Acad Year
(18/19)

Project No.
(B090)

DEVELOPMENT OF DYNAMIC PATH REPLANNING ALGORITHM FOR AUTONOMOUS MOBILE ROBOTS

DEVELOPMENT OF DYNAMIC PATH REPLANNING ALGORITHM FOR AUTONOMOUS MOBILE ROBOTS

NANYANG TECHNOLOGICAL UNIVERSITY

HUANG ZHIWEI

U1522408L

SCHOOL OF MECHANICAL AND AEROSPACE ENGINEERING

NANYANG TECHNOLOGICAL UNIVERSITY

Year (2018/2019)

DEVELOPMENT OF DYNAMIC PATH REPLANNING ALGORITHM FOR AUTONOMOUS MOBILE ROBOTS

SUBMITTED

BY

HUANG ZHIWEI

SCHOOL OF MECHANICAL AND AEROSPACE ENGINEERING

A final year project report presented to
Nanyang Technological University
in partial fulfillment of the requirements for the
Degree of Bachelor of Engineering Science (Mechanical Engineering)
Nanyang Technological University

Year (2018/2019)

Abstract

For an autonomous mobile robot to navigate within its environment, it must be able to plan a path from its current location to a desired goal location. Path planning algorithms have been a key concept in developing autonomous robot motion. The existing algorithms with node-based or sampling-based search are capable of generating a collision-free path in a static environment given enough time. However, in real-world situations, the environment of the robot usually consists of one or more moving obstacles. These obstacles may at any point in time obstruct the path of the robot. Whenever the robot encounters any unpredictable moving obstacles, it must dynamically determine the next course of action in order to prevent a collision. This report will cover the development of a dynamic path replanning algorithm for mobile robot navigation in a 3-dimensional configuration space with both static and randomly moving obstacles. A study was done on the existing path planning and replanning algorithms to first develop a basic replanning algorithm in 2-dimensional configuration space. After simulations were done to verify and improve on its effectiveness, the technique will be augmented and implemented in 3-dimensional space. The proposed replanning technique incorporates the use of rapidly exploring random tree (RRT) and its variants. Trajectory estimation of the moving obstacle is a core concept of the replanning algorithm. It allows the robot to alter its existing motion path based on the future position and estimated trajectory of the obstacle. Various simulation results show the effectiveness of the proposed method.

Acknowledgement

The final year project that I have embarked on since the start of Academic Year 2018/2019 has been a very enriching, enlightening and educational experience for me. I have been exposed to the robotics industry, especially in the field of robot motion planning. In addition, I reinforced my MATLAB software capabilities and gained exposure to Robotics Operating System (ROS) and C++ during the span of the project. These valuable learning experiences would not be possible without the guidance and support of a few outstanding individuals and the School of Mechanical and Aerospace Engineering, Nanyang Technological University, Singapore (NTU) as a Final Year Project in Academic Year 2018/2019.

I would like to express my heartfelt gratitude to Professor Low Kin Huat for granting me the invaluable opportunity to work under him and for his kind support, expert opinion and precious feedback throughout the project. I am also highly indebted to all whom I have worked with at the Air Traffic Management Research Institute (ATMRI) for their help and advice. Mr Liu Xin, Research Associate, for his constant professional guidance since the initial phase of the project and Dr Li Boyang, Research Fellow, for his expert advice and support in the remaining phase of the project.

Table of Contents

List of Figures

List of Tables

Chapter 1: Introduction

1.1 Background

Path planning is one of the most popular research topics in the area of robotics. It has a wide variety of application in many other fields including artificial intelligence for video games, virtual reality and autonomous agent simulation [1]. Traditionally, the main focus of a path planning algorithm was to generate a collision-free path between a specified start and end state in a configuration space. The extensive amount of research in path planning has resulted in a growing number of robust path planning algorithms such that it becomes trivial to simply find a collision-free path in a static environment. In recent years, the research focus has evolved to a more advanced form: optimal path planning in the least possible time. This is mainly driven by the society needs towards intelligent automation, especially in industrial robotics and manufacturing automation.

1.2 Motivations

Recent developments in path planning algorithms make it increasingly robust in terms of execution speed and optimality of path. However, majority of path planning algorithms requires a static and fully known environment while generating a path [2]. This makes it difficult for path planning algorithms to be implemented in a dynamic real-world environment where it may be full of unknown or moving obstacles. Hence, there is a need to adapt path planning algorithms by using it skillfully in dynamic replanning techniques to avoid moving obstacles which are common in real-world environments. Many mobile robots are lacking this feature and are thus not deemed to be real-world operational.

Furthermore, it also noted that most path planning algorithm research mainly concentrate only on two-dimensional (2-D) configuration spaces, limiting the movement behaviors of robots only to a planar surface [2]. There is a research gap for dynamic path replanning in three-dimensional (3-D) space which has great potential applications but also faces tremendous difficulty in terms of an additional dimension of complexity, uncertainty and kinematic constraints.

A potential application of 3-D dynamic path replanning would be in the multi-rotor unmanned aerial vehicle (UAV) category, a special class of mobile robot. UAVs need a dynamic 3-D path planning algorithm to handle the complex, unknown and high dimensional environment. The development of an efficient and dynamic replanning algorithm for drones will be a good step to enhance autonomous UAV operational capabilities. This will open huge opportunities for drone usage in many areas such as agriculture, defence, environmental monitoring and protection, public safety and surveillance, emergency services and traffic management [3]. In addition, with the emergence of more smart city concepts that UAVs can play a role in, the demand for UAVs capable of autonomous motion and obstacle avoidance is increasing as more cities are starting to acknowledge the potential of drones.

1.3 Objective and Scope

The objective of this project is to develop an effective and efficient dynamic path replanning algorithm in a 3-D environment with both static and dynamic moving obstacles. The proposed algorithm is based on the trajectory prediction of the moving obstacle as the mobile robot transverse the preplanned path during the path execution phase. Existing path planning algorithms such as Rapidly Random Exploring Trees (RRT) and its optimal variant RRT* will be employed in the replanning algorithm. The 2-D replanning method proposed by Connel and Hung [4] will be thoroughly analysed to search for potential areas that can be improved upon. A modified 2-D replanning algorithm will be developed and simulation will first be done on MATLAB to evaluate its effectiveness. The replanning algorithm will be further augmented to become a novel 3-D replanning algorithm which can be implemented in a 3-D environment. Simulations in a 3-D environment will also be done on MATLAB to verify the robustness of the algorithm and for further iterative improvement.

The scope of this project will include a review of existing path planning algorithms and some of their recently developed dynamic variants. Simulations will present dynamic environments consisting of multiple static obstacles and one randomly moving obstacle. Both the robot and the obstacle are set to move at the same speed and will remain constant throughout the entire run. The obstacle will be moving in a straight-line path between randomised vertices during the path execution phase of the robot. The implementation scope of the 3-D replanning algorithm will be limited to a single mobile robot. Single query sampling-based methods, more specifically RRT or RRT*, will be used in the replanning algorithm.

Chapter 2: Literature Review

A literature review was done to understand existing path planning methods. There is a variety of 3-D path planning algorithms such as sampling-based algorithms, node-based algorithms, optimization-based algorithms, model-based algorithms, bioinspired algorithms and multifusion based algorithms. [2]. Sampling-based and node-based optimal algorithms will be discussed further in detail as they are the most commonly used 3-D path planning algorithm for UAV scenarios [5]. By studying popular path planning methods, the strengths and weaknesses of each algorithm could be identified. This will aid in the selection of an appropriate method to better suit the requirement of a dynamic replanning algorithm in 3-D. The literature review will also briefly cover other research work in the field of dynamic path planning.

2.1 Sampling-Based Algorithms

Sampling-based algorithms are amongst one of the most popular methods for path planning. Sampling-based algorithms include Rapidly Exploring Random Trees (RRT) and many of its improved variants such as RRT* and RRT-connect, and Probabilistic Road Map (PRM) method. The main reason for its popularity is due to its capabilities in high-dimensional spaces and the ability to generate probabilistically complete solutions [6].

2.1.1 RRT Algorithm

RRT method is first proposed by LaValle in 1998 [7]. RRT rapidly searches the configuration space to generate a path connecting the start node and the goal node [2]. RRT constructs a tree using random sampling in search space. The tree starts from an initial node (x_{start}) as its root and expands to find a path towards the goal node (x_{goal}) in every sampling iteration. In each iteration, a new node is randomly sampled from the configuration space, x_{rand} (line 3 of algorithm 1). If the randomly sampled node lies in the obstacle-free region, a nearest node, $x_{near,}$ is searched from the tree (line 4 of algorithm 1). If the random node is accessible to the nearest node within the predefined step size, the tree is expanded by connecting the random node and nearest node. Otherwise, a new node, x_{new}, that lies along the line between the random node and nearest node and within the predefined step size will be used (line 5 of algorithm 1). A collision check process is then run to check if the extension from the new node, x_{new}, to the nearest node, x_{near}, is collision free (line 6 of algorithm 1). Only if the new path is collision free, the tree expands (line 7 of algorithm 1). When the tree successful expanded to the goal node, x_{goal}, or sufficiently close to the goal node, the path is found and returned (lines 8 to 12 of algorithm 1). The RRT expansion process can also end when a predefined time period expires or a fixed number of iterations are executed [6]. Algorithm 1 outlines the basic RRT algorithm and Figure 2-1 shows an illustration of how it works.

