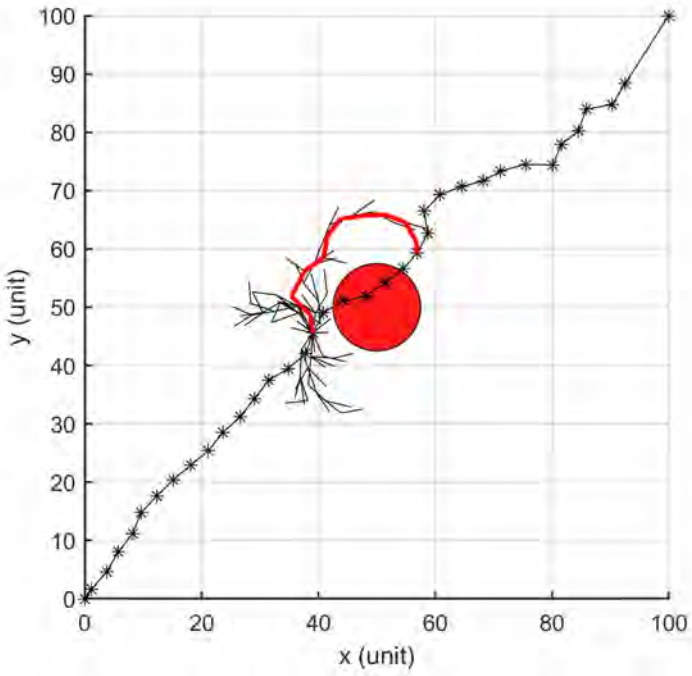


Figure 16: Re-stitched path generated by the secondary RRT*N loop in 2D environment after encountering the dynamic obstacle

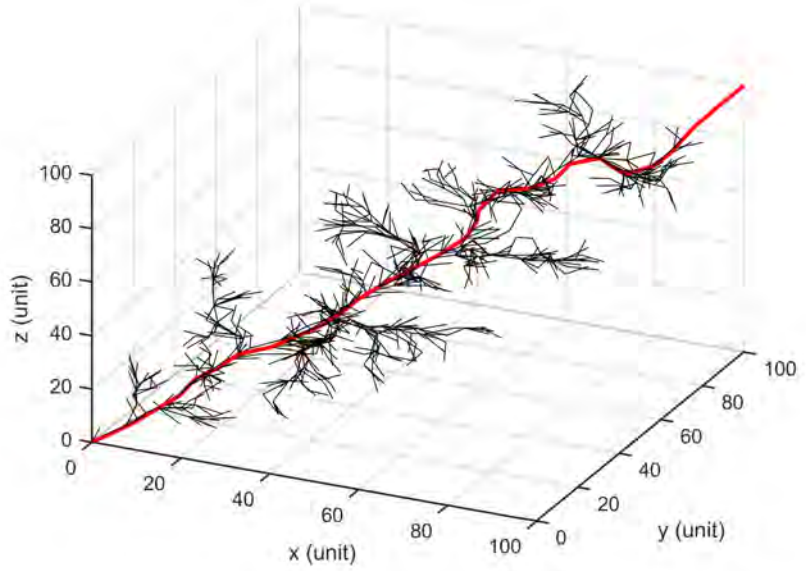


Figure 17: Initial path generated by RRT*N in 3D environment before encountering then dynamic obstacle

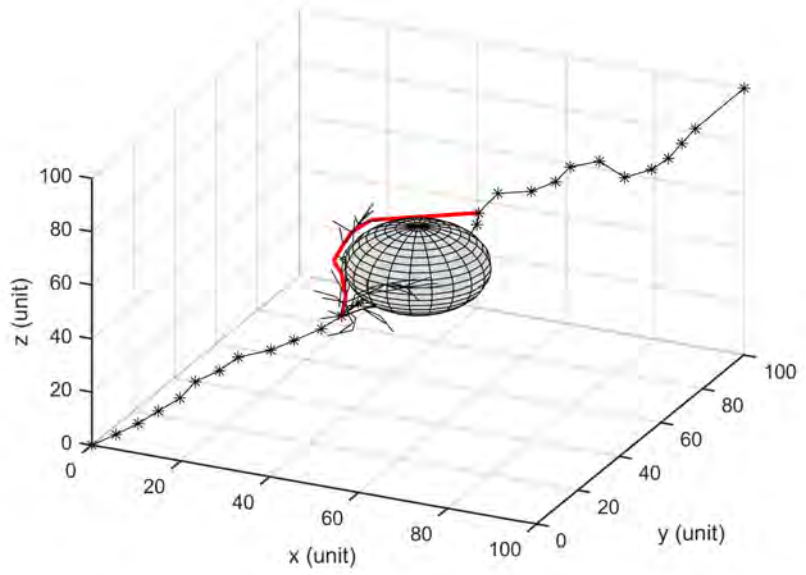


Figure 18: Re-stitched path generated by the secondary RRT*N loop in 3D environment after encountering the dynamic obstacle

As seen in both the 2D and 3D case, the nodes from the initial path that are no longer valid once the dynamic obstacle has been detected, need to be replaced. This is done by the re-stitching process shown in Figures 17 and 18 and explained earlier in Table 3. Further testing and in detail analysis of the outputs and statistic of the results will be shown in the following section.

**2.2.4. Path smoothing.** Even after having a greatly improved algorithm in terms of processing time, we still have paths that have similar path quality to the classical method. These paths are often irregular and hence are not effective for practical implementation or maybe even not dynamically feasible if the robot platform cannot move in such jagged paths.

Due to this, it was decided to undertake an approach to improve upon the obtained path quality through a moving average zero phase shift filter meaning that the path starting and ending points will not be subject to any shifting and this is done by filtering the path coordinates in both directions (forward and reverse). This process is done utilizing the "filtfilt" function under MATLAB [57]. The filter type undertaken is a Finite Impulse Response (FIR) filter, which is constructed as a discrete transfer function.

The obtained function is a transfer function with a denominator always being a single term with coefficient and order equal to the window size rounded off and the numerator has terms one more than the rounded off window size value with incremental orders (zero to window size value rounded off) where all the coefficients are one.
The window size value in our case is usually taken to be as follows:

$$WindowSize = \frac{P}{Z} \tag{36}$$

Where P is the number of nodes present in the final calculated path and Z is a scaling factor that determines the linearity/smoothness of the path. We usually take Z to be 8 because by trial and error it was found that Z should be around 8% of the environment dimension.

An example of such filter where the window size is 5 is shown below:

$$F = \frac{z^5 + z^4 + z^3 + z^2 + z + 1}{5z^5} \tag{37}$$

It is to be noted however, that despite having introduced a safety factor to all the environmental obstacles, there remains the risk that the smoothed path may be in conflict with the environment. Due to this reason, we incorporated the looping control algorithm which is Algorithm 3 that was discussed in the previous subsection (2.2.3) that ensures that the smoothed path would never be at risk of crossing the specified safety boundaries.

# Chapter 3: Results and Discussion

This chapter will present the compiled results for the simulation and implementation tests carried out to compare our proposed approach to the classic RRT* method. The testing will include 2D simulation, 2D implementation, and 3D simulation in static scenarios then in dynamic scenarios. Also, the hardware and software utilized for this work are presented.

## 3.1. Hardware and Software Used

For all the simulations presented in this thesis, the following software and hardware specs were used:

- Software: MATLAB 2017a
- Operating system: Windows 10 64-bit
- Processor: Intel Core i7-3610QM 4-core 8-thread running at 2.3 GHz clock speed
- Memory: dual-channel DDR3 8GB running at 800MHz clock speed

And for all the implementations presented in this thesis, the following software and hardware specs were used:

- Software: MATLAB and SIMULINK 2017b and Quarc 2.7.2220
- Operating system: Windows 10 64-bit
- Processor: Intel Core i5-4460 4-core 8-thread running at 3.20 GHz clock speed
- Memory: dual-channel DDR3 4 GB running at 800 MHz clock speed

The Qbot2, discussed previously in subsection 2.1.2, is the platform on which the implementation was carried out. Its specifications are listed below:

- On-board computer: Gumstix DuoVero Zephyr with integrated 802.11 b/g/n Wi-Fi
- QUARC maximum sample rate: 1000 Hz

- Camera resolution: 640x480
- Depth sensing: 11-bit
- Depth sensor range: 0.5-6 m

It is to be noted that the sensors used for the localization were 2 wheel encoders, a 3-axis gyroscope, and a z-axis angle measurement (heading).

## 3.2. Static Scenario

In this section we will be presenting some environments, in which our novel RRT*N is tested against the classic RRT*. The performances of the two methods related to the processing time to generate an initial path while maintaining similar path length and quality, will be compared.

**3.2.1. Simulation in 2D.** The first step in the static testing process is to test our new algorithm in 2D simulation and compare its performance to RRT*. We present an environment with three square obstacles. The environment where the test is carried out is shown in Figure 19.

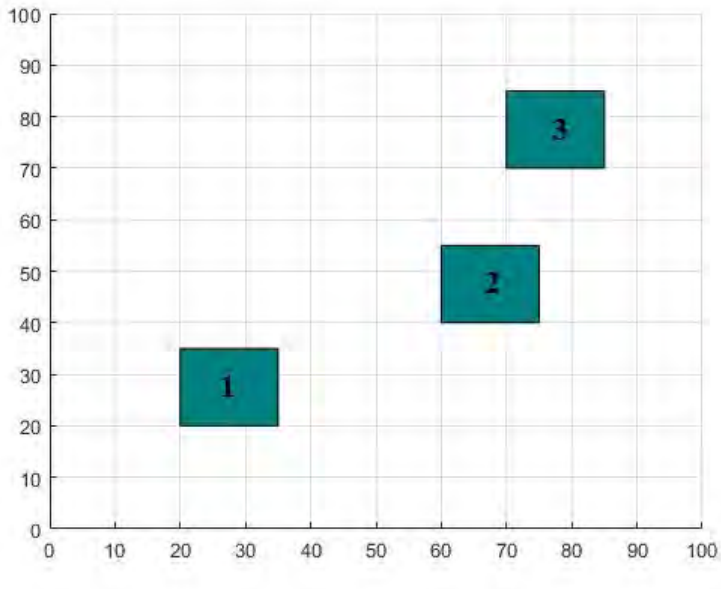


Figure 19: Environment used for first 2D static test

The environment size is 100x100 units with both axes ranging from 0 to 100 units. The obstacle sizes and the location of their bottom left vertices are in Table 11.

Table 11: First 2D static test obstacle details

| Obstacle number | x-y Dimension (units) | x-y Location |
|---|---|---|
| 1 | 15x15 | (20, 20) |
| 2 | 15x15 | (60, 40) |
| 3 | 15x15 | (70, 70) |

The test was then executed with the following parameters and a typical run for RRT* and RRT*N are shown in Figures 20 and 21.

- 2500 nodes
- Starting point: (0, 0)
- Goal point: (100, 100)
- Step size: 2 units
- Neighborhood size: 5 units
- Robot vision range: 5 units
- Normal distribution: 2 $\sigma$ within 15% of $L$

The results obtained are for 100 trials for each algorithm and they are summarized in Table 12.

Table 12: First 2D static test results

| Parameter | RRT* | RRT*N |
|---|---|---|
| Average time (sec) | 48.55 | 14.50 |
| Time standard deviation (sec) | 17.25 | 8.01 |
| Average length (units) | 166.13 | 161.67 |
| Length standard deviation (units) | 5.11 | 4.26 |
| Success rate | 62% | 99% |

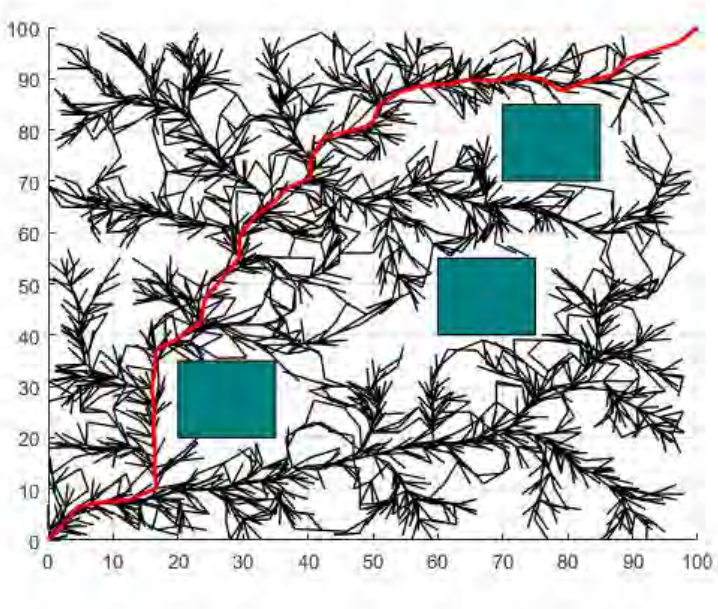


Figure 20: Typical RRT* run in first 2D static scenario

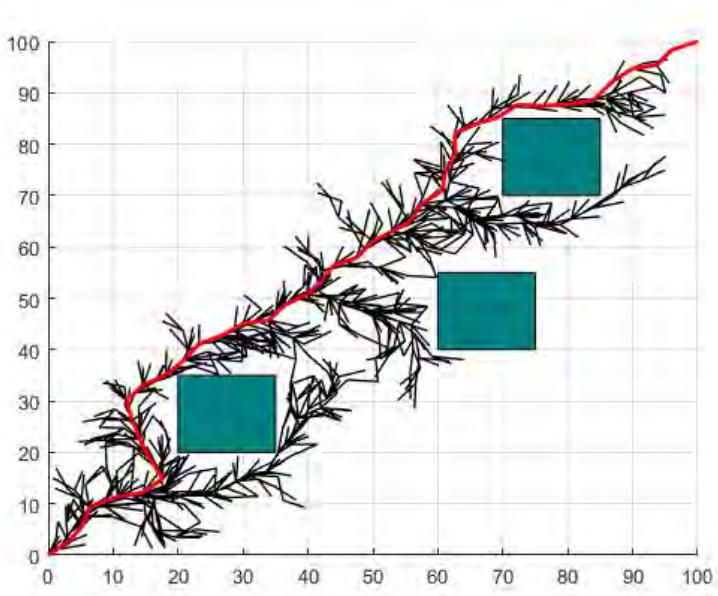


Figure 21: Typical RRT*N run in first 2D static scenario

The trees generated in Figures 20 and 21 show clearly the impact of the normal distribution in RRT*N and how it directly relates to reducing processing time. The full results are presented in Figures 22 and 23.