Algorithm 1: RRT Algorithm (x_{start}, x_{goal}, step_size, n)

```
1   T ← InitiliseTree($x_{start}$)
2   for i = 1 to i = n do
3     $x_{rand}$ = Sample(i)
4     $x_{near}$ = Nearest(T, $x_{rand}$)
5     $x_{new}$ = Steer($x_{rand}$, $x_{near}$, step_size)
6     if Obstaclefree($x_{new}$, $x_{near}$) then
7       T ← InsertNode($x_{new}$, $x_{near}$, T)
8     if CloseEnough($x_{new}$, $x_{goal}$) then
9       Obstaclefree($x_{new}$, $x_{goal}$)
10      T ← InsertNode($x_{new}$, $x_{goal}$, T)
11      Break
12  return T
```

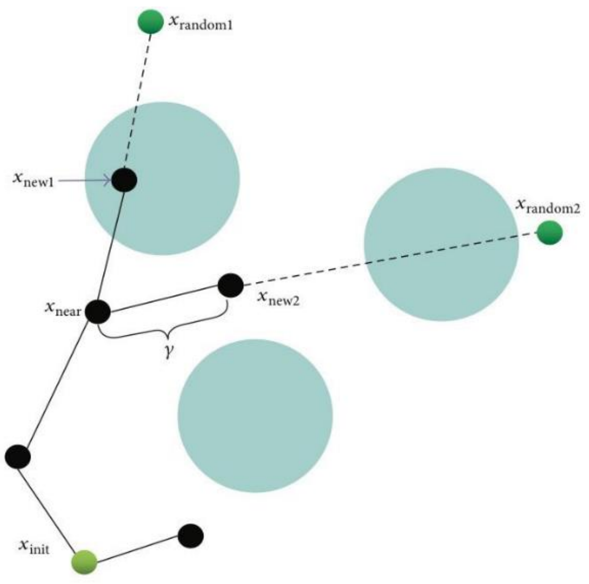


Figure 2-1: Illustration of RRT Algorithm [6]

Some of the major functions mentioned in the RRT Algorithm will be described in further detail.

- *Sample* function generates a random point in the configuration space, x_{rand}.
- *Nearest* function searches the current tree for a node that is nearest to x_{rand} and returns that node, x_{near}.
- *Steer* is a function that attempts to connect x_{near} to $x_{rand,}$ if the incremental distance is less than step_size, x_{new} will be assigned as x_{rand}, else x_{new} is along the path connecting x_{near} to x_{rand} at an incremental distance of step_size from x_{near}.
- *Obstaclefree* function is used to check if the newly formed branch of the tree lies within the obstacle free region.
- *InsertNode* function inserts x_{new} into the existing tree with x_{near} as its parents.
- *CloseEnough* checks if x_{new} is within the step_size to the goal node, x_{goal}.

2.1.2 RRT* Algorithm

RRT* is an improved version of RRT. It was first proposed in late 2010 by Sertec and Emilio [8]. Algorithm 2 below shows that RRT* works similar to RRT with the exception of two new features in the algorithm in lines 7, 8 and 10. These two features are near neighbor search and selection, and rewiring tree operations [9].

Algorithm 2: RRT* Algorithm (x_{start}, x_{goal}, step_size, n)

```
1   T ← InitiliseTree($x_{start}$)
2   for i = 1 to i = n do
3       $x_{rand}$ = Sample(i)
4       $x_{near}$ = Nearest(T, $x_{rand}$)
5       $x_{new}$ = Steer($x_{rand}$, $x_{near}$, step_size)
6       if Obstaclefree($x_{new}$, $x_{near}$) then
7           $X_{nearnb}$ ← Near(T, $x_{new}$, r)
8           $x_{min}$ = Chooseparent($X_{nearnb}$, $x_{new}$, $x_{near}$)
9           T ← InsertNode($x_{min}$, $x_{near}$, T)
10          T ← Rewire(T, $X_{nearnb}$, $x_{min}$, $x_{new}$)
11      if CloseEnough($x_{new}$, $x_{goal}$) then
9           Obstaclefree($x_{new}$, $x_{goal}$)
10          T ← InsertNode($x_{new}$, $x_{goal}$, T)
11          Break
12  return T
```

Unlike RRT, the near neighbor search in RRT* performs a search to find the best parent node for the new node. This search process is done within area or volume enclosed by a specified radius, r. This radius has been defined by $r = \gamma\,(\frac{\log(n)}{n})^{\frac{1}{d}}$, where d is the search space dimension and γ is the planning constant based on the environment [6]. Line 7 of algorithm 2 shows the searching process around the neighborhood of the new node to return a set of suitable nodes, X_{nearnb}. Next, the cost associated with connecting to each of these nodes, including the initial parent node, x_{near}, are examined by the *Chooseparent* function and the node that provides the lowest cost path to reach x_{new} will selected as the parent. The new node, x_{new}, will then be inserted into the RRT* tree with x_{min} as its parent. Figure 2-2 below shows that path 0-1-5-9 has the lowest cost and therefore 5 is selected as the parent of node 9 [10].

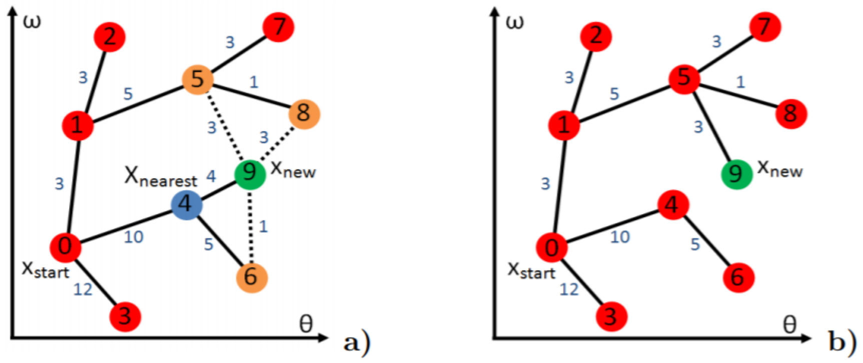


Figure 2-2: (a) Near Neighbor Search, (b) Chooseparent function [10]

The second new feature of RRT*, the rewiring operation rebuilds the tree within the same neighborhood in order to minimize cost to travel between nodes in the tree. In this function, the new node, x_{new}, is now the start point and the neighborhood of nodes, X_{nearnb}, are potential endpoints of a new path. If the path from the new node x_{new} to any of the nearby nodes in X_{nearnb} is obstacle free and the total cost of this new path is lower than the current cost to reach the same nearby node, then the new node x_{new} is a better parent than the current parent of the nearby node. x_{new} is then set as the new parent of the nearby node. In the rewiring process, the edge between the nearby node and its old parent will be removed and a new edge is added between x_{new} and the nearby node. An illustration of the process can be seen in Figure 2-3 below.

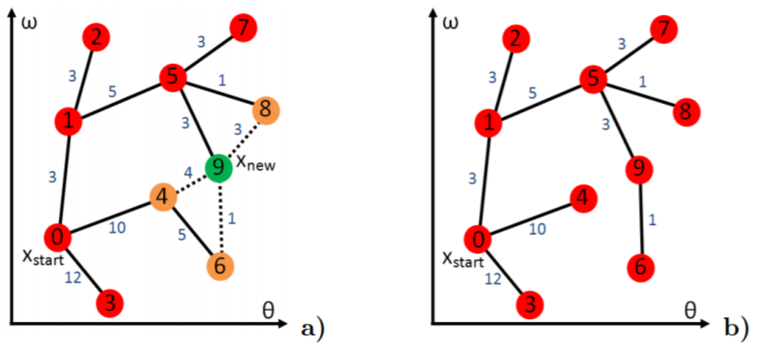


Figure 2-3: (a) Checking path cost for rewiring, (b) Rewired tree [10]

These two additional features, the near neighbor search and selection function, and rewiring tree function alters the way the search tree grows as it explores the environment [9]. It can be seen from Figure 2-4 that with fewer nodes used, there is not much difference between RRT and RRT* but the tree structure and path optimality difference become obvious when a large number of nodes are used. This is an important consideration to be kept in view when applying these algorithms into a dynamic environment where the real-time factor is crucial.

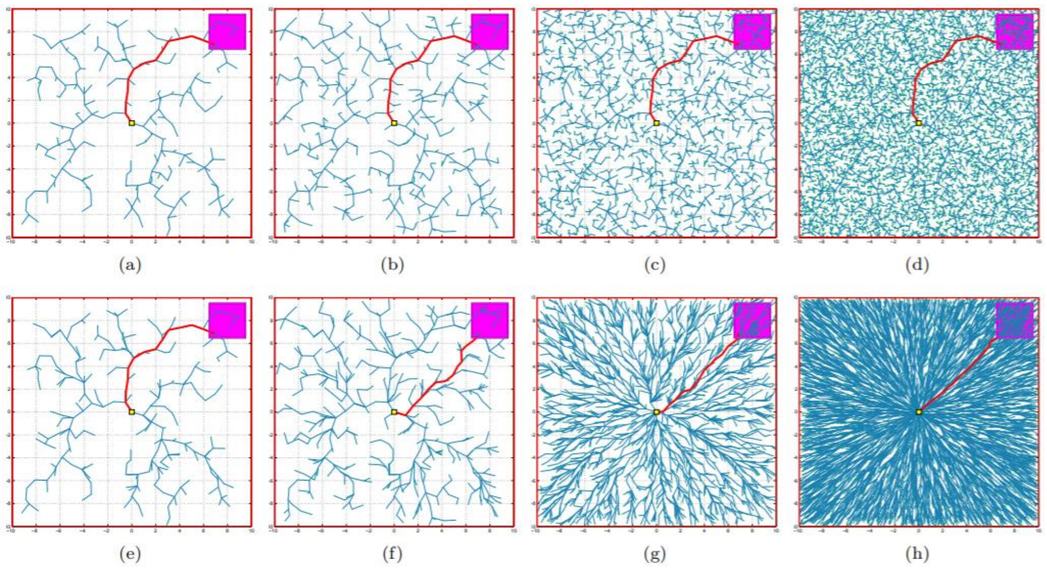


Figure 2-4: Comparison between space exploration and path refinement in RRT (e)-(h) compared to RRT (a)-(d) [9]*

The tree generated by RRT consists of random branches that move in all directions including backwards. On the other hand, in RRT* the *Chooseparent* function ensures that new branches are directed away from the start location [4]. The *Rewire* function optimizes the branches within the tree by removing the occurrence of a zigzag path that may result from RRT due to the randomness in node generation. It helps to smoothen the tree branches in each iteration, removing the jagged and unnecessary steps on the path. Since both functions are optimizing the cost to reach each node within the tree in each iteration, the path found will be asymptotically suboptimal [9].

Similar to RRT, the stopping criteria for RRT* can be user-defined. Common criteria include a time-out, a predefined number of nodes sampled or a predefined number of nodes added to the search tree [11].

2.2 Node-Based Algorithms

Node-based algorithms are one of the oldest path planning techniques. This class of algorithm is also known as graph-based algorithm as it decomposes the configuration space into a grid and each grid point or node corresponds to a vertex in a graph. These algorithms search the graph from a starting node, exploring adjacent nodes until it reaches the destination node. Node-based algorithms include Dijkstra's algorithm [12], A* algorithm [13], D* algorithm [14] and many other improved variants of A* and D*. They are commonly applied in the robotics and video game industry. For example, most algorithms used in the game industry stems from the A* algorithm [15] and graph-based algorithms have been most widely implemented in assembly and disassembly path planning problems in the manufacturing industry [16].

2.2.1 Dijkstra's Algorithm

Dijkstra's algorithm was introduced by Edsger Dijkstra in 1959 to determine the minimum cost path between two nodes [12]. It does so by checking all possible combinations of nodes in the graph starting from the initial node. A brief description of the Dijkstra's algorithm is as shown below.

1. All nodes in the graph are initially placed in an “open” list with infinite cost and the initial node with zero cost is set as the current node.
2. The current node will consider all adjacent neighbours from the open list and calculate the total cost to reach each of the neighbouring nodes from the initial node through the current node. Costs are then assigned to the neighbouring nodes if the newly calculated cost is lower than the previously assigned cost.
3. The current node will then be removed from the “open” list and placed into a “closed” list. Nodes in the closed list will not be reconsidered in future calculations.
4. A new current node with the lowest assigned cost will be selected from the open list and the cycle of checking adjacent neighbours, calculating and assigning costs repeats (steps 2 and 3).
5. The algorithm ends when the destination node is reached or when no adjacent nodes could be found from the open list (no solution). For the first case, the path joining all the nodes will be the optimal collision-free path.

2.2.2 A-Star (A*) Algorithm

A* is an extension of Dijkstra’s algorithm, proposed by Hart, Nilsson and Raphael in 1968 [13]. The overall steps in A* are exactly the same as in Dijkstra’s algorithm except that it uses a heuristic function to calculate the cost of adjacent neighbour nodes. The heuristic approach improves computational efficiency by using some extra information on the graph to give priority to certain nodes [17]. With reference to the steps in section 2.2.1, the main difference between A* and its predecessor is the cost computation in step 2. The new cost computational function is as shown in equation 1,

$$f(n) = g(n) + h(n) \tag{1}$$

where *f(n)* is the estimated minimum total cost of the solution going through node n, *g(n)* is the minimum cost to reach node n found by A* and *h(n)* is the heuristic function that estimates the minimum cost of travel from node n to the goal node. The heuristic function must not overestimate the true cost to get to the goal node for it to be admissible. Most heuristic functions are usually calculated based on the Euclidean distance or Manhattan distance between node n and the goal node [17]. Hence, the heuristic function helps to direct the node expansion towards the goal node and generally improves the path planning time.

2.2.3 D-Star (D*) Algorithm

The D* algorithm is a further extension of A* introduced by Anthony Stentz in 1994 [14]. D* share some resemblance of A* but is capable of planning and replanning when its environment is dynamic. There are two main functions in D*, the process state function and the modify cost function. The process state function works similar to A* but does a backward search from the goal node to the start node to generate a global path. In the modify cost function, when the environment changes or obstacles appear in the path of the robot, it is able to update cost parameters affected nodes based on the new information. This enables D* to generate a new path around the obstacle efficiently while retaining most of its already calculated path in the first function [17].

2.3 Comparison and Evaluation of Algorithms

Both sampling-based and node-based algorithms have their general strengths and limitations and within each category, the different variants have their own unique characteristics as well. Table 2-1 will highlight each algorithm's strengths and weaknesses.

Table 2-1: Strength and Weakness comparison between path planning algorithms

Method	Strengths	Weaknesses
Rapidly Exploring Random Tree (RRT)	1. Relatively simple 2. Probabilistically complete 3. Applicable to high dimensional spaces 4. Fast searching ability due to low time complexity [2]	1. Non-optimal 2. Non-deterministic 3. Inconsistent convergence time 4. Difficulties in narrow environments 5. Static Threat only
RRT*	1. Asymptotic optimality 2. More deterministic than RRT 3. Probabilistically complete 4. Applicable to high dimensional spaces	1. Slightly longer execution time than RRT 2. Inconsistent convergence time 3. Difficulties in narrow environments 4. Static Threat only
A*	1. Complete 2. Optimal 3. Efficient	1. High space complexity 2. Limited to low-dimensional spaces [18] 3. Static Threat only
D*	1. Complete 2. Optimal 3. Efficient 4. Ability to be applied in dynamic environments	1. High space complexity 2. Limited to low-dimensional spaces [18]

Although A* and D* are attractive algorithms that have been widely used in robotics, these methods struggle to cope in high dimensional environment due to the high space-time complexity nature of $O(b^d)$. It becomes computationally expensive in high dimensional spaces due to the combinatorial explosion of nodes in the graph [18] or when fine resolution between nodes is required [17]. Due to difficulties in terms of memory and computation time for complex configurations such as for robots with multiple degrees of freedom, RRT and its variants are commonly used for such applications [19].

As the overall goal of this project is to develop an efficient replanning algorithm for a mobile robot in 3-D, the limitation of A* and D* becomes significant despite some of its advantages like completeness and optimality. There is a natural trade-off between optimality, completeness and computational complexity [20]. Sampling-based algorithms sacrifice completeness and optimality for a much lower computational complexity. For UAV applications, limited payload and real-time constraints would require a low computational complexity. Hence, RRT and RRT* are chosen due to its applicability and efficiency in high dimensional spaces.

2.4 Related Works in Dynamic Path Planning

From the literature review in sections 2.1 – 2.3, most popular path planning algorithms can generate a collision-free path in a known static environment. However, a real-world environment may be unknown and comprise of dynamic obstacles moving in unpredictable directions. This makes it exceptionally difficult for the planning algorithm to take them into account. The path planning algorithm can only work on a snapshot of the environment and plan a collision-free path based on it. The robot has to replan in an event that its preplanned path becomes obstructed by a moving obstacle. In addition, it needs to do in dynamically and in real-time to prevent an impending collision.

One method employed by most dynamic path planning algorithms would be the use of both local and global path planning. Global path planning generates a collision-free path from the start pose to end pose based on the known information of its environment. Global path planning is good as it is able to form a complete path from start to end offline. However, it is usually slow and expensive to be done repeatedly. Global path planning will encounter problems when the entire environment is unknown or only partially known. Local path planning, on the other hand, is a more reactive approach that reviews the updated information collected from its sensors to create a short and localized path to avoid obstacles [21]. Although local path planning is much faster and applicable to environments that change rapidly or consist of moving obstacles, it may get trapped in a local optimal. Most dynamic path planning algorithms adopt an initial global path planning approach followed a localise path planning approach as the environment changes. By retaining the information from the global path planning process and constantly performing localised updates as the environment changes, dynamic path planning algorithms are able to exploit the advantages of both approaches.

Although the basic RRT algorithm can be considered under global path planning, it has a growing number of dynamic path planning variants in recent years. Execution-extended RRT (ERRT) [22] reuses some of the nodes from the first global RRT process to regrow a new search tree when the updated information invalidates the current path. Dynamic RRT (DRRT) [23] is a more efficient variant than ERRT and is the probabilistic version to the deterministic D* family of algorithms. The root of the search tree is located at the goal node so that most of the previous search tree can be reused after trimming away invalidated branches due to the updated information. Regrowing these branches in DRRT will be much faster than regrowing an entire search tree. RRT^X [11] is the first real-time sampling-based variant that constantly refines the path for optimality and repairs the same search tree whenever a change is detected. This refining and repairing of the path occur in real-time while the robot is navigating its path. It is applicable in partially known environments and dynamic environments where obstacle appear, disappear and moves.

While the dynamic path planning algorithms stated above are robust and effective in dynamic environments, they are recently introduced and not widely available. It is challenging to implement DRRT and RRT^X due to the more complex nature of the algorithm. A new dynamic replanning algorithm proposed by Connel and Hung [4] was based on RRT* with some resemblance to ERRT and DRRT. In the algorithm, branches of the search tree in collision with the moving obstacle are invalidated and nodes generated by the initial search are considered to speed up the replanning process. Further elaboration on the algorithm will be done in chapter 3.

Chapter 3: Dynamic Path Replanning Algorithm in 2-D

3.1 Overview

In this chapter, the dynamic replanning algorithm proposed by Connel and Hung [4] for the robot to avoid a random moving obstacle upon detection is being studied in great detail, improved upon and simulated. The replanning algorithm proposed in this chapter models after [4] however some additional steps and concepts were added to improve the algorithm's overall efficiency and effectiveness. The modified replanning algorithm is first shown to be effective in an open area 2-D environment with a randomly moving obstacle through simulation before augmenting it for implementation in a 3-D environment in Chapter 4. The whole algorithm and corresponding functions have been coded in MATLAB and will be further elaborated in the following sections. A simplified flowchart of the entire simulation is depicted in Figure 3-1. The RRT* path planning algorithm on MATLAB used during the replanning process is adopted from Sai Vemprala [24].