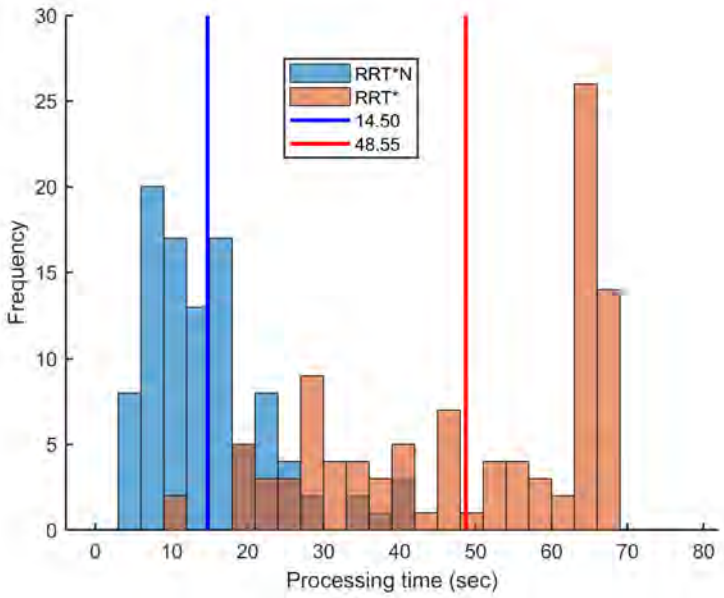


Figure 22: Algorithms CPU times distribution for 100 runs in first 2D static scenario

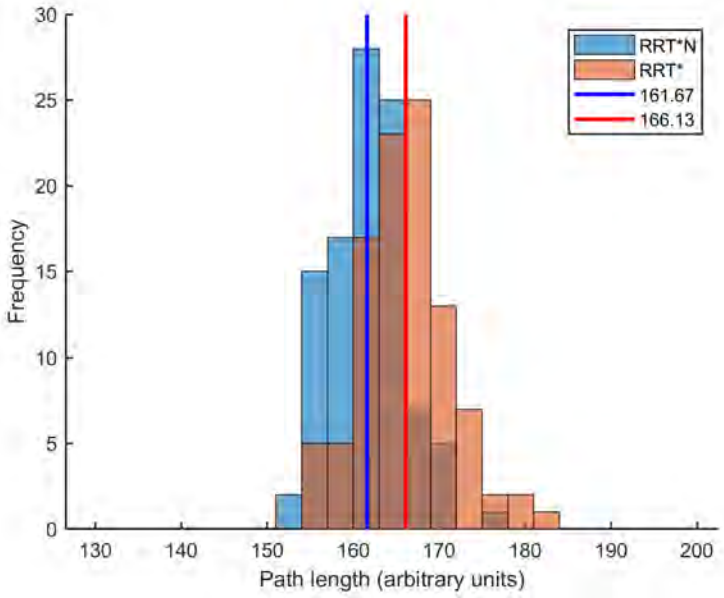


Figure 23: Algorithms path lengths distribution for 100 runs in first 2D static scenario

The results show clearly the superiority of RRT*N as it is capable of providing paths of similar lengths and quality to the ones obtained by the RRT* but only consum-

ing 30% of the processing time taken by the classic method. However, the first test is rather basic and a more realistic test is to be carried out. This brings us to the second test in 2D static environment. The environment size is 200x200 units with both axes ranging from -100 to 100 units. The environment where the test is carried out is shown in Figure 24. The obstacle sizes and the location of their bottom left vertices are in Table 13.

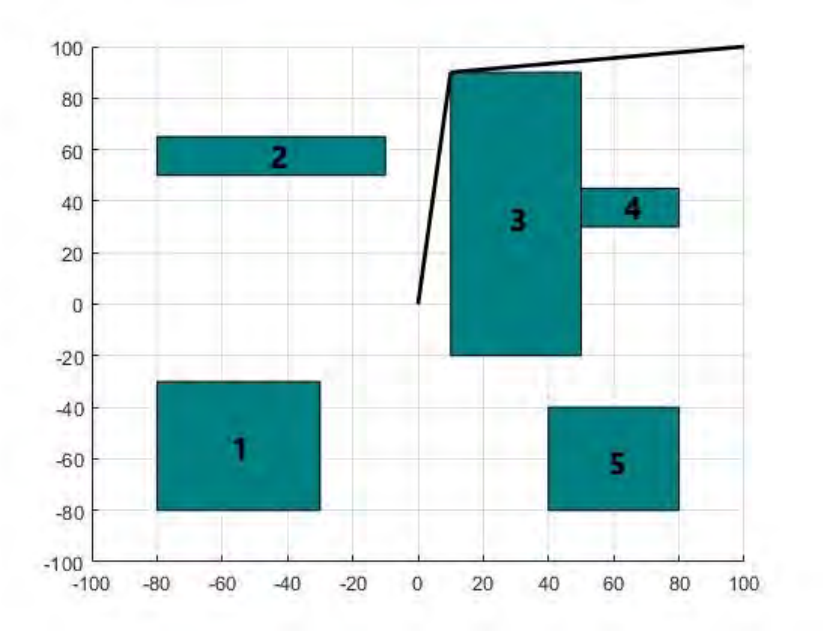


Figure 24: Environment used for second 2D static test and the theoretical shortest path

This is a more comprehensive test compared to the previous one as it demonstrates a scenario where both algorithms will not be capable of going along the least cost path so it restricts both due to the nature of the geometry. This gives us a more consistent comparison of the performance of RRT*N compared to RRT*.

Table 13: Second 2D static test obstacle details

| **Obstacle number** | **x-y Dimension (units)** | **x-y Location** |
|---|---|---|
| 1 | 50x50 | (-80, -80) |
| 2 | 70x15 | (-80, 50) |
| 3 | 40x110 | (10, -20) |
| 4 | 30x15 | (50, 30) |
| 5 | 40x40 | (40, -80) |

The test was then executed with the following parameters and a typical run for RRT* and RRT*N are shown in Figures 25 and 26.

- 5000 nodes
- Starting point: (0, 0)
- Goal point: (100, 100)
- Step size: 2 units
- Neighborhood size: 5 units
- Robot vision range: 5 units
- Normal distribution: 2 $\sigma$ within 45% of $L$
- Theoretical shortest path length: 181.11 units

As mentioned, with both algorithms it is the least likely scenario that they will follow the least cost path as it goes through a narrow passage hence restricting the propagation of the tree. Despite that we see RRT*N's superiority as it neglects the irrelevant regions and focuses on going around the relevant obstacle and towards the goal.

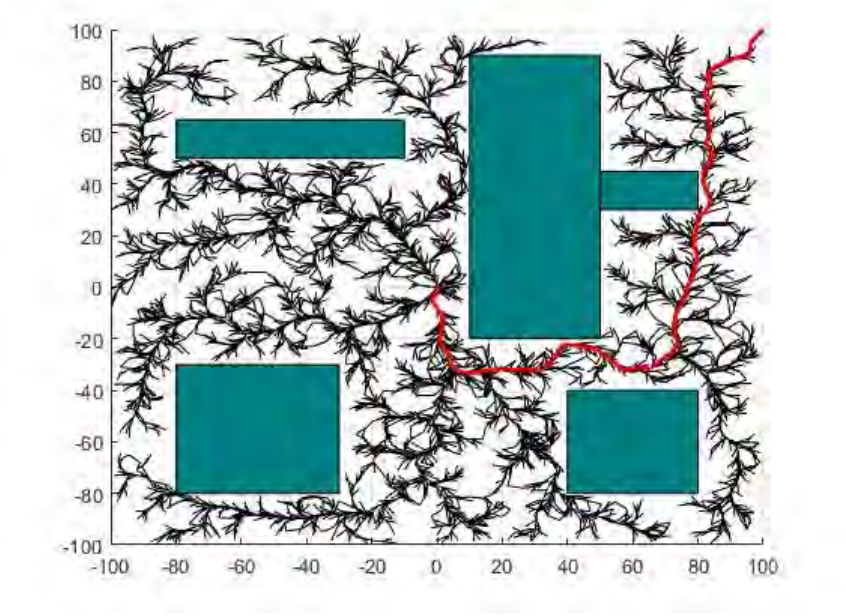


Figure 25: Typical RRT* run in the second 2D static scenario

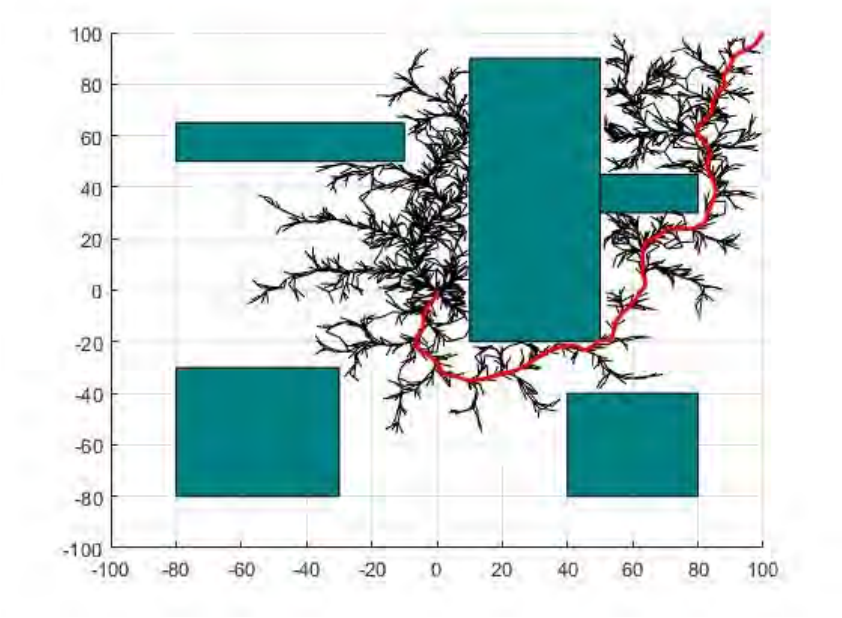


Figure 26: Typical RRT*N run in the second 2D static scenario

It is clear that the RRT*N is more likely to reach $Z_{goal}$ quicker due to its probability distribution having a higher tendency of driving its branches towards the goal region. This also contributes to it not favoring exploring distant regions in the map which typically is time consuming. However, in both cases, the path was not along the direction of the shortest path mainly due to the relatively small number of nodes used in those trials which goes to show that RRT*N does not contribute much towards the quality or the cost of the path as it does with the computational time. Average length is the average geometric lengths of the final obtained paths and the success rate is the percentage of the trials in which the tree managed to reach within vision range of the goal point. The results obtained are for 250 trials for each algorithm and they are summarized in Table 14 and demonstrated in Figures 27 and 28.

Table 14: Second 2D static test results

| **Parameter** | **RRT*** | **RRT*N** |
|---|---|---|
| Average time (sec) | 121.58 | 38.46 |
| Time standard deviation (sec) | 11.42 | 19.18 |
| Average length (units) | 259.11 | 240.99 |
| Length standard deviation (units) | 29.41 | 13.32 |
| Success rate | 7.2% | 94% |

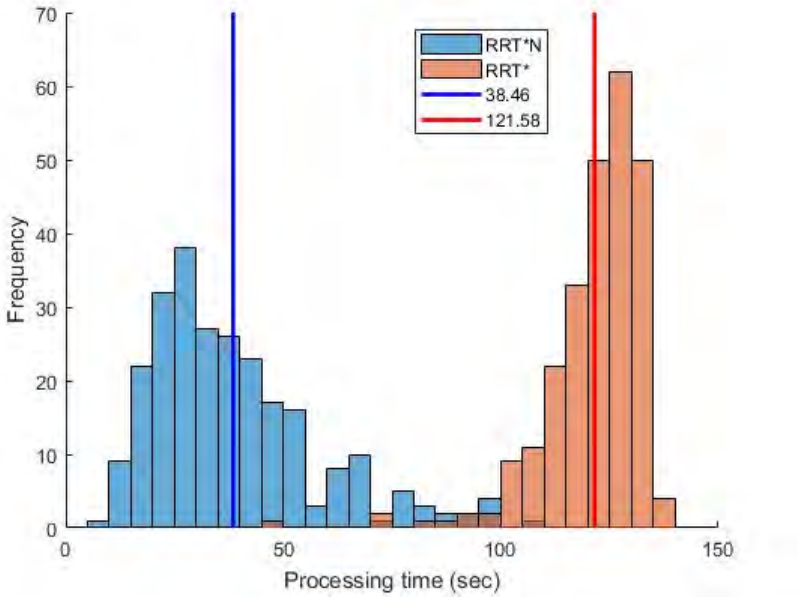


Figure 27: Algorithms CPU times distribution for 250 runs in the second 2D static scenario

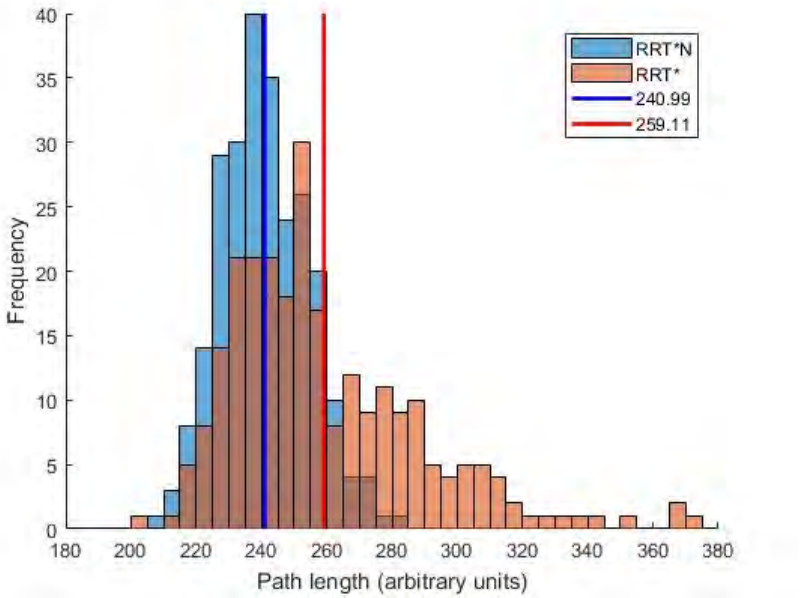


Figure 28: Algorithms path lengths distribution for 250 runs in the second 2D static scenario

As seen from Table 14, RRT*N significantly improves on the CPU time on average (31.6% of RRT* time) with trade off in standard deviation (67.8% increase).