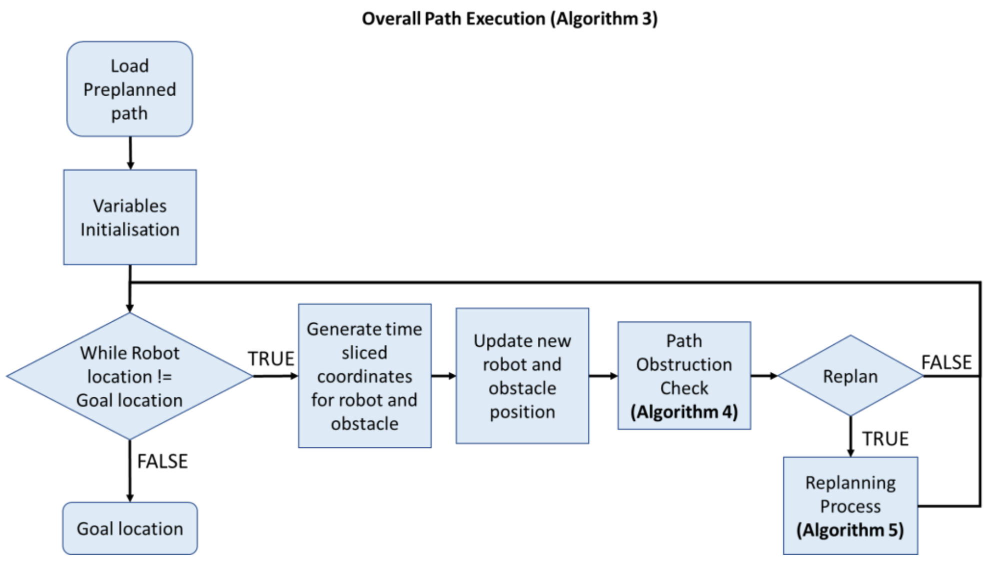

Overall Path Execution (Algorithm 3)
Load Preplanned path
Variables Initialisation
While Robot location != Goal location
TRUE
Generate time sliced coordinates for robot and obstacle
Update new robot and obstacle position
Path Obstruction Check (Algorithm 4)
Replan
FALSE
TRUE
Replanning Process (Algorithm 5)
FALSE
Goal location

Figure 3-1: Flowchart for overall path execution algorithm during simulation

3.2 Path Execution of Robot

To demonstrate the path replanning algorithm, an environment needs to be set up. The robot is given a configuration space with one or more static obstacles and is to build a tree using RRT* to find a path between the specified start point and goal point. Once the path is generated, it is then saved and to be used in all the following simulations as the preplanned path. This is similar to an actual robot planning a path with the data collected from its sensors of the current environment. Only after the initial planning process will the robot execute the optimal path found earlier. All future simulations will also have the same environment for the preplanned path to remain valid.

During the simulation, the optimal preplanned path is first loaded onto the map and the robot will execute it on a frame by frame basis. A randomly moving obstacle is added to the environment during the simulation and has been programmed to randomly cross with the preplanned path. The robot will traverse the preplanned path by travelling from one node to another based on a specified speed set in line 3 of algorithm 3. This process (lines 7 and 9 of algorithm 3) continues until the robot reaches the end of the preplanned path, its goal node. If the robot encounters a random moving obstacle while executing the preplanned path, a replan event will be triggered.

Algorithm 3: PathExecution()

```
1   Load Preplannedpath
2   ObsInitialisation()
3   SetRobotandObsSpeed()
4   VariablesInitilisation()
5   while robotLocation != GOAL do
6     GenerateObsTimeSliceCoord()
7     GenerateRobTimeSliceCoord()
8     UpdateObsLocation()
9     UpdateRobLocation()
10    PathObstructionCheck()
11    if Replan then
12      Replanning()
13    end
14  end
```

The path execution algorithm is the main function, with multiple functions in the inner loop to update the position of the robot and obstacle in each time step, check for path obstruction and replan with RRT* if necessary.

3.3 Random Moving Obstacles and Detection

The random moving obstacle in the simulation forces the robot to dynamically plan around the obstacle using RRT*. For the obstacle to move around in the environment, there needs to be a predefined path for it to follow. In line 2 of algorithm 3, the size and the shape of the obstacle are set together with a list of points which will become the vertices of the path that the obstacles will follow. Upon initialization of the simulation, the obstacle start position will be randomized from the list of vertices. Similar to the robot, the speed of the obstacle is also set in line 3 of algorithm 3. The vertices of the obstacle path are selected such that the robot will have the opportunity to move slightly before first encountering the obstacle. Furthermore, the vertices have been specially selected such that there is a high chance for the robot to encounter the obstacle. Some vertices have been located right on or beside the optimal path itself. When the simulation first begins or whenever the obstacle completed its previous path from one vertex to another, a random vertex is chosen from the list to be its next destination vertex for its path (line 6 of algorithm 3). The obstacle will travel on the path in an uninterrupted manner until it reaches the destination vertex (line 8 of algorithm 3) and the cycle repeats.

Sensors are excluded from the simulation as the objective was to first develop the algorithm to allow the robot to do dynamic replanning. However, in place of sensors, a detection range is placed on the robot. This is almost equivalent to real-world robots with sensors such as Light Detection and Ranging (LIDAR) which can detect both static and moving obstacles. This detection range can be adjusted accordingly in the simulation (line 1 of algorithm 4). When the moving obstacle comes within the range, it must be observed for two timesteps to obtain the relevant information to calculate the direction of the moving obstacle (lines 1 to 3 of algorithm 4) and follow up with a path obstruction check.

3.4 Obstruction Check Based on Trajectory Prediction

The flowchart in Figure 3-2 and algorithm 4 provides more details behind line 10 in algorithm 3. Within two timesteps in the detection range, the coordinates of both the robot and obstacle are stored and will be used as inputs for future calculations. By applying a series of trigonometry concepts on the direction vector from the robot to the moving obstacle and velocity vectors of the robot and obstacle respectively, it can be determined whether the moving obstacle is blocking the path of the robot. In a two-dimensional configuration space, a basic inverse tangent can be used to find the angles of the vectors.

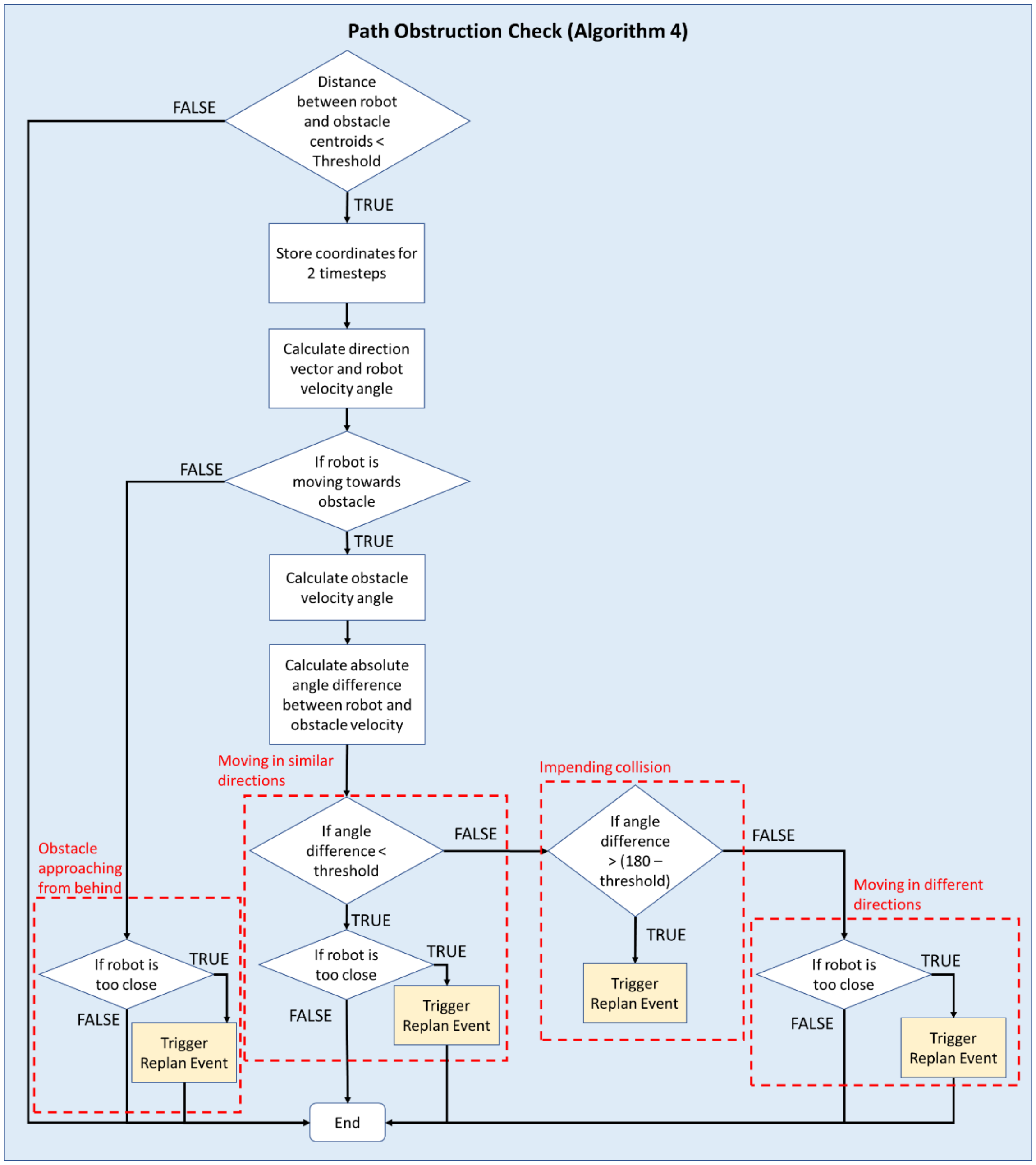

Path Obstruction Check (Algorithm 4)
Distance between robot and obstacle centroids < Threshold
FALSE
TRUE
Store coordinates for 2 timesteps
Calculate direction vector and robot velocity angle
If robot is moving towards obstacle
FALSE
TRUE
Calculate obstacle velocity angle
Calculate absolute angle difference between robot and obstacle velocity
Moving in similar directions
Impending collision
Obstacle approaching from behind
If angle difference < threshold
FALSE
TRUE
If angle difference > (180 – threshold)
FALSE
TRUE
Moving in different directions
If robot is too close
TRUE
FALSE
Trigger Replan Event
If robot is too close
TRUE
FALSE
Trigger Replan Event
Trigger Replan Event
If robot is too close
TRUE
FALSE
Trigger Replan Event
End