Also, path length in RRT*N is improved as it is 33.1% longer than the optimum path of length 181.108 discussed previously while RRT* is 43.1% while having a significantly lower standard deviation (45.3% of RRT*). Another important indicator is the success rate, with RRT*N having a success rate of 94% (235 out of 250 trials) and RRT* having only a 7.2% success rate (18 out of 250 trials) in this scenario.

**3.2.2. Implementation in 2D.** After having confirmed in the simulation that our algorithm is indeed superior in terms of processing time, we move on to implement it in an actual 2D static environment. The environment is the Mechatronics Lab at AUS. An image of the real environment is shown in Figure 29 followed by its construction in MATLAB shown in Figure 30. The green rectangles represent the tables and the walls.

Figure 29: Real environment used for implementation in 2D static scenario

It is to be noted that the obstacles that are reconstructed in MATLAB are only the major ones like the labels and walls and mentiond. This means that later in the tests any newly introduced obstacle will be detected and automatically incorporated.

Figure 30 only shows a MATLAB representation of the provided map and hence a lot of the minor details are missing. For instance the tables are considered as cubes view from the top, this means that the robot is not allowed to move below them but only around them. Also a safety factor of 5 centimeters is incorporated into all obstacles as a precautionary measure.

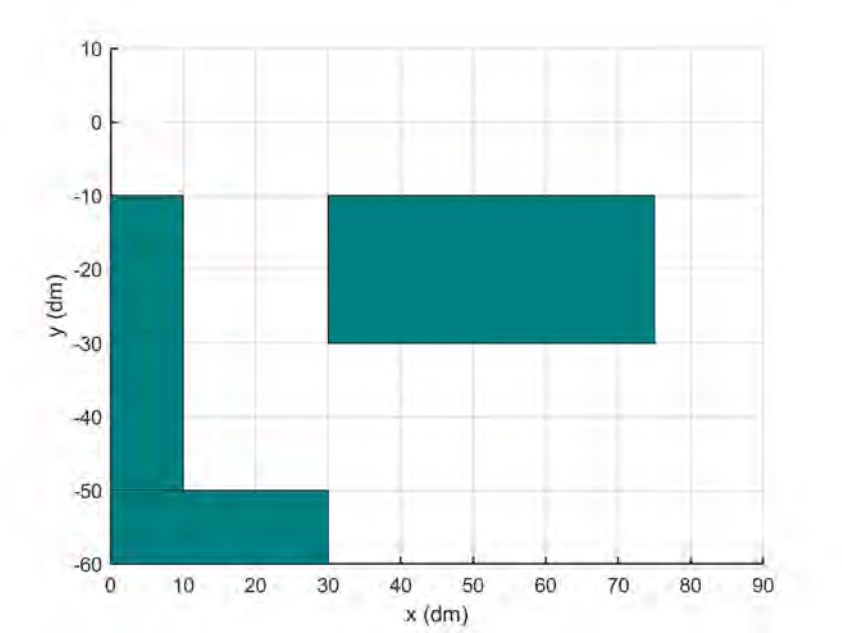


Figure 30: MATLAB reconstruction of environment used for implementation in 2D static scenario

The test was then executed with the following parameters and a typical run for RRT*N is shown in Figures 31 and 32.

- 5000 nodes
- Starting point: (0, 0)
- Goal point: (70, -50)
- Step size: 2 units
- Neighborhood size: 5 units
- Robot vision range: 5 units
- Normal distribution: 2 $\sigma$ within 45% of $L$

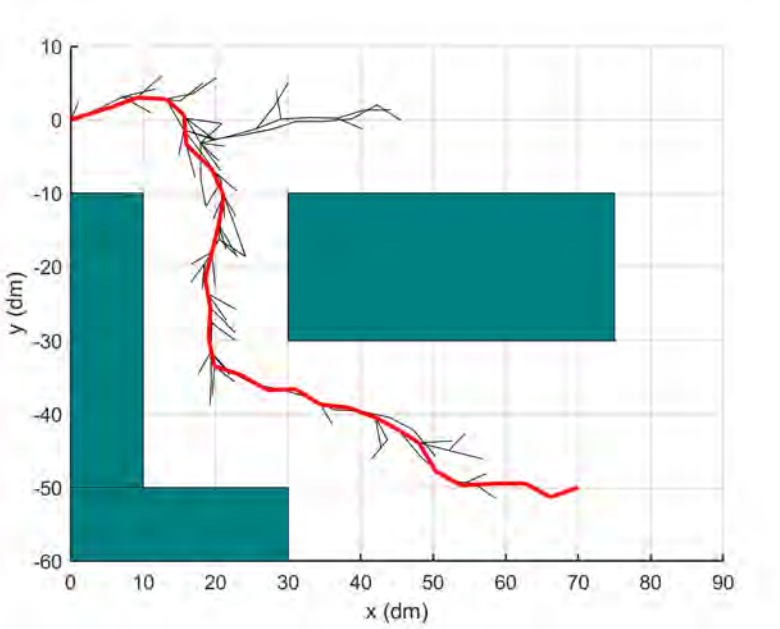


Figure 31: Typical RRT*N path for the provided 2D static scenario

Once the path is obtained, the Kobuki will now follow this path. The speed is set to 2 dm/sec and the maximum allowable rotational speed is $\pi$ rad/sec. The path that the robot follows as recorded by the encoders is shown in Figure 32.

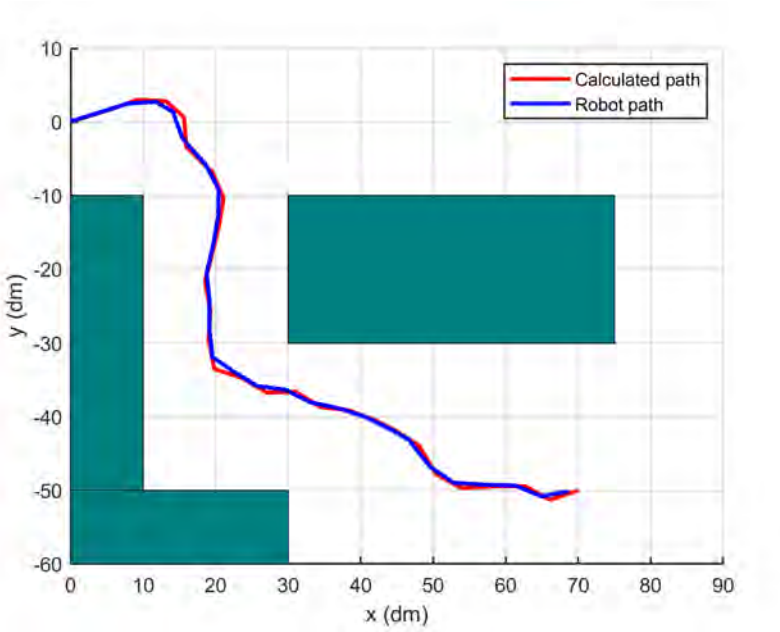


Figure 32: Typical RRT*N implementation in real 2D static scenario

The speed and rotational speed measurements for the robot in this particular trial are shown in Figures 33 and 34. It is visible that the speed is slightly fluctuating but still consistently following the specified value. The rotational speed is much more fluctuating because of the highly irregular path and the sampling time being 0.2 seconds.

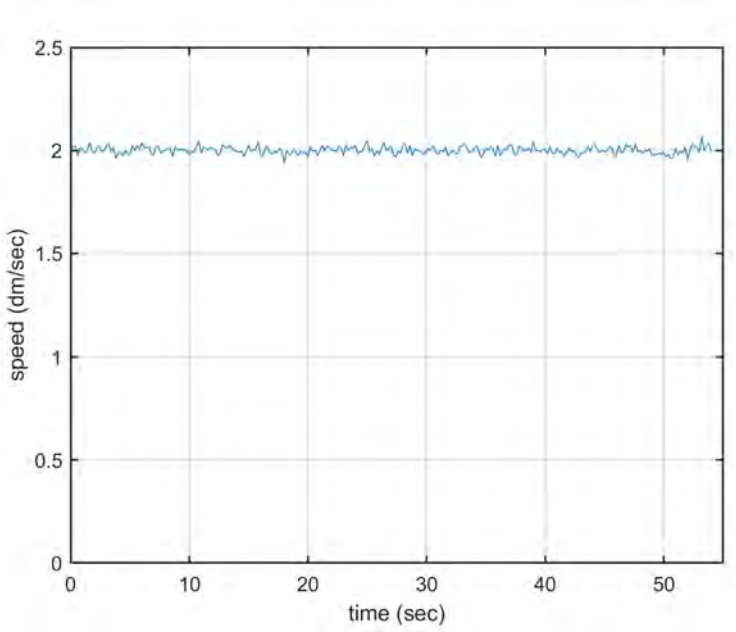


Figure 33: Kobuki speed in 2D static test

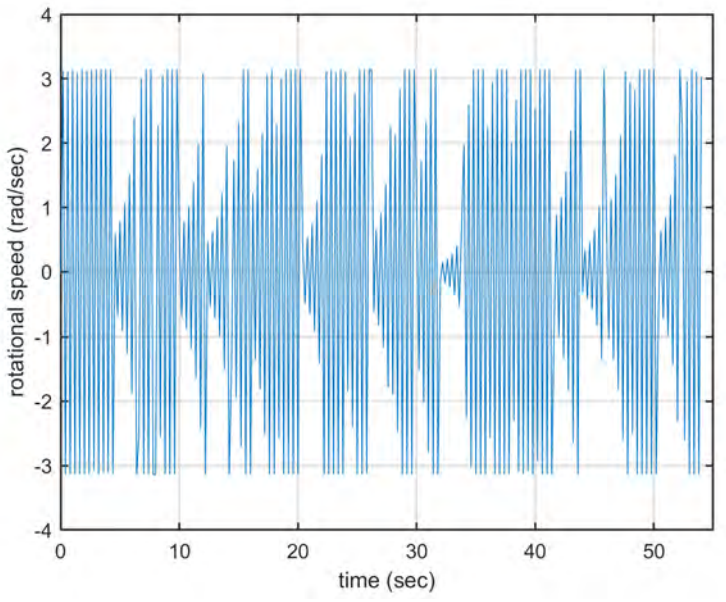


Figure 34: Kobuki rotational speed in 2D static test

The original and the smoothed paths are shown in Figures 35 and 36 following the approach discussed in subsection 2.2.2 for path smoothing.

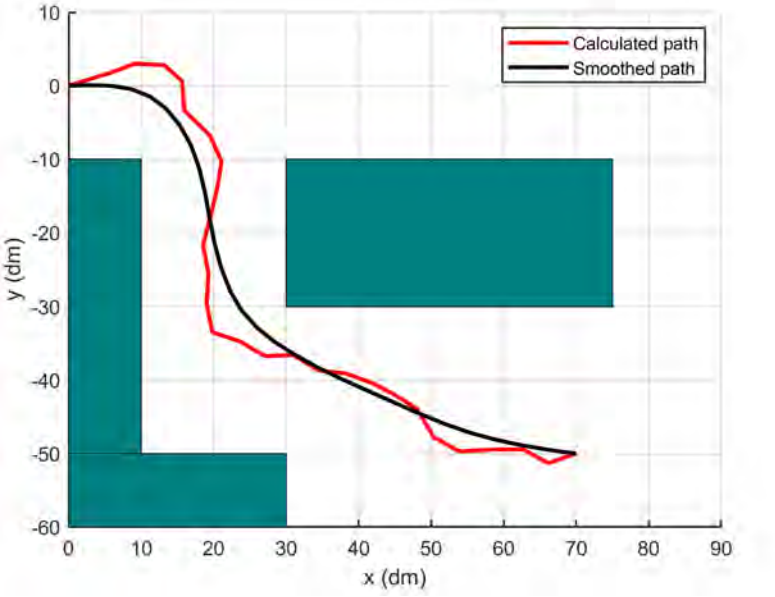


Figure 35: Final resultant path for robot following after smoothing

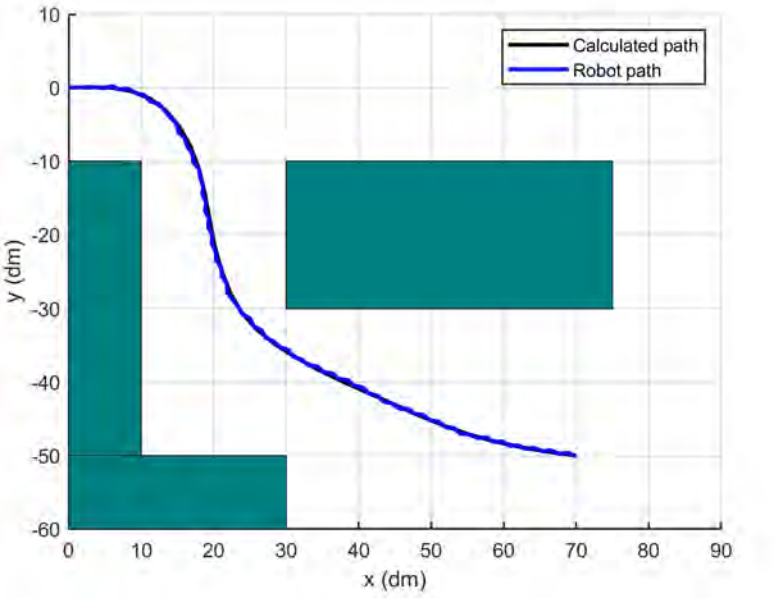


Figure 36: Typical smoothed RRT*N implementation in real 2D static scenario

Again, the speed and rotational speed measurements for the robot in this particular trial are shown in Figures 37 and 38. It is visible that the speed is slightly fluctuating but still consistently following the specified value. The rotational speed is still fluctuating but relatively less than seen in the unsmoothed scenario.