Figure 3-2: Flowchart of path obstruction check algorithm

Algorithm 4: PathObstructionCheck()

```
1  if pdist < threshold then
2     StoreCoordinates()
3     timestep++
4     if timetep == 2 then
5        DirectionVectortoObs = atan2(($Y_{obs}$ – $Y_{rob}$), ($X_{obs}$-$X_{rob}$)
6        RobotVelocityVector = atan($V_y$, $V_x$)
7        if PathBlocked() then
8           ObsVelocityVector = atan($V_y$, $V_x$)
9           VelocityAngleDifference = | RobotVelocityVector – ObsVelocityVector |
10          if SimliarDirectionCheck() then
11             if Tooclose() then
12                Replan == TRUE
13             end
14          elseif CheckforImpendingCollision() then
15             Replan == TRUE
16          elseif Tooclose() then
17             Replan == TRUE
18          end
19       elseif Tooclose() then
20          Replan == TRUE
21       end
22       timestep = 0
23    end
24 end
```

The angle of the direction vector from the robot to the moving obstacle calculated (line 4 of algorithm 4) according to the equation below with an illustration in Figure 3-3,

$$angle_{direction\,vector} = atan2((Y_{obs} - Y_{robot}), (X_{obs} - X_{robot})) \qquad (1)$$

where (X_{robot}, Y_{robot}) is the position of the robot and (X_{obs}, Y_{obs}) is the estimated position of the obstacle. The angle of the direction vector will lie between the range of $(-\frac{\pi}{2}, \frac{\pi}{2})$.

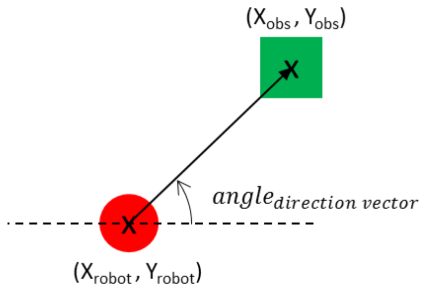


Figure 3-3: Angle of direction vector from robot to obstacle

To determine if the robot is moving towards the obstacle, the angle of the robot velocity vector needs to be calculated (line 5 of algorithm 4). The equation is as follows,

$$angle_{Velocity\_robot} = atan2(V_y , V_x) = atan2((Y_{robot2} - Y_{robot1}), (X_{robot2} - X_{robot1})) \qquad (2)$$

V$_x$ and V$_y$ are the X and Y component of the velocity vector respectively and can be derived from the change in Y position and change in X position within the 2 timesteps. If the absolute difference between the angle of the direction vector from the robot to the moving obstacle and angle of the robot velocity vector is less than some specified angle threshold (equation 3), it can be concluded that the robot is moving towards the dynamic obstacle (line 6 of algorithm 4). Figure 3-4 is an illustration consisting of different scenarios to show how the robot determines if it is moving towards the obstacle.

$$\left| angle_{direction\ vector} - angle_{Velocity_{robot}} \right| < angle_{threshold} \qquad (3)$$

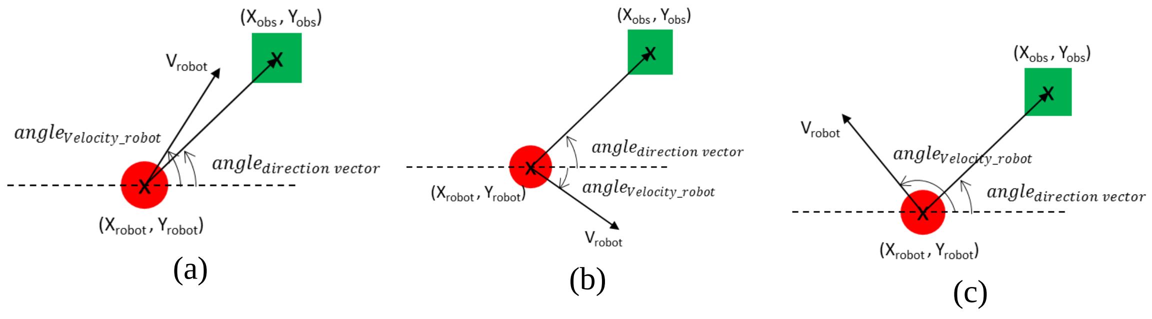
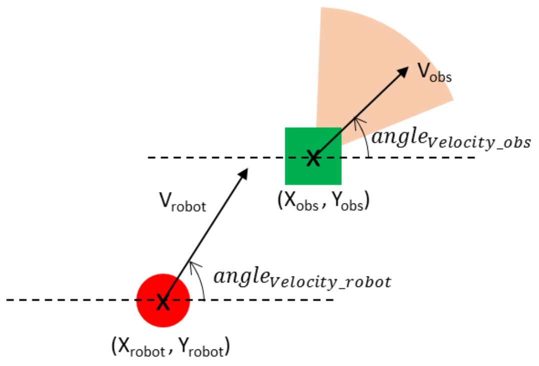


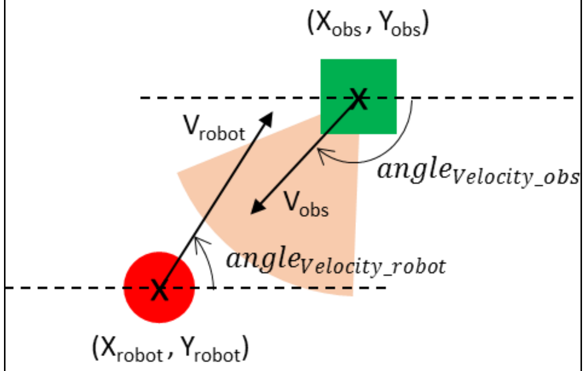
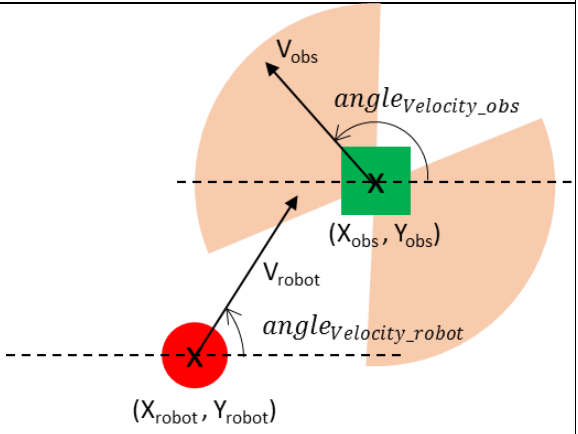
Figure 3-4: (a) Robot moving towards obstacle, (b) & (c) Robot not moving towards obstacle

After the robot is deemed to be moving towards the moving obstacle, there are three possible scenarios which require different courses of action. To identify which scenario it is, the velocity vectors of the robot and obstacles must first be compared. The angle of the obstacle velocity vector can first be obtained by using equation (2) (line 7 of algorithm 4). The absolute angle difference between the robot and obstacle velocity vectors (line 8 of algorithm 4) is calculated as such,

$$angle_{velocity\ difference} = \left|angle_{Velocity\_robot} - angle_{Velocity\_obs}\right| \qquad (4)$$

First, if the result from (4) is less than the angle threshold, the robot and obstacle are moving in a similar direction. Second, if the result from (4) is greater than (180 – angle$_{threshold}$), the robot and obstacle are estimated to be moving towards each other. Third, if the result from (4) does not lie within the ranges in the first and second scenarios, the robot is moving towards the obstacle but the robot and moving obstacle are moving in different directions. Illustrations of the 3 scenarios are shown below in Figures 3-5 – 3-7.

Scenario 1: Robot and obstacle are moving in a similar direction. This occurs when the absolute angle difference between the velocity vectors is less than the angle threshold. The orange shaded region in Figure 3-5 represents the range of obstacle velocity vectors for this scenario.	 V_{obs} $angle_{Velocity\_obs}$ V_{robot} (X_{obs}, Y_{obs}) $angle_{Velocity\_robot}$ (X_{robot}, Y_{robot}) Figure 3-5: Scenario 1, robot and obstacle are moving in similar directions

Scenario 2: Robot and obstacle are moving towards each other. This happens when the absolute angle difference between the velocity vectors is more than 180 degrees minus the angle threshold. The orange shaded region in Figure 3-6 represents the range of obstacle velocity vectors for this scenario.	 (X_{obs}, Y_{obs}) V_{robot} $angle_{Velocity\_obs}$ V_{obs} $angle_{Velocity\_robot}$ (X_{robot}, Y_{robot}) Figure 3-6: Scenario 2, robot and obstacle are moving towards each other
Scenario 3: The robot is moving towards the obstacle but the robot and obstacle are moving in different directions. The orange shaded region in Figure 3-7 represents the range of obstacle velocity vectors for this scenario.	 V_{obs} $angle_{Velocity\_obs}$ (X_{obs}, Y_{obs}) V_{robot} $angle_{Velocity\_robot}$ (X_{robot}, Y_{robot}) Figure 3-7: Scenario 3, robot and obstacle are moving in different directions

In the first scenario, since the robot and obstacles are moving in similar directions, the robot will ignore the obstacle and continue following its preplanned path unless it gets too close. If the proximity of the robot and obstacle gets too close while the robot is continuing its path, a replan event is triggered (line 12 and 13 of algorithm 4). However, if the obstacle or path of the robot changes direction, the robot will have to recalculate accordingly and choose a course of action based on any of the other two scenarios. For the second scenario, it is clear that the robot and obstacle are moving towards each other and there will be an impending collision if nothing is done. The robot immediately triggers a replan event to find another path to avoid the collision (line 9 and 10 of algorithm 4). The course of action for the robot in the final scenario is analogous to the first scenario. Since the obstacle is moving in a different direction from the robot, the robot will ignore the obstacle and continue its path unless it gets too close. A replan event with be triggered if the proximity of the robot and obstacle is too close (line 15 and 16 of algorithm 4).