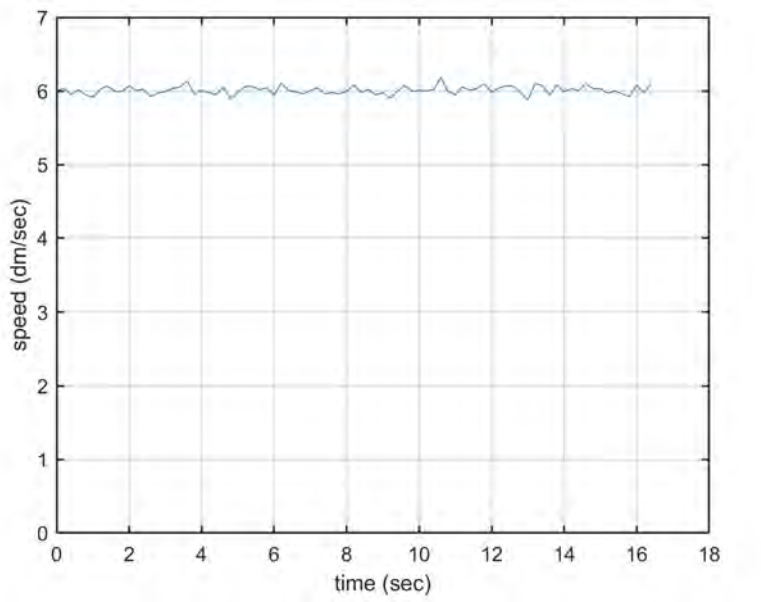


Figure 37: Kobuki smoothed speed in 2D static test

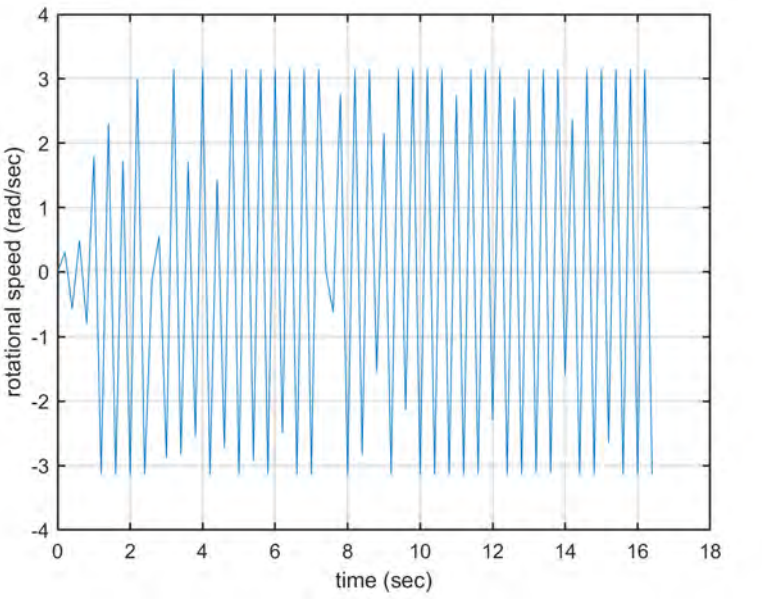


Figure 38: Kobuki smoothed rotational speed in 2D static test

It is clear that the smoothed path is much more feasible to approach the maximum possible speed. We still prefer to stay below the maximum speed as the speed is more stable and consistent when driven at 6 dm/sec.

**3.2.3. Simulation in 3D.** The final step in the static testing process is to test our new algorithm in 3D simulation and compare its performance to RRT*. Initially, we present an environment with five spherical obstacles blocking a direct path from the starting point towards the goal and have tested both algorithms in this scenario. The environment size is 100x100x100 units with all axes ranging from 0 to 100 units. The environment where the test is carried out is shown in Figure 39. The obstacle sizes and the location of their centers are in Table 15.

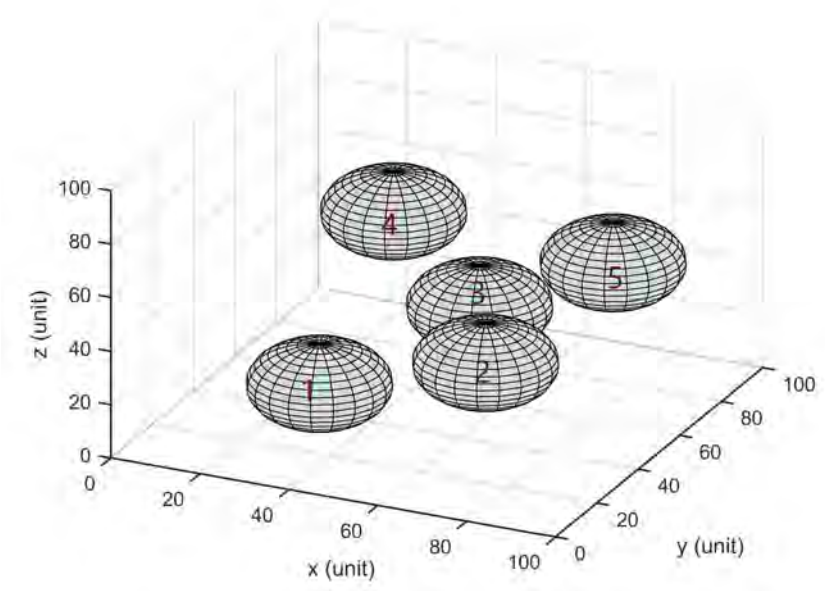


Figure 39: Environment used for 3D static test

Table 15: 3D static test obstacle details

| Obstacle number | Obstacle radius (units) | x-y-z Center Location |
|---|---|---|
| 1 | 15 | (32, 32, 17) |
| 2 | 15 | (67, 37, 32) |
| 3 | 15 | (47, 77, 22) |
| 4 | 15 | (37, 57, 67) |
| 5 | 15 | (77, 77, 47) |

The test was then executed with the following parameters and a typical run for RRT* and RRT*N are shown in Figures 40 and 41.

- 10000 nodes
- Starting point: (0, 0, 0)
- Goal point: (100, 100, 100)
- Step size: 3 units
- Neighborhood size: 7.5 units
- Robot vision range: 5 units
- Normal distribution: 2 $\sigma$ within 5% of $L$

It should be stated that the trials only used 2500 nodes unlike the 10000 used in the actual tests because with the large number of nodes the environment becomes heavily cluttered and it is very difficult to see the actual tree and how the final path looks like especially since it is a 3D environment.

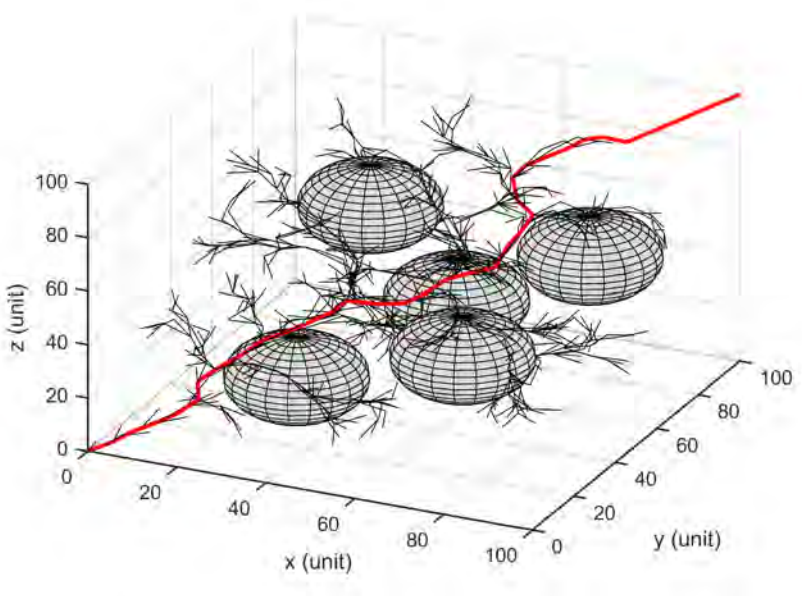


Figure 40: Typical RRT* run in 3D static scenario

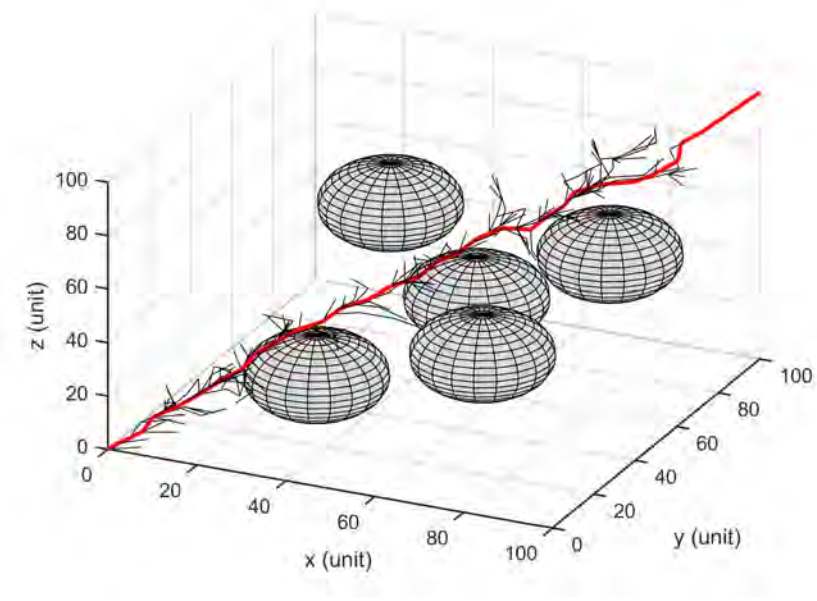


Figure 41: Typical RRT*N run in 3D static scenario

The results obtained are for 100 trials for each algorithm and they are summarized in Table 16.

Table 16: 3D static test results

| **Parameter** | **RRT*** | **RRT*N** |
|---|---|---|
| Average time (sec) | 75.29 | 23.03 |
| Time standard deviation (sec) | 35.02 | 22.84 |
| Average length (units) | 199.06 | 186.79 |
| Length standard deviation (units) | 11.74 | 6.86 |
| Success rate | 59% | 96% |

All 100 obtained paths are found in Figures 42 and 43 to provide prospect into the different behavior of the two different algorithms. In both cases, the blue paths represent a successful navigation, where the robot reached the goal region, before running out of nodes. The red paths, however, represent failed attempts.

An important piece of information to take not of here is that due to the difficulty of having complete visibility of the environment it might not be very clear where the red paths are failing. Another reminder that the failure criteria is not only in case of collision but also if the tree does not manage to reach the goal region.

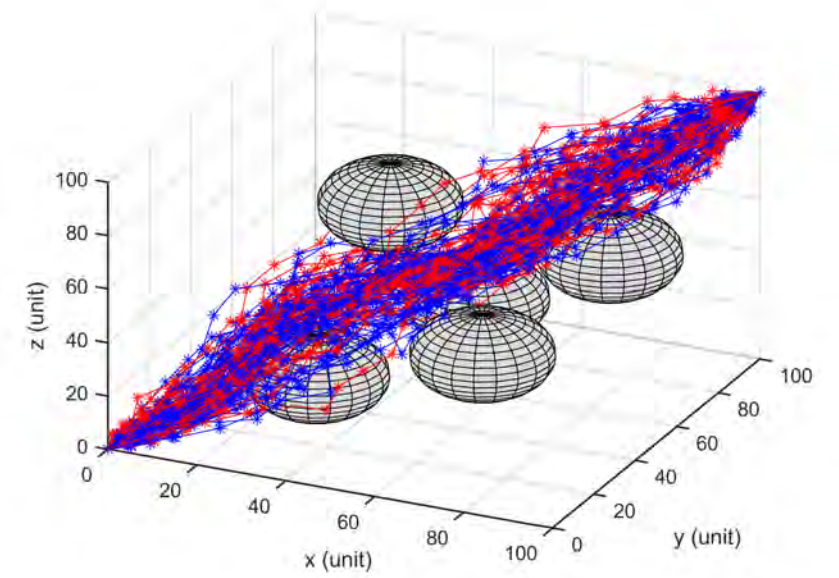


Figure 42: 100 RRT* runs in 3D static scenario

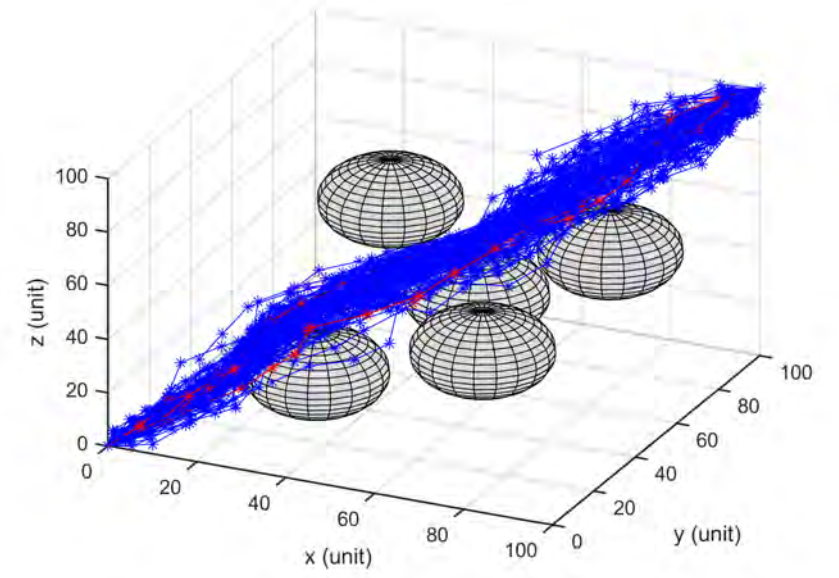


Figure 43: 100 RRT*N runs in 3D static scenario

Apart from the large contrast in the success rate, the paths from the RRT* are more inconsistent and spread out compared to those acquired by RRT*N due to the node region being more compact in RRT*N.

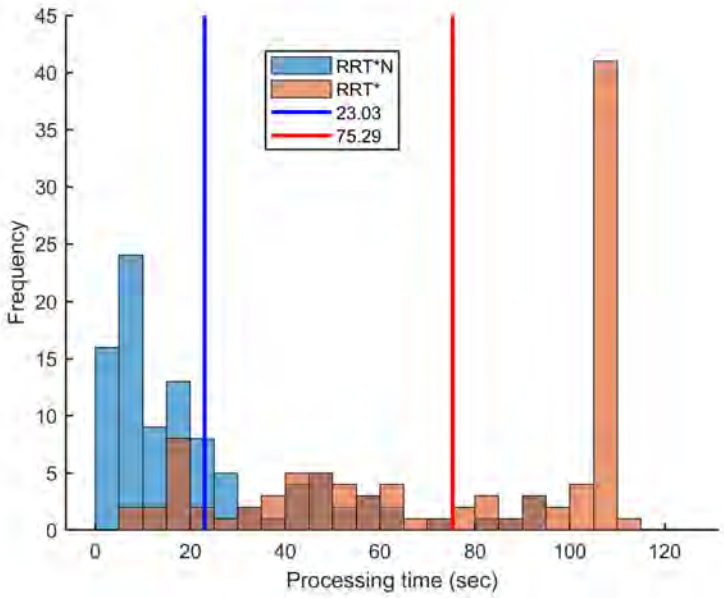


Figure 44: Algorithms CPU times distribution for 250 runs in 3D static scenario

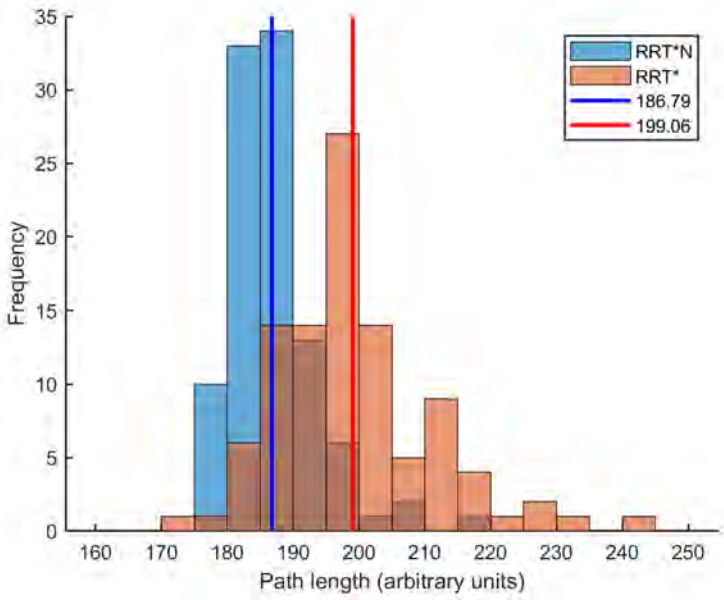


Figure 45: Algorithms path lengths distribution for 250 runs in 3D static scenario

The RRT*N reaches the goal region in a time approximately one third of that needed by the classic algorithm. As shown in Table 16, in static 3D scenarios, RRT*N

is more than three times faster than the RRT*. Moreover, Figures 44 and 45 give more in depth information showing that the time consumption for RRT*N is more consistent with most of the trials succeeding in less than 20 seconds. Also, the path length for the RRT*N is improved as its average value is 93.84% of that produced by RRT*. Another indicator is the success rate, where RRT*N has a 96% success rate compared to 59% for the RRT*, in this scenario.

## 3.3. Dynamic Scenario

Having satisfactorily simulated the algorithm in 2D and 3D static environments and also implemented it in 2D static environment, we will do the same but in a dynamic scenario. The aim again is to improve on the processing times while also providing equivalent paths in terms of length and quality.

**3.3.1. Simulation in 2D.** Again, the first step in the dynamic testing process is to test our new algorithm in 2D simulation and compare its performance to RRT*. Figure 46 shows an environment with a single circular dynamic obstacle that will be attempting to block a direct path from the starting point towards the goal.

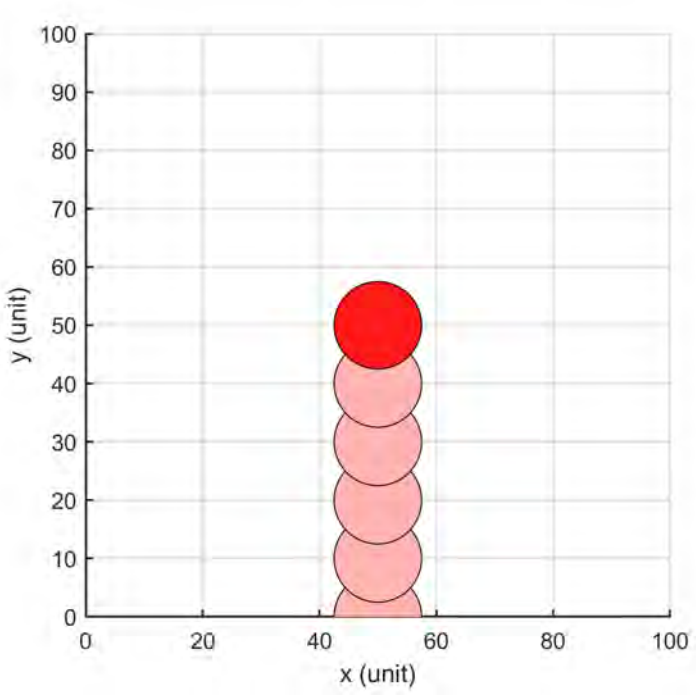


Figure 46: Dynamic obstacle path in 2D

The environment size is 100x100 units with both axes ranging from 0 to 100 units. The circular obstacle is presumed to appear within vision range at the center of the map once the robot gets close enough. The obstacle is circular with radius of 7.5 units and it will come to a stop at the position (50, 50). In subsection 2.2.3 the manner by with which the path is rerouted is discussed.

The test was then executed 100 times with the following parameters and all 100 obtained paths are found in Figures 47 and 48 to provide prospect into the different behavior of the two different algorithms. In both cases, the blue paths represent a successful navigation, where the robot reached the goal region, before running out of nodes. The red paths, however, represent failed attempts.

- 5000 nodes
- Starting point: (0, 0)
- Goal point: (100, 100)
- Step size: 3 units
- Neighborhood size: 7.5 units
- Robot vision range: 5 units
- Normal distribution: 2 $\sigma$ within 10% of $L$

The results obtained are for 100 trials for each algorithm and they are summarized in Table 17.

Table 17: 2D dynamic test results

| **Parameter** | **RRT*** | **RRT*N** |
|---|---|---|
| Average time (sec) | 3.83 | 1.61 |
| Time standard deviation (sec) | 1.77 | 0.69 |
| Average length (units) | 144.81 | 166.29 |
| Length standard deviation (units) | 4.16 | 6.4728 |
| Success rate | 44% | 100% |

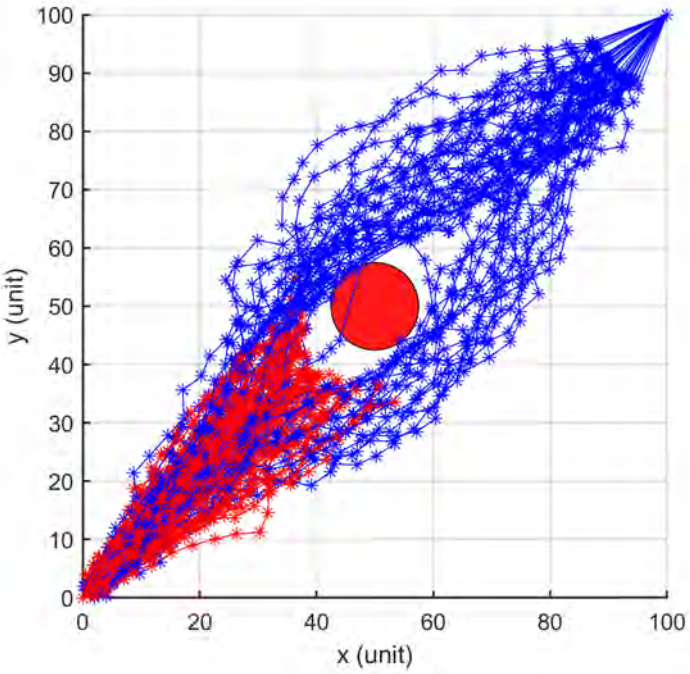


Figure 47: 100 RRT* runs in 2D dynamic scenario

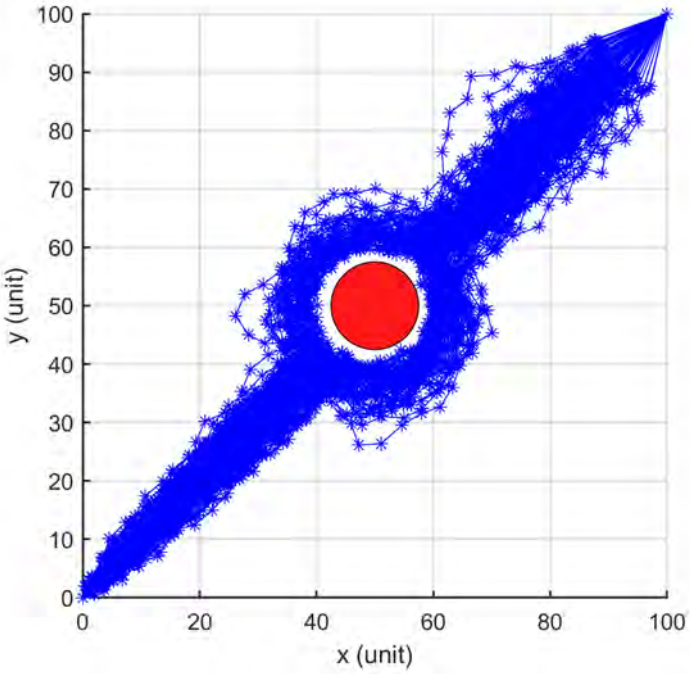


Figure 48: 100 RRT*N runs in 2D dynamic scenario

Figures 49 and 50 show the complete results obtained for the 100 trials of the 2D dynamic test.

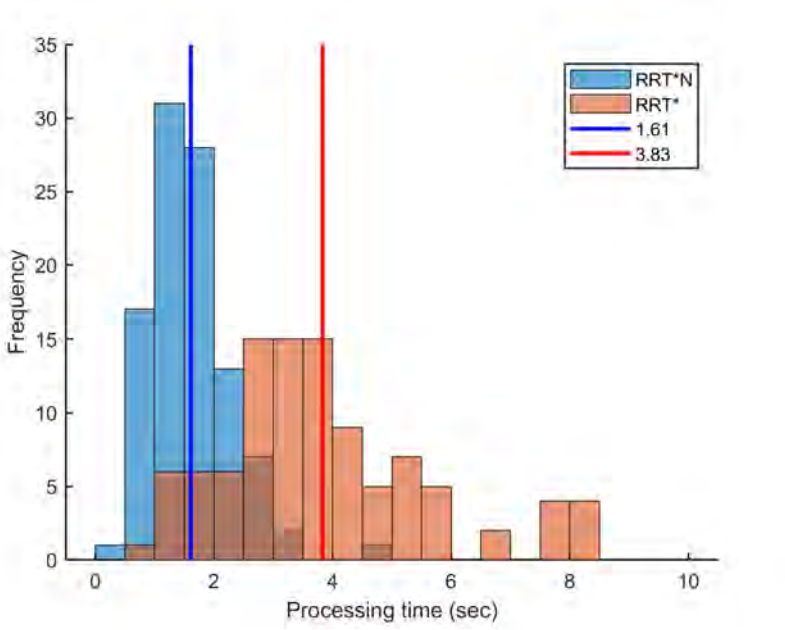


Figure 49: Algorithms CPU times distribution for 100 runs in 2D dynamic scenario

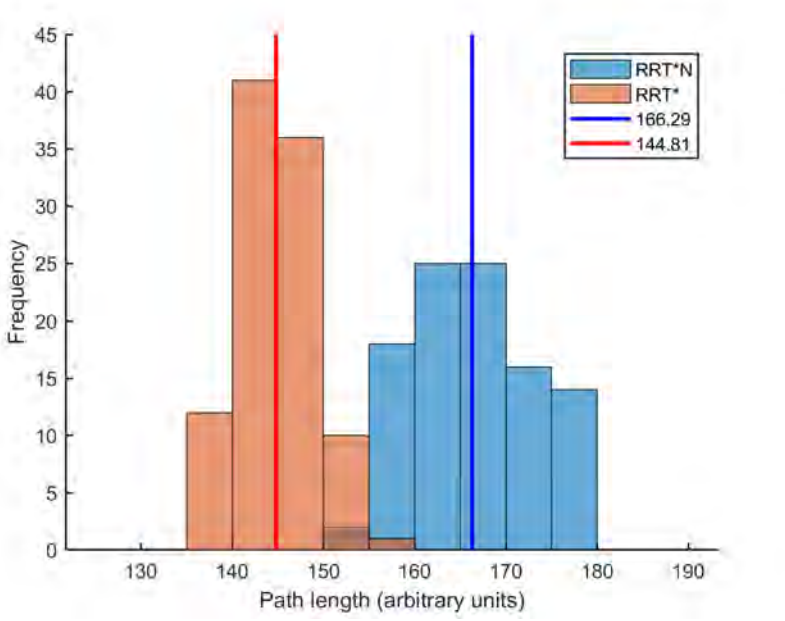


Figure 50: Algorithms path lengths distribution for 100 runs in 2D dynamic scenario

The results given in Table 17 show that RRT*N takes around 40% of the time needed by the classical RRT* in handling dynamic obstacles at the cost of having roughly 15% longer paths. However, the success rate is 100%, which is excellent compared to only 44%, for the case of RRT*. Also, Figures 49 and 50 show the distribution of the different runs for the CPU time and the paths lengths, respectively. Figure 49 shows that RRT*N has a more compact processing time distribution, which is closer to its average value of 1.6 seconds while RRT* has a wider distribution of the performance with some trials taking as long as 8 seconds.