One major limitation in the algorithm proposed by Devin Connell and Hung Manh La in [4] is the failure to account for the situation where the obstacle is coming towards the robot from behind. In this case, the robot will not be deemed to be moving towards the obstacle and no replan event will be triggered even as the obstacle comes into the detection range set. In Figure 3-8, the obstacle is seen to be approaching the obstacle from behind. However, using the result from equation (1) and (2) as inputs, it can be shown that the result of equation (3) is false. The robot is deemed to be not moving towards the obstacle and no further checks will be done, skipping lines 7 – 18 of algorithm 4. Despite the obvious impending collision, no replan event will occur.

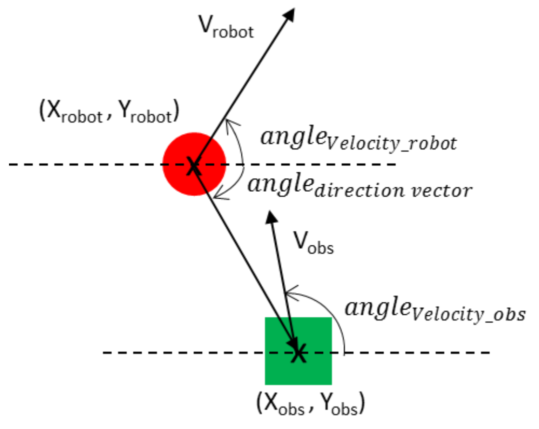


Figure 3-8: Obstacle approaching robot from behind

To mitigate this issue where the obstacle approaches the robot from behind, lines 18 - 20 of algorithm 4 are added. This sets a specified boundary around the robot when equation (3) is false, the robot is not deemed to be moving towards the obstacle. The boundary acts as a safety net and will trigger a replan event when the obstacle comes too close to the robot from behind.

3.5 Path Replanning and Rewiring Operations

Algorithm 5 below provides more details behind line 12 in algorithm 3. When a replan event is triggered in the section above, the steps in algorithm 5 will run. Otherwise, the robot will skip this algorithm and continue on its preplanned path.

Algorithm 5: Replanning()	
1	TrajectoryPredictionMultiplier()
2	PredictedObsPosition()
3	InvalidateNodes()
4	SelectReplanGoalLocation()
5	SetReplanSamplingLimits()
6	RRT*(RobotCurrentLocation, GoalLocation, PredictedObsPosition)
7	RewireReplannedPath()

The algorithm proposed by Devin Connell and Hung Manh La did not attempt to predict the future position of the obstacle for RRT* replanning. While the algorithm was tested to be effective in a 2-D complex maze environment with moving obstacles, this is likely due to the limited space in the maze environment which forces the replan path to take another route in the maze. However, when implemented into an open space 2-D configuration, observations from simulations showed that it is not effective in avoiding the moving obstacle. This is likely due to the robot replanning based on the current data which will lose its validity as the obstacle and robot moves. With less restriction on space, the number of possible replan paths tremendously. The time dimension must be considered to remove some of the possible replan paths that loses validity with time. The steps in algorithm 5 has been modified to account for obstacle trajectory prediction and will differ from the one proposed by Devin Connell and Hung Manh La.

In this section, two new methods to increase the effectiveness of avoiding the moving obstacle will be introduced. First, a multiplier to predict the future position of the obstacle based on its current trajectory (lines 1 and 2 of algorithm 5). The next method is to artificially enlarge the size of the obstacle to result in a more deliberate effort to replan a path that avoids the surrounding area around the obstacle.

The first step in the replan event is to estimate an obstacle trajectory prediction multiplier (line 1 of algorithm 5). This multiplier should be proportional to the current distance between the obstacle and the robot. The further the obstacle is from the robot, the larger the multiplier and vice visa. It is assumed that the obstacle will continue in the same straight-line motion path. This enables trajectory prediction of the obstacle, allowing the robot to calculate the predicted future location of the moving obstacle with the multiplier (line 2 of algorithm 5). By adding the product of the multiplier and obstacle velocity vector to the obstacle's current position, the predicted obstacle location can be found. This is similar to the kinematic concept of calculating the final displacement of a moving object using the equation $s_2 = s_1 + u*t$. The equation is as follows,

$$Predicted\ obstacle\ coordinates = (X_{obs}, Y_{obs}) + multiplier * V_{obs} \qquad (5)$$

Using the predicted obstacle coordinates, any nodes that are within a specified range around it are considered to be in collision with the obstacle and are invalidated (line 3 of algorithm 5).

In the subsequent steps (line 4 and 5 of algorithm 5), an assumption is made to speed up the replanning process by RRT*. It is assumed the preplanned path is the most optimal path leading to the goal location and the robot should return to this path as soon as it is done avoiding the moving obstacle. Hence, with the start node being the current location of the robot upon triggering the replan event, the replan goal node is to be selected from the preplanned path just beyond the obstacle rather than the goal location itself. It is done to reduce the size of the configuration space for localized RRT* planning. This will help to keep the replanning time as short as possible such that the robot can react in real-time to the moving obstacle with minimal delay.

The selected replan goal node must satisfy three criteria. First, it has to be more than 2 nodes away from the current robot position. It must be possible to find 2 nodes further down the path else this criterion is invalid. Next, the replan goal node selected must be more than a specified distance from the current position for an effective replan path. Finally, the selected replan goal node along the preplanned path must be beyond the invalidation range from the obstacle. As long as any of these three criteria are not satisfied, the next node along the preplanned path will be used as the replan goal node. The process continues until the goal location is selected as the replan goal node.

Before running RRT* for the localized replanning, the configuration area must be established for the random sampling in RRT* (line 5 of algorithm 5). While a smaller area will result in a faster path planning time, it may also result in no feasible replan path. The limits of the configuration area have been set based on start and goal point with some specified allowance to ensure sufficient space is provided for the replan path. The RRT* path planning algorithm will run with the set sampling limits together with the start and node goal and the predicted future position of the obstacle as inputs (line 6 of algorithm 5). To account for prediction error and the size of the robot, the size of the obstacle has been artificially increased during the RRT* process such that the replan path generated will avoid the area around the obstacle as well. Finally, once the replan path is generated, it will need to be rewired to the preplanned path (line 7 of algorithm 5). The robot will then travel on this new replan path which avoids the moving obstacle and eventually links back to the initial preplanned path (algorithm 3). If the robot encounters another obstacle and determines that another replan is required, the replan process will be repeated. The replan goal location must always follow the 3 criteria described above.

3.6 MATLAB Simulation in 2-D

Based on the algorithm described from sections 3.2 to 3.5, a MATLAB program has been written to simulate and evaluate its effectiveness. Simulations are conducted using a laptop with Intel i7 2.4GHz processor and 8GB RAM.

3.6.1 Implementation Details

A two-dimensional configuration space of 550 x 550 pixels with a static square obstacle of 100 pixels with its bottom right corner at (350, 150) is created. The start node is set to be at (0, 0) and the goal location at (505, 505). Using the RRT* path planning algorithm MATLAB program adopted from Sai Vemprala [24], a preplanned path has been generated in red in Figure 3-9. The preplanned path is then saved as a file to be used for future simulations.

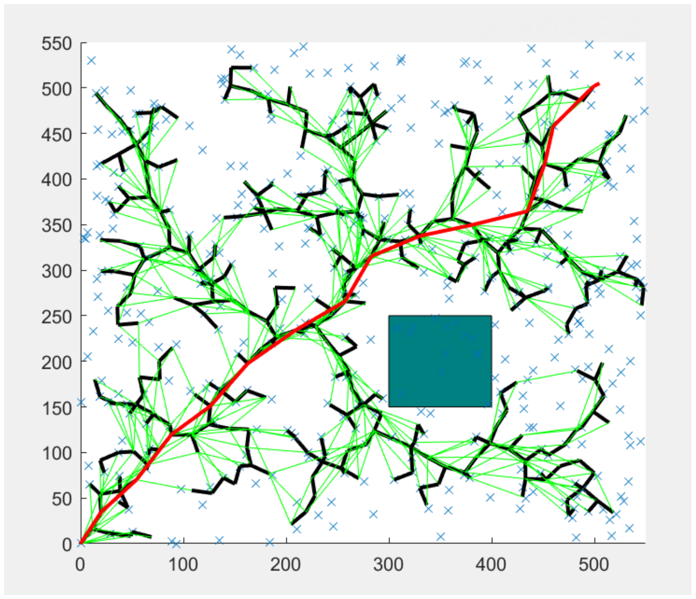


Figure 3-9: RRT planning in MATLAB*

Table 3-1 below list the important parameters to be set for the simulation.