Table 18 summarizes the statistics of the implementation rerouting process done using classic RRT* and RRT*N algorithms. The results obtained are for 100 trials for each algorithm and they are demonstrated in Figure 51.

Table 18: 2D dynamic rerouting implementation results

| **Parameter** | **RRT*** | **RRT*N** |
|---|---|---|
| Average time (sec) | 0.18 | 0.30 |
| Time standard deviation (sec) | 0.08 | 0.44 |
| Success rate | 61% | 100% |

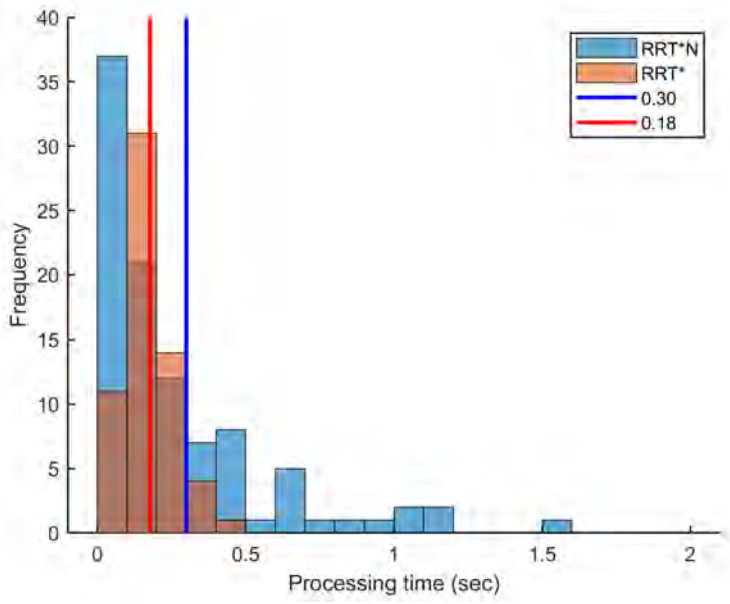


Figure 51: Rerouting CPU times distribution for 100 runs in 2D dynamic scenario

At first glance it would seem that RRT* is superior in the dynamic rerouting implementation for this scenario. However, when considering that the RRT* has only 61% of success rate, compared to a 100% success rate for RRT*N, the above conclusion should be reconsidered. Moreover, the distributions in Figure 51 explain further the superiority of the RRT*N, as close to 90% of the RRT*N simulations took less than 0.5 seconds and almost 50% took less than 0.25 seconds. However, some outliers can explain the higher average value, compared to the RRT*. While in RRT* the average is lower and the distribution is more compact is because almost 40% of the trials failed and are therefore not considered.

**3.3.2. Implementation in 2D.** The dynamic version of RRT*N was tested in 2D using the same scenario as in the static test, but with one dynamic obstacle, which is another Kobuki turtlebot. A critical point of information is that the obstacle moves at a maximum relative speed of 0.2 m/s. The dynamic situation is shown Figure 52.

Figure 52: Real 2D dynamic scenario

The robot was tested several times and the tests were always successful. The static environment information was always provided to construct an initial tree. The

dynamic obstacle is then brought into the picture after the robot has already started moving. The RRT*N variant is quick enough to allow the robot to recalculate its path as soon as the obstacle is within 1 meter of the robot. The robot has a speed that is roughly 0.6 m/s since the algorithm can consistently provide an alternative path and re-stitch it to the original one in less than half a second, otherwise the robot has to stop and wait for the new path to be provided. Figure 53 shows the initial path obtained.

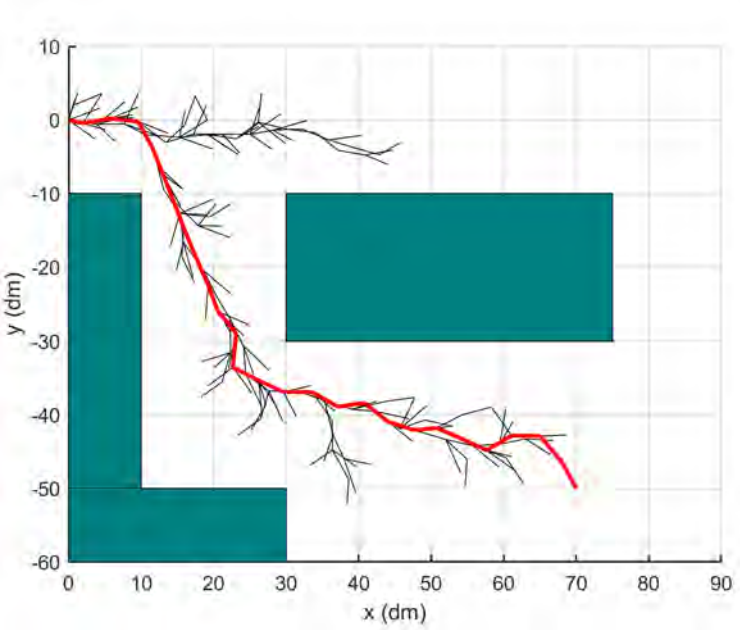


Figure 53: Typical RRT*N path for the provided 2D dynamic scenario

After this initial path is obtained, the dynamic obstacle was moved to obstruct it. Figures 54 and 55 show this scenario and how the path was rerouted as previously explained and discussed in subsection 2.2.4, which is to be utilized as a detour. This rerouting process is triggered once the obstacle is visible to the robot which is basically when it is within 1 m distance of it. As we have demonstrated section 3.2.2 it is much more realistic to test the resultant paths after smoothing. So in this subsection we will only present smoothed paths following scenarios.

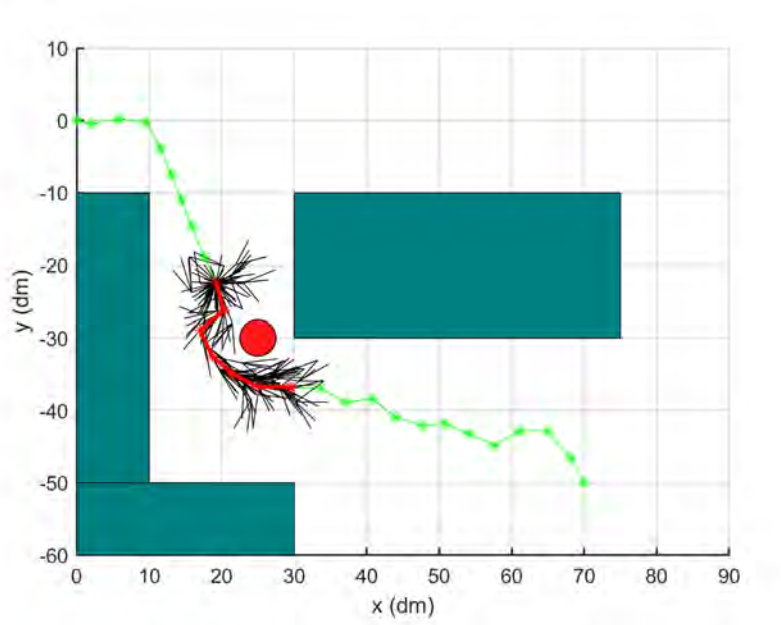


Figure 54: Typical rerouted RRT*N path for the provided 2D dynamic scenario

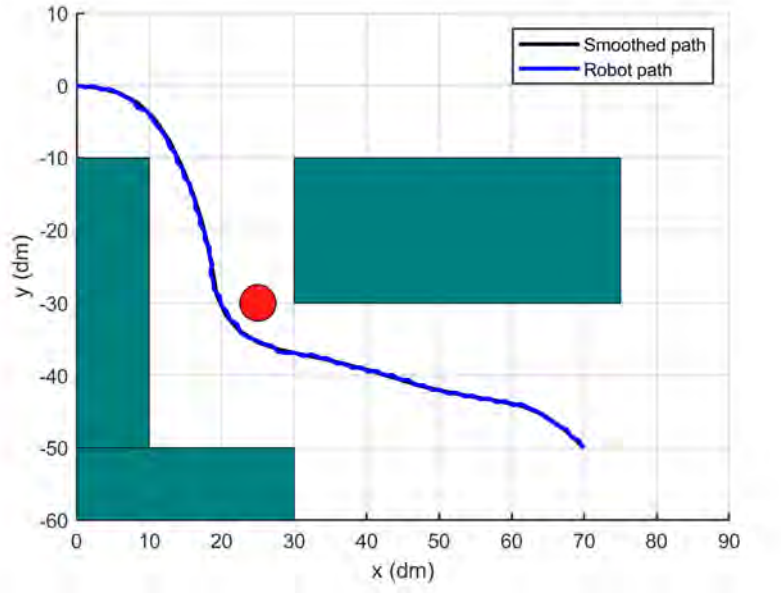


Figure 55: Typical smoothed RRT*N implementation in real 2D dynamic scenario

The speed and rotational speed measurements for the robot in this particular trial are shown in Figures 56 and 57. It is visible that the speed is slightly fluctuating but

still consistently following the specified value. The rotational speed is again fluctuating like before because of the sampling time being 0.2 seconds.

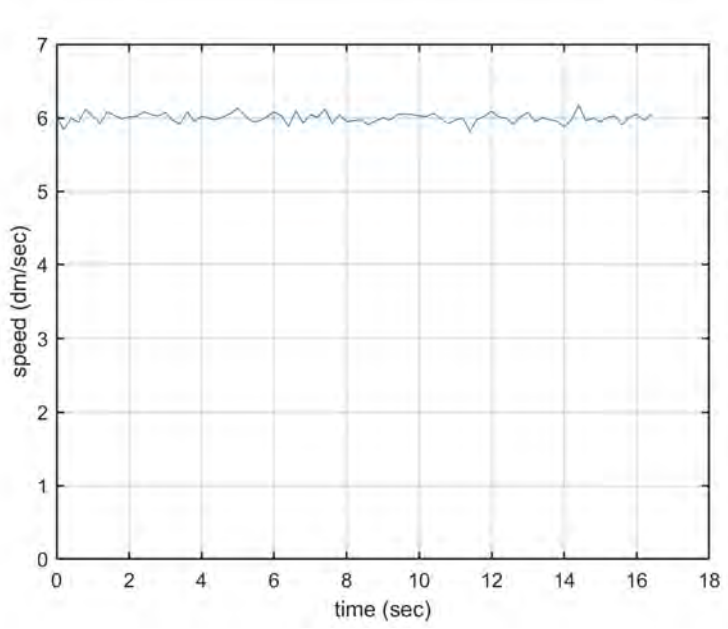


Figure 56: Kobuki smoothed speed in 2D dynamic test

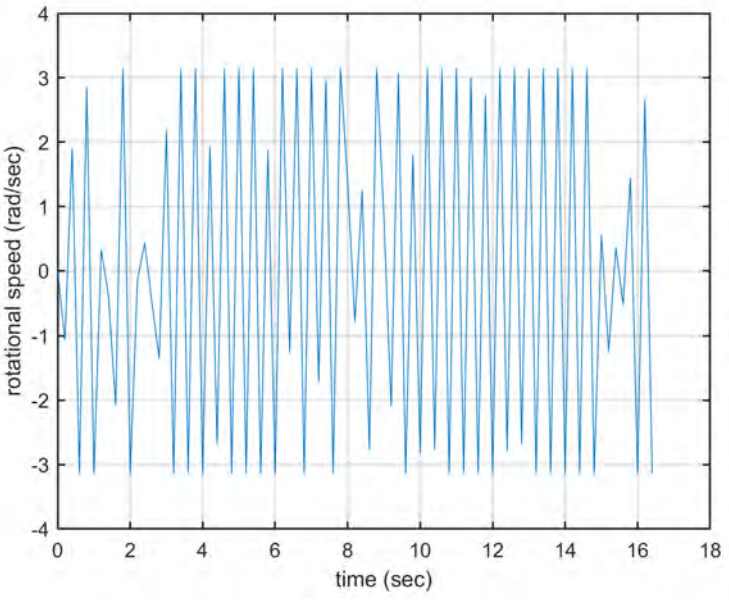


Figure 57: Kobuki smoothed rotational speed in 2D dynamic test

Overall, it is shown that RRT*N is capable of navigating through the presented dynamic environment, with a 100% success rate. However, in some trials, the robot has to stop and wait for the rerouted path because of the long processing times.

**3.3.3. Simulation in 3D.** The final step in the dynamic testing process is to test our new algorithm in a 3D environment and the obtained results are compared with those obtained, in the same environment, from the RRT*. We present an environment with one spherical obstacle blocking a direct path from the starting point towards the goal and have tested both algorithms in this scenario. The environment size is 100x100x100 units with all axes ranging from 0 to 100 units. Again, subsection 2.2.4 contains the figure of the environment where the test is carried along with the details of the approach. The obstacle radius is 15 units and it its center starts from the bottom of the environment at the point (50, 50, 0) and moves up to the center of the environment at the point (50, 50, 50). Figure 58 shows the environment and the dynamic obstacle's direction of motion.

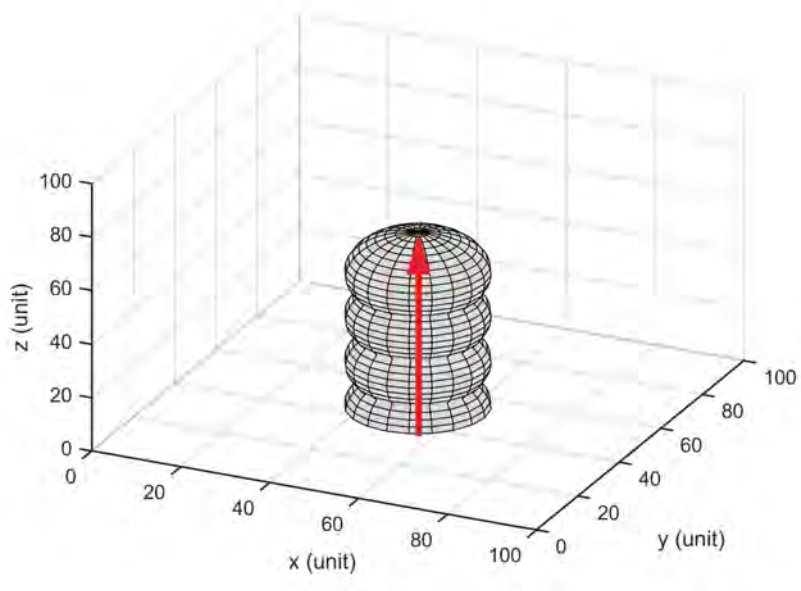


Figure 58: Dynamic obstacle path in 3D

The test was then executed 100 times with the following parameters. All 100 obtained paths to provide prospect into the different behaviors of the two different al-

gorithms are plotted in Figures 59 and 60. In both cases, the blue paths represent a successful navigation, where the robot reached the goal region, before running out of nodes. The red paths, however, represent failed attempts.

- 10000 nodes
- Starting point: (0, 0, 0)
- Goal point: (100, 100, 100)
- Step size: 3 units
- Neighborhood size: 7.5 units
- Robot vision range: 5 units
- Normal distribution: 2 $\sigma$ within 10% of $L$

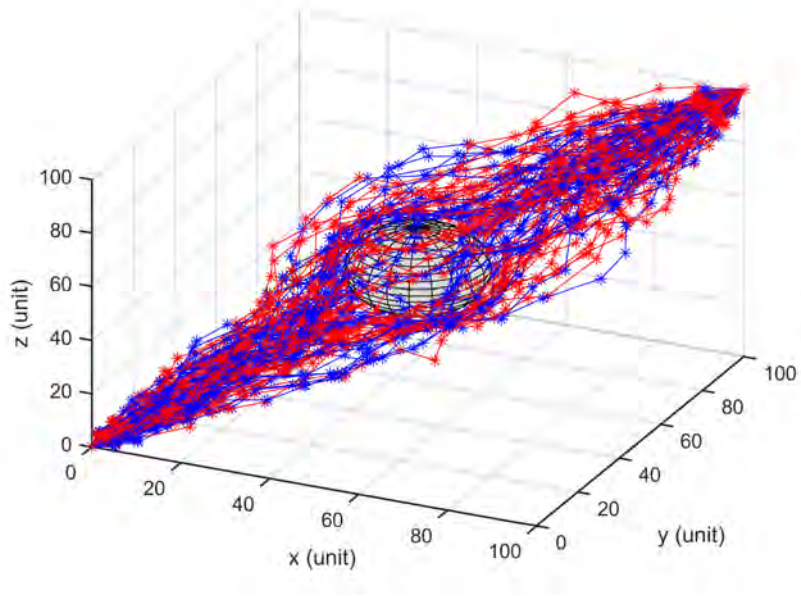


Figure 59: 100 RRT* runs in 3D dynamic scenario

Again, an important piece of information to take not of here is that due to the difficulty of having complete visibility of the environment it might not be very clear where the red paths are failing. Another reminder that the failure criteria is not only in case of collision but also if the tree does not manage to reach the goal region.

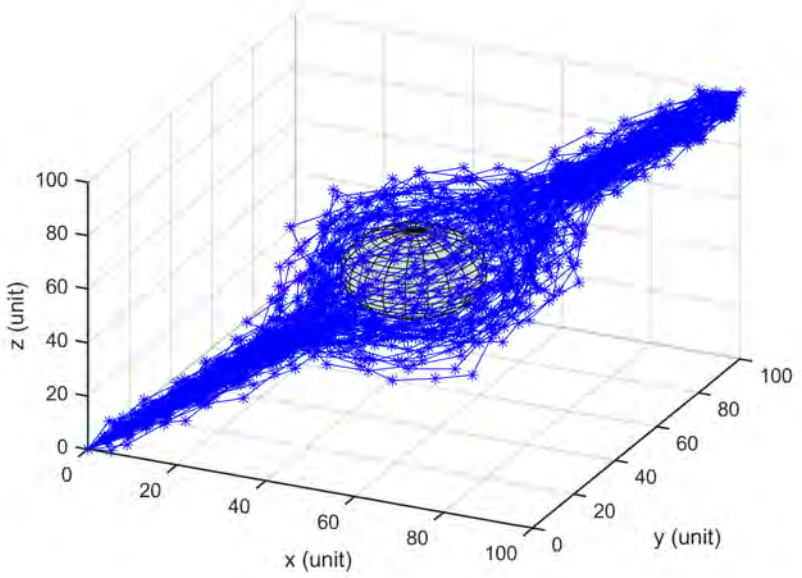


Figure 60: 100 RRT*N runs in 3D dynamic scenario

The results obtained are for 100 trials for each algorithm and they are summarized in Table 19.

Table 19: 3D dynamic test results

| **Parameter** | **RRT*** | **RRT*N** |
|---|---|---|
| Average time (sec) | 61.79 | 2.75 |
| Time standard deviation (sec) | 26.24 | 3.17 |
| Average length (units) | 208.67 | 208.40 |
| Length standard deviation (units) | 7.81 | 9.03 |
| Success rate | 46% | 100% |

Figures 61 and 62 show that in the dynamic 3D scenarios, the RRT*N still significantly improves on the CPU time as it requires, on average of only 4.45% of that required by the RRT*. However, the path lengths are similar for both methods. Moreover, the RRT*N has a 100% success rate, whereas this rate is limited to 46% in the case of the RRT*, for this simulation case.

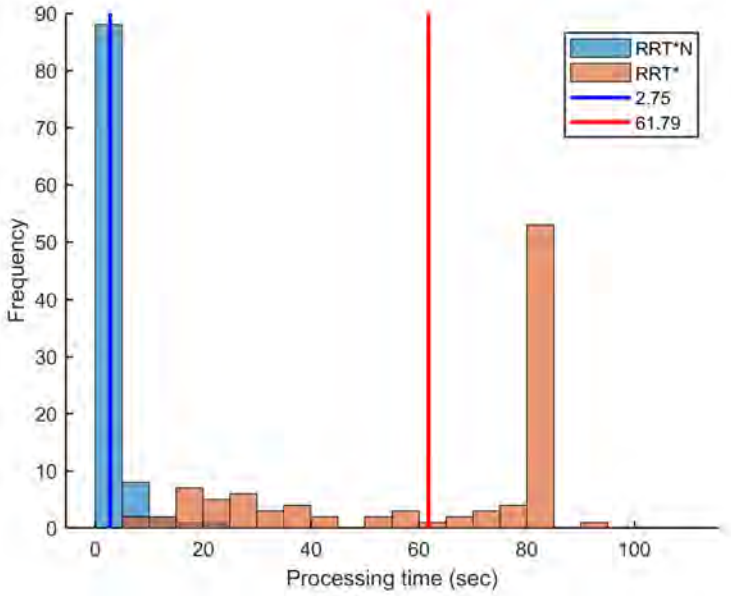


Figure 61: Algorithms CPU times distribution for 100 runs in 3D dynamic scenario

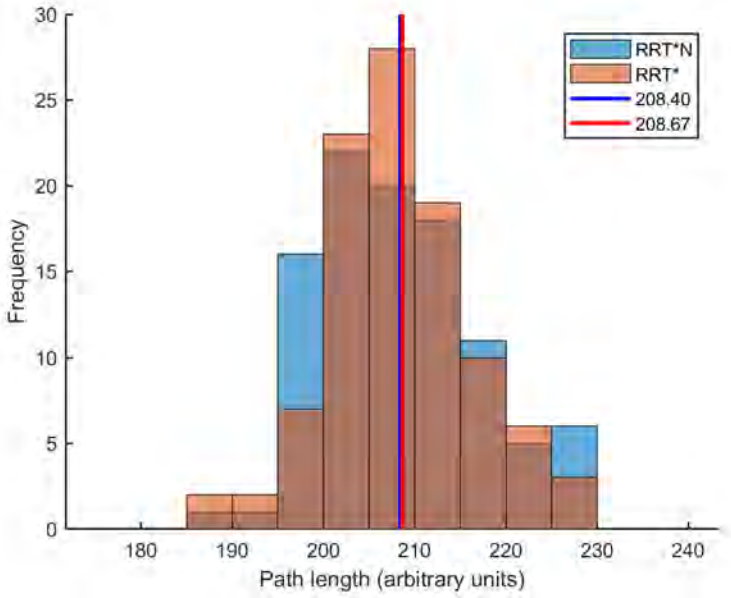


Figure 62: Algorithms path lengths distribution for 100 runs in 3D dynamic scenario

A frame by frame representation of how the rerouting process occurs is shown in Figure 63. The first two frames demonstrate the tree growing but still not being

obstructed by the moving obstacle. By the third frame the path had been generated but soon after the obstacle will come into a position that is obstructing it. The three remaining frames demonstrate the rerouting process and finally in the last frame the end result shows the new path such that the obstacle is now avoided.

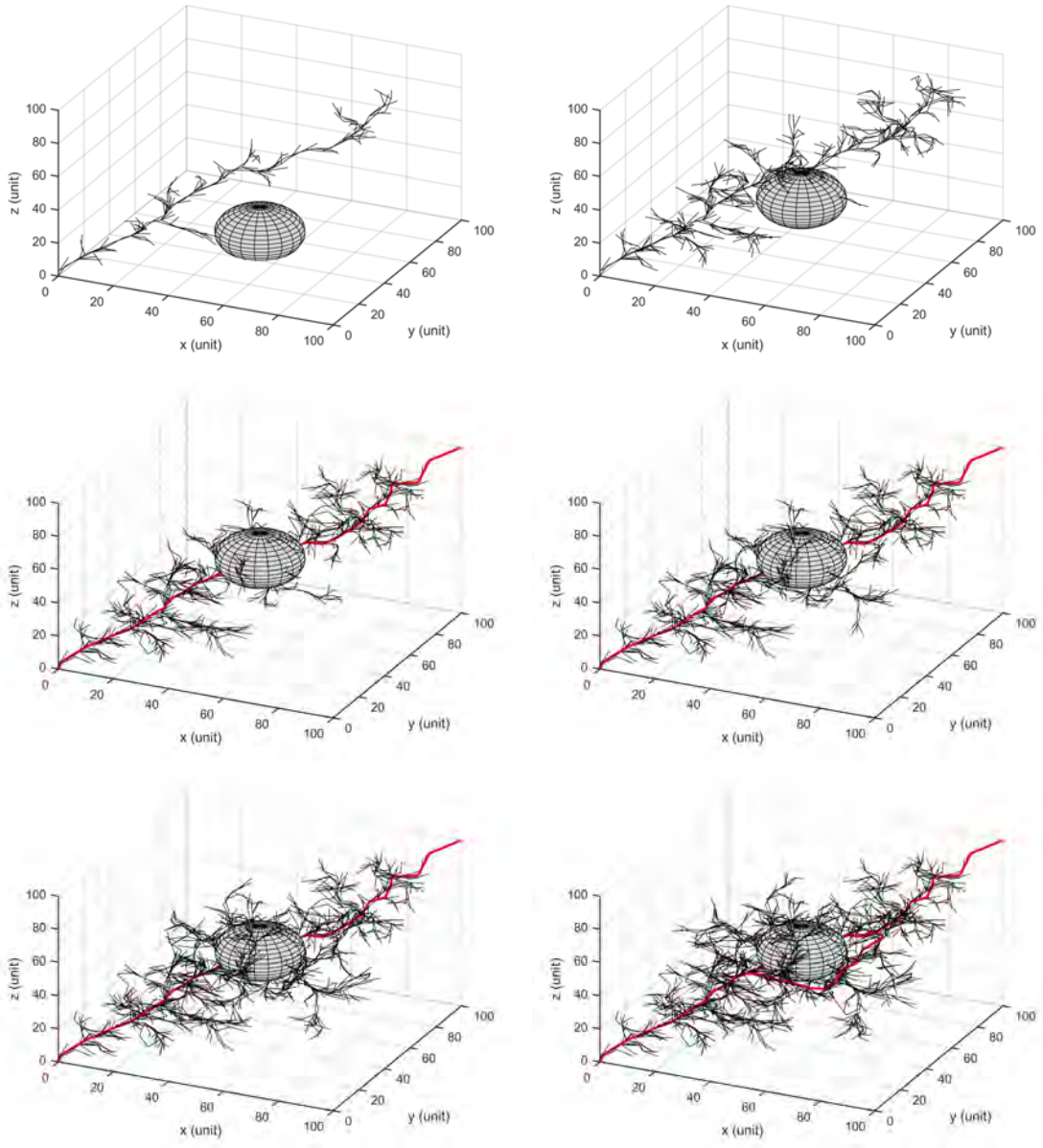


Figure 63: Dynamic rerouting process in 3D

## Conclusion

In this research we tackled the problem of long processing times and unsatisfactory path quality that is present in the Rapidly-Exploring Random Tree Star (RRT*) algorithm for path planning in two dimensional and three dimensional environments both in static and dynamic scenarios. To combat this drawback, our novel approach, the Rapidly-Exploring Random Tree Star Normal (RRT*N) was proposed which addresses these drawbacks by incorporating Gaussian distribution into the process of node generation for RRT*. By altering the probability distribution by which the nodes are generated we directly influence the processing time by a great margin and slightly improving path quality. Further improvement to path quality was achieved by incorporating a filter to the points of the initially acquired path which results in significantly smoother and more feasible paths. Finally, the algorithm was adapted for dynamic situations by incorporating a sub-algorithm that runs in a loop, and this allows the path to get adaptively changed in accordance to the present situation.