Table 3-1: List of parameters to set for simulation

Type	**Variable**	**Value**
Fixed Parameter	List of obstacle destinations vertices	Obs_D_x = [150, 450, 200, 150, 400, 50, 200, 300, 300, 100, 200]; Obs_D_y = [250, 300, 100, 100, 400, 100, 200, 400, 300, 50, 350];
	Moving obstacle shape and size	Square, 20 pixels
	Obstacle speed	5
	Robot speed	5
	Size of robot	10 pixels radius
	Shape of robot	Circle
Variable Parameter	Detection range threshold	75 pixels
	Angle threshold	0.87 radians (50 degrees)
	Multiplier	Distance between obstacle and robot / 10
	Sampling range allowance	+- 30 pixels from start and goal position
	Size of enlarged moving obstacle	40 pixels (200% increase in size)

All subsequent simulations will follow these variables in the same environment with one static variable. The main difference between the simulations is how the obstacle move. There are 11 different vertices that the obstacle can travel between. Figure 3-10 below shows the position of all possible vertices that can be selected to form a path for the obstacle. The initial preplanned path is plotted in blue while the vertices are plotted in orange. There are ample opportunities for the moving obstacle to cross path with the robot to test the replanning algorithm.

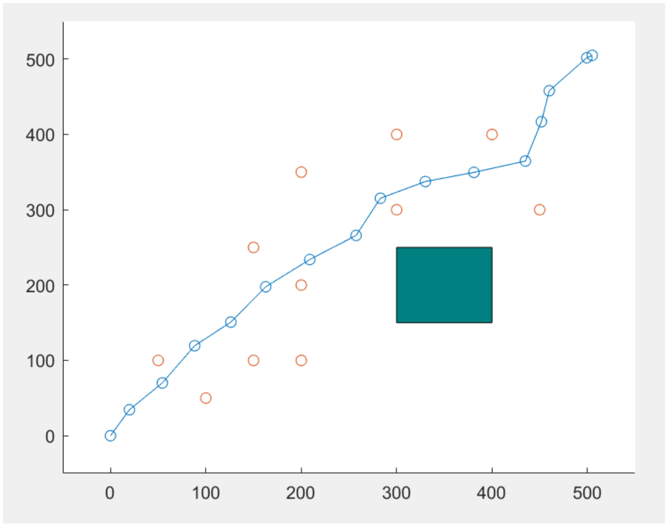


Figure 3-10: All possible vertices for obstacle path in the simulated

Every movement of the robot and moving obstacle is done frame by frame. Due to the serial processing nature in MATLAB, whenever a replan event occur, the frame will pause and wait for the replanning to be done before rewiring the new path and continuing. The initial preplanned path is plotted in blue with replanned paths plotted in different colours for easier visualisation in Figure 3-11.

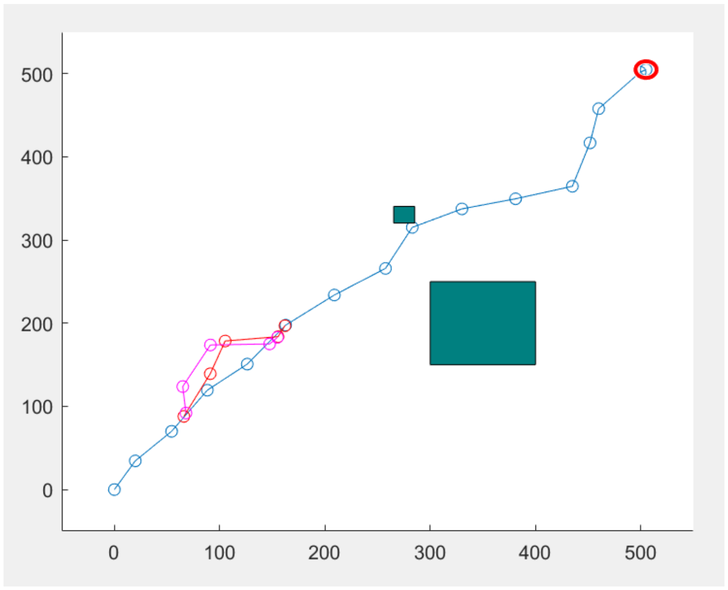


Figure 3-11: Completed path execution example with replan paths in different colours

3.6.2 Results and Discussion

From the simulation, it can be observed that the algorithm is capable of identifying the type of interaction between the robot and obstacle, predict the future obstacle position and replanning a new path in a relatively short period of time to avoid the moving obstacle. Figure 3-12 demonstrates the robot’s ability to plan a path around the moving obstacle to avoid it effectively. The artificially enlarged obstacle has been plotted at the predicted position to explicitly show how it influences the direction of the replan path. The predicted position is calculated from the direction of travel of the moving obstacle and the current obstacle location.

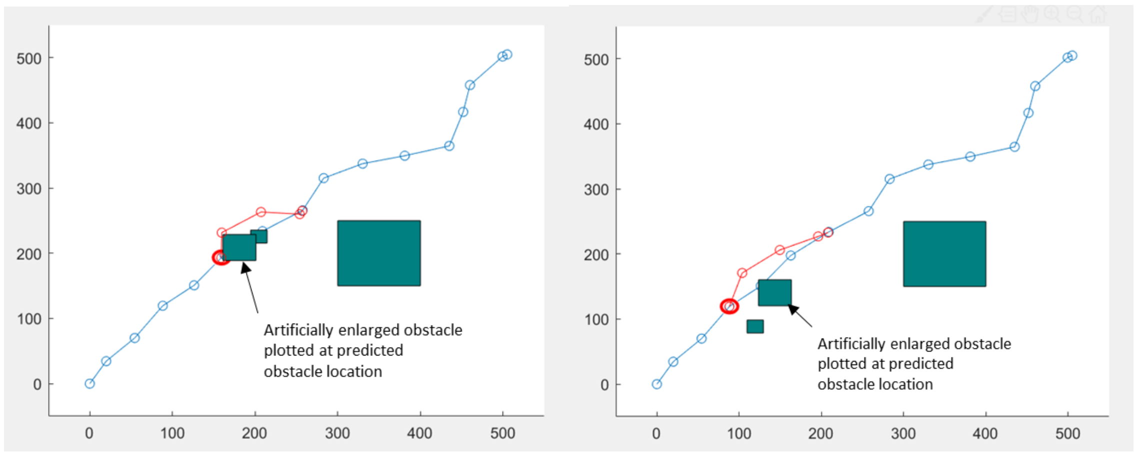


Figure 3-12: Two examples where the robot decides to replan a new path due to an impending collision with the obstacle

Several runs and basic visual check were done to ensure the effectiveness of the algorithm before proceeding on to modifying it to be 3-D compatible. The algorithm will undergo more rigorous testing and simulation during the 3-D implementation.

Chapter 4: Dynamic Path Replanning Algorithm in 3-D

4.1 Overview

In this chapter, the newly improved and tested dynamic replanning algorithm from chapter 3 will be further modified to encompass the third dimension. The overall steps of the path execution algorithm in 3-D is the same as the 2-D path executing algorithm. However, with the third dimension, both the path execution and replanning process becomes more complex. Furthermore, as both the robot and obstacle can move in any direction in 3-D space, a more robust path obstruction check algorithm will be required. Using the concept of intersecting lines in a 3-D vector space, a check is done to determine if there is an impending collision between robot and obstacle based on obstacle trajectory prediction. The whole algorithm and corresponding functions have been coded in MATLAB and will be further elaborated in the following sections. The 3-D geometry toolbox developed by David Legland is used to create and visualize 3-D objects in the MATLAB simulation and perform path collision check during the replanning process [25]. The RRT* path planning algorithm on MATLAB used during the replanning process is adopted from Sai Vemprala [24].

4.2 Path Execution of Robot

The path execution steps are essentially the same in both 2-D and 3-D. For this section, it can maintain most of its logical structure but has to account for the third dimension, refer to algorithm 3 in section 3.2 for more details. The robot now has a specified volume and is provided with a 3-D configuration space with one or more static obstacles. The initial preplanned path will be generated in 3-D using RRT or its variants based on the static environment. All future simulations will also have the same 3-D environment for the preplanned path to remain valid. During the simulation, the robot will traverse the preplanned path by travelling from one node to another based on a frame by frame basis based on the specified speed which accounts for all 3 dimensions. Similar to the 2-D implementation, the path execution process will only end when the robot reaches the end of the preplanned path, its goal node. Whenever the robot encounters a random moving obstacle while executing its path, a replan event will be triggered.

4.3 Random Moving Obstacles and Detection

The random moving obstacle serves the same purpose of forcing the robot to dynamically replan to prevent a collision and still moves in the same randomized manner between vertices similar to the 2-D implementation. However, the dynamic obstacle now has a height and volume and is capable of moving in the z-axis after accounting for the third-dimension.

Sensors are also excluded from the 3-D simulation. A detection sphere of a specified radius is placed on the robot. Although there is no direct equivalent of any sensor capable of doing spherical check, multiple 3-D depth cameras or other sensors facing in different directions will achieve that effect as well. When the moving obstacle comes within the detection sphere range, it must be observed for at least two timesteps to obtain the relevant information to calculate the direction of the moving obstacle (lines 1 and 2 of algorithm 6) and follow up with a path obstruction check.