All in all, the proposed RRT*N method was shown to perform significantly better than the classic RRT* in terms of success rate and processing time, while maintaining a path quality and having a consistent success rate above 95% in all tests. The speed of path generation is around 3 times faster in static situations, and up to 20 times faster in dynamic situations, compared to the RRT*. Our claim is backed with static and dynamic simulations and testing. This work was accomplished as a result of conducting a comprehensive review of path planning approaches, which led us to select the RRT* method as the basis of the proposed algorithm while also carefully studying the different RRT* variants in our literature review to justify our novel RRT*N method.

Future work is to incorporate Fuzzy Logic into the process of selecting the adequate parameters for algorithm especially for the dynamic scenario as it will allow for greater flexibility in implementation. Also, we aim to switch the platform of work from MATLAB/SIMULINK to ROS as it allows for a greater flexibility in sensor selection and data acquisition which opens up broader fields for work with our method. Finally, the next big step is to implement our method in 3D as well both in static and dynamic scenarios, which will be done utilizing a drone with stereo cameras mounted to enable it to effectively navigate using RRT*N.

## References


[1] C. Laugier and R. Chatila, *Autonomous Navigation in Dynamic Environments*. Springer, 2007.

[2] B. Lau, C. Sprunk, and W. Burgard, "Efficient grid-based spatial representations for robot navigation in dynamic environments," *Robotics and Autonomous Systems*, vol. 61, no. 10, pp. 1116–1130, 2013.

[3] H. Jahanshahi, "Geometric operators for motion planning," Ph.D. dissertation, Cornell University, 2018.

[4] J. Cortes, "Motion planning algorithms for general closed-chain mechanisms," Ph.D. dissertation, Laboratory for Analysis and Architecture of Systems, 2003.

[5] S. Choudhury, "Adaptive motion planning," Ph.D. dissertation, Carnegie Mellon University, 2018.

[6] J. Himmelstein, "Geometric operators for motion planning," Ph.D. dissertation, National Institute of Applied Sciences of Toulouse, 2008.

[7] P. E. Hart, N. J. Nilsson, and B. Raphael, "A formal basis for the heuristic determination of minimum cost paths," *IEEE Transactions on Systems Science and Cybernetics*, vol. 4, no. 2, pp. 100–107, 1968.

[8] W. Zeng and R. L. Church, "Finding shortest paths on real road networks: the case for A*," *International Journal of Geographical Information Science*, vol. 23, no. 4, pp. 531–543, 2009.

[9] A. Stentz, "Optimal and efficient path planning for partially-known environments," in *Proceedings of the 1994 IEEE International Conference on Robotics and Automation*, 1994, pp. 3310–3317.

[10] S. Koenig and M. Likhachev, "Fast replanning for navigation in unknown terrain," *IEEE Transactions on Robotics*, vol. 21, no. 3, pp. 354–363, 2005.

[11] J. Ng and T. Braunl, "Performance comparison of bug navigation algorithms," *Journal of Intelligent and Robotic Systems*, vol. 50, no. 1, pp. 73–84, 2007.

[12] Y. Zhu *et al.*, "A new bug-type navigation algorithm considering practical implementation issues for mobile robots," in *2010 IEEE International Conference on Robotics and Biomimetics*, 2010, pp. 531–536.

[13] T. Asano *et al.*, "Visibility of disjoint polygons," *Algorithmica*, vol. 1, no. 1, pp. 49–63, 1986.

[14] C. W. Warren, "Global path planning using artificial potential fields," in *Proceedings, 1989 International Conference on Robotics and Automation*, 1989, pp. 316–321.

[15] P. Vadakkepat, K. C. Tan, and W. Ming-Liang, "Evolutionary artificial potential fields and their application in real time robot path planning," in *Proceedings of the 2000 Congress on Evolutionary Computation*, 2000, pp. 256–263.

[16] L. E. Kavraki *et al.*, "Probabilistic roadmaps for path planning in high-dimensional configuration spaces," *IEEE Transactions on Robotics and Automation*, vol. 12, no. 4, pp. 566–580, 1996.

[17] R. Geraerts and M. H. Overmars, "A comparative study of probabilistic roadmap planners," in *Algorithmic Foundations of Robotics V*, 2004, pp. 43–57.

[18] S. M. LaValle and J. J. Kuffner, "Rapidly-exploring random trees: Progress and prospects," in *Algorithmic and Computational Robotics: New Directions*, 2000, pp. 293–308.

[19] S. Karaman and E. Frazzoli, "Sampling-based algorithms for optimal motion planning," *The International Journal of Robotics Research*, vol. 30, no. 7, pp. 846–894, 2011.

[20] J. J. Kuffner and S. M. LaValle, "RRT-connect: An efficient approach to single-query path planning," in *Proceedings 2000 ICRA. Millennium Conference. IEEE International Conference on Robotics and Automation. Symposia Proceedings*, vol. 2, 2000, pp. 995–1001.

[21] D. Youakim and P. Ridao, "Motion planning survey for autonomous mobile manipulators underwater manipulator case study," *Robotics and Autonomous Systems*, vol. 107, pp. 20–44, 2018.

[22] S. Karaman *et al.*, "Anytime motion planning using the RRT*," in *2011 IEEE International Conference on Robotics and Automation*, 2011, pp. 1478–1483.

[23] O. Adiyatov and H. A. Varol, "Rapidly-exploring random tree based memory efficient motion planning," in *2013 IEEE International Conference on Mechatronics and Automation*, 2013, pp. 354–359.

[24] H. Gong *et al.*, "A path planning algorithm for unmanned vehicles based on target-oriented rapidly-exploring random tree," in *2017 11th Asian Control Conference (ASCC)*, 2017, pp. 760–765.

[25] J. D. Gammell, S. S. Srinivasa, and T. D. Barfoot, "Informed RRT*: Optimal sampling-based path planning focused via direct sampling of an admissible ellipsoidal heuristic," in *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2014, pp. 2997–3004.

[26] A. H. Qureshi and Y. Ayaz, "Intelligent bidirectional rapidly-exploring random trees for optimal motion planning in complex cluttered environments," *Robotics and Autonomous Systems*, vol. 68, pp. 1–11, 2015.

[27] D. J. Webb and J. van den Berg, "Kinodynamic RRT*: Asymptotically optimal motion planning for robots with linear dynamics," in *2013 IEEE International Conference on Robotics and Automation*, 2013, pp. 5054–5061.

[28] D. J. Webb and J. van den Berg, "Kinodynamic RRT*: Optimal motion planning for systems with linear differential constraints," *arXiv preprint arXiv:1205.5088*, 2012.

[29] D. Lee, H. Song, and D. H. Shim, "Optimal path planning based on spline-RRT* for fixed-wing UAVs operating in three-dimensional environments," in *2014 14th International Conference on Control, Automation and Systems (ICCAS 2014)*, 2014, pp. 835–839.

[30] A. Abbadi *et al.*, "Rapidly-exploring random trees: 3d planning," in *2012 18th International Conference of Soft Computing*, 2012.

[31] J. A. Cobano *et al.*, "Efficient collision-free trajectory planning for WSN data collection with unmanned aerial vehicles," *IFAC Proceedings Volumes*, vol. 46, pp. 220–225, 2013.

[32] Y. Dong, C. Fu, and E. Kayacan, "RRT-based 3d path planning for formation landing of quadrotor UAVs," in *2016 14th International Conference on Control, Automation, Robotics and Vision (ICARCV)*, 2016, pp. 1–6.

[33] P. Pharpatara, B. Hérissé, and Y. Bestaoui, "3-D trajectory planning of aerial vehicles using RRT*," *IEEE Transactions on Control Systems Technology*, vol. 25, no. 3, pp. 1116–1123, 2017.

[34] C. Fulgenzi *et al.*, "Probabilistic navigation in dynamic environment using rapidly-exploring random trees and gaussian processes," in *2008 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2008, pp. 1056–1062.

[35] M. Mohanan and A. Salgoankar, "A survey of robotic motion planning in dynamic environments," *Robotics and Autonomous Systems*, vol. 100, pp. 171–185, 2018.

[36] B. Akgun and M. Stilman, "Sampling heuristics for optimal motion planning in high dimensions," in *2011 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2011, pp. 2640–2645.

[37] J. Nasir *et al.*, "RRT*-smart: A rapid convergence implementation of RRT*," *International Journal of Advanced Robotic Systems*, vol. 10, no. 7, p. 299, 2013.

[38] M. Jordan and A. Perez, "Optimal bidirectional rapidly-exploring random trees," *Computer Science and Artificial Intelligence Laboratory, Massachusetts Institute of Technology, Cambridge, MA, Tech. Rep. MIT-CSAIL-TR-2013-021*, 2013.

[39] O. Arslan and P. Tsiotras, "Use of relaxation methods in sampling-based algorithms for optimal motion planning," in *2013 IEEE International Conference on Robotics and Automation*, 2013, pp. 2421–2428.

[40] C. Moon and W. Chung, "Kinodynamic planner dual-tree RRT (dt-RRT) for two-wheeled mobile robots using the rapidly exploring random tree," *IEEE Transactions on Industrial Electronics*, vol. 62, no. 2, pp. 1080–1090, 2015.

[41] D. Alejo *et al.*, "Efficient trajectory planning for WSN data collection with multiple UAVs," in *Cooperative Robots and Sensor Networks 2015*, 2015, pp. 53–75.

[42] G. Csorvasi, A. Nagy, and D. Kiss, "RTR+C*CS: An effective geometric planner for car-like robots," in *Proceedings of the 2015 16th International Carpathian Control Conference (ICCC)*, 2015, pp. 85–90.

[43] X. Lan and S. D. Cairano, "Continuous curvature path planning for semi-autonomous vehicle maneuvers using RRT," in *2015 European Control Conference (ECC)*, 2015, pp. 2360–2365.

[44] J. Suh, J. Gong, and S. Oh, "Energy-efficient high-dimensional motion planning for humanoids using stochastic optimization," in *2015 IEEE-RAS 15th International Conference on Humanoid Robots (Humanoids)*, 2015, pp. 564–569.

[45] A. H. Qureshi and Y. Ayaz, "Potential functions based sampling heuristic for optimal path planning," *Autonomous Robots*, vol. 40, no. 6, pp. 1079–1093, 2016.

[46] D. Yi, M. A. Goodrich, and K. D. Seppi, "Homotopy-aware RRT*: Toward human-robot topological path-planning," in *2016 11th ACM/IEEE International Conference on Human-Robot Interaction (HRI)*, 2016, pp. 279–286.

[47] D. Devaurs, T. Simeon, and J. Cortes, "Optimal path planning in complex cost spaces with sampling-based algorithms," *IEEE Transactions on Automation Science and Engineering*, vol. 13, no. 2, pp. 415–424, 2016.

[48] S. Choudhury *et al.*, "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)*, 2016, pp. 4207–4214.

[49] S. M. LaValle, *Planning Algorithms*. Cambridge University Press, 2006.

[50] I. Noreen, A. Khan, and Z. Habib, "A comparison of RRT, RRT* and RRT*-smart path planning algorithms," *International Journal of Advanced Computer Science and Applications*, vol. 16, no. 10, pp. 20–27, 2016.

[51] I. Noreen, A. Khan, and Z. Habib, "Optimal path planning using RRT* based approaches: A survey and future directions," *International Journal of Advanced Computer Science and Applications*, vol. 7, no. 11, pp. 97–107, 2016.

[52] R. Siegwart *et al.*, *Introduction to Autonomous Mobile Robots*. MIT press, 2011.

[53] Quanser, *Qbot2 User Manual*, https://www.quanser.com/products/qbot-2-quarc/, Quanser, 2011.

[54] M. A. Al-Omari, M. A. Jaradat, and M. Jarrah, "Integrated simulation platform for indoor quadrotor applications," in *2013 9th International Symposium on Mechatronics and its Applications (ISMA)*, 2013, pp. 1–6.

[55] A. Gibiansky, "Quadcopter dynamics , simulation , and control," http://andrew.gibiansky.com/blog/physics/quadcopter-dynamics/, 2012, accessed May 10, 2017.

[56] A. Bircher *et al.*, "An incremental sampling-based approach to inspection planning: the rapidly exploring random tree of trees," *Robotica*, vol. 35, no. 6, p. 1327–1340, 2017.

[57] Matlab, "Matlab signal processing toolbox," 2017, the MathWorks, Natick, MA, USA.

## Vita

Hussein Ali Mohammed was born in 1994, in Abu Dhabi, United Arab Emirates. He received his primary and secondary education in Abu Dhabi, UAE. He received his B.Sc. degree in Mechanical Engineering from the American University of Sharjah in 2016.

In September 2018, he joined the Mechatronics Engineering master's program in the American University of Sharjah as a graduate teaching assistant. During his master's study, he co-authored one conference paper and two journal papers. His research interests are in autonomous robot navigation, adaptive control, and robotic manipulation.