4.4 Obstruction Check Based on Trajectory Prediction

The major difference between the 2-D and 3-D implementation is the path obstruction check between the robot and the obstacle. Algorithm 6 shows the 3-D path obstruction check algorithm. Within the 2 timesteps in the detection sphere, coordinates of both robot and obstacle are stored (line 1 and 2 of algorithm 6) and will be used to calculate the velocity vectors of the robot and obstacle respectively (lines 5 and 6 of algorithm 6). The velocity vectors are calculated by following equation 6 below. A simplified flowchart of the 3-D path obstruction check algorithm is shown in Figure 4-1.

$$Velocity_{vector} = [(X_2 - X_1), (Y_2 - Y_1), (Z_2 - Z_1)] \tag{6}$$

Algorithm 6: PathObstructionCheck3D()

```
1   if pdist < threshold then
2     StoreCoordinates()
3     timestep++
4     if timetep == 2 then
5       RobotVelocityVector()
6       ObsVelocityVector()
7       if ParallelCheck() & SimiliarDirectionCheck() then
8         if Tooclose() then
9           Replan == TRUE
10        end
11      else
12        if CheckforImpendingCollision() then
13          Replan == TRUE
14        elseif Tooclose() then
15          Replan == TRUE
16        end
17      end
18      timestep = 0
19    end
20  end
```

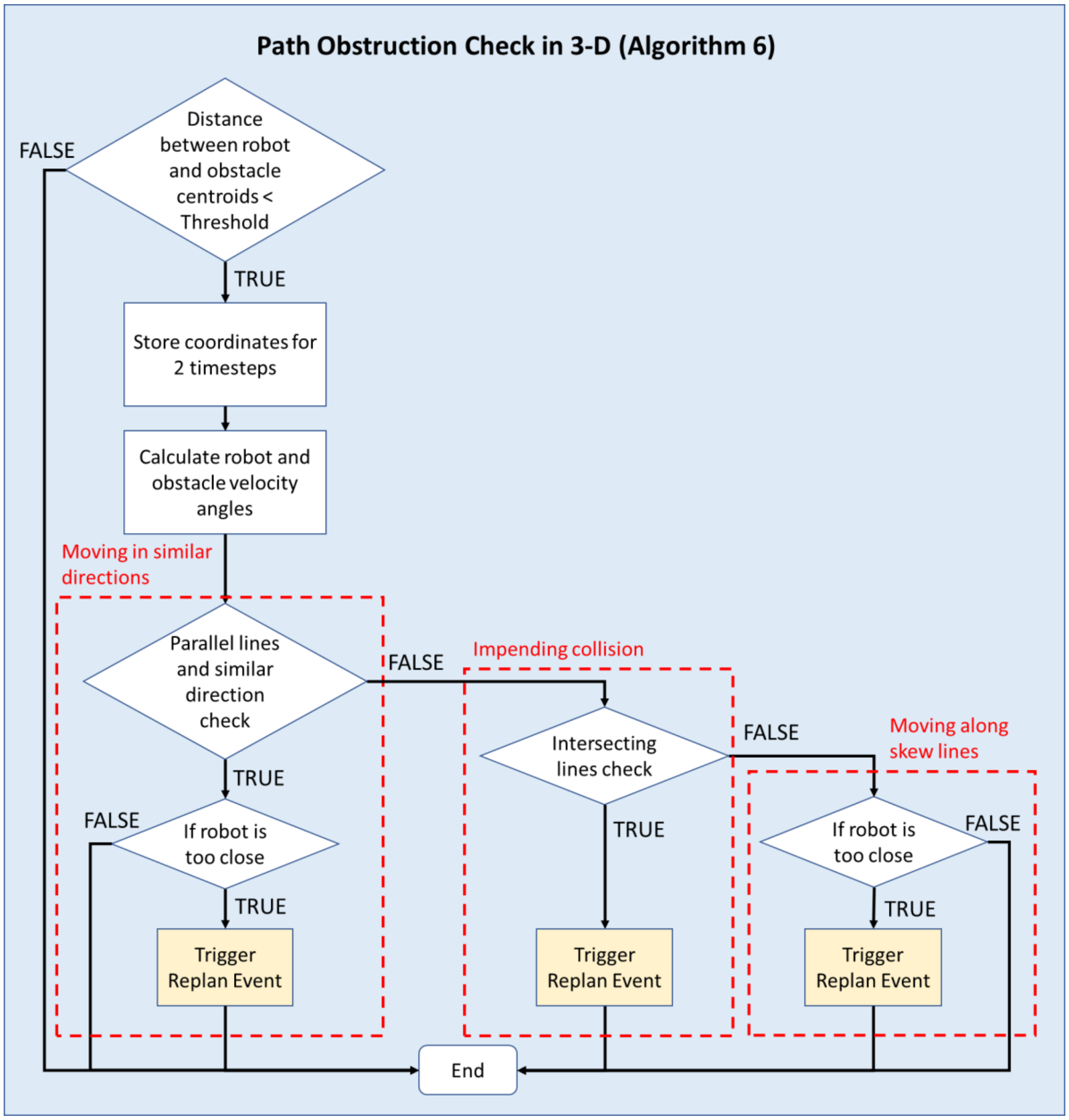

Path Obstruction Check in 3-D (Algorithm 6)
Distance between robot and obstacle centroids < Threshold
FALSE
TRUE
Store coordinates for 2 timesteps
Calculate robot and obstacle velocity angles
Moving in similar directions
Parallel lines and similar direction check
FALSE
Impending collision
TRUE
FALSE
If robot is too close
TRUE
Trigger Replan Event
Intersecting lines check
TRUE
Trigger Replan Event
FALSE
Moving along skew lines
If robot is too close
FALSE
TRUE
Trigger Replan Event
End

Figure 4-1: Simplified flowchart for path obstruction check in 3-D

With the third dimension, the interaction between the robot and obstacle becomes much more complex. The robot and obstacle can now move in any direction in 3-D space and can approach each other in numerous ways. Previous trigonometry concepts used in the 2-D implementation may still apply but becomes a lot more complex as there are now

3 different directional angles to account for in a 3-D cartesian coordinate system. A new method of path obstruction check is proposed in this section. Using the 3-D coordinate geometry concepts, there are three different types of relationship that two lines in 3-D space can have. Two lines in a three-dimensional space can either be parallel lines, non-parallel and intersecting lines or skew lines, which are non-parallel and non-intersecting lines. Figure 4-2 below will illustrate the three relationships.

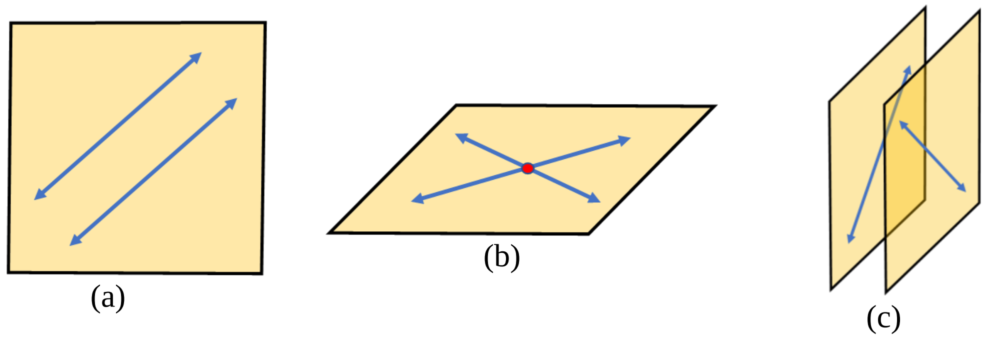


Figure 4-2: (a) Parallel lines on the same plane, (b) Intersecting lines, (c) Skew lines

By modelling the velocity vectors of the robot and obstacle into 2 different straight-line path trajectory and analyzing them, the infinite number of interactions between the robot and obstacle in 3-D space can be classified into 3 distinct cases. The first case of parallel lines would correspond to the robot and obstacle moving in similar or opposite directions. In the second case, the non-parallel and intersecting lines resemble the situation where the robot and obstacle are going to collide at a point. Finally, the last case where the lines are non-parallel and non-intersecting can be related to the situation where the robot and obstacle moving in different planes such as flying at different elevations. Figure 4-3 will illustrate all three cases.

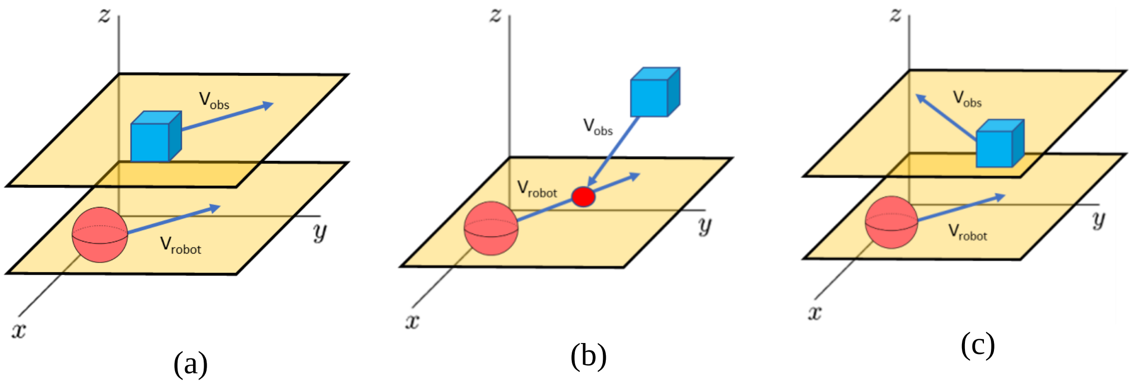


Figure 4-3: Case 1: Robot and obstacle moving in similar direction, (b) Case 2: Impending collision between robot and obstacle, (c) Case 3: Robot and obstacle moving in different planes

The straight-line path trajectories can be derived from the velocity vector and the current position of the robot and obstacle respectively.

$$Path\ trajectory\ line = (X_{robot}, Y_{robot}, Z_{robot}) + s * V_{robot} \quad (7)$$

Equation 7 above shows how to determine the path trajectory line of the robot, where (X_{robot}, Y_{robot}, Z_{robot}) are the current coordinates of the robot, V_{robot} is the velocity vector of the robot and *s* is a constant to be determined. The velocity vector of the obstacle can be determined by substituting the respective obstacle coordinates and the obstacle velocity vector into equation 7.

In the first case, since the direction of travel are parallel, the robot and obstacle can be moving in similar or opposite directions. While moving in opposite directions may imply an impending collision in 2-D, it is not always the case in 3-D. In a 3-D configuration, when the robot and obstacle direction of travel are parallel and opposite, there is a possibility that both path trajectory lines lie in different planes or are located far enough on the same plane. The robot and obstacle will not collide unless the two opposite parallel paths are sufficiently close. An illustration is shown in Figure 4-4 below.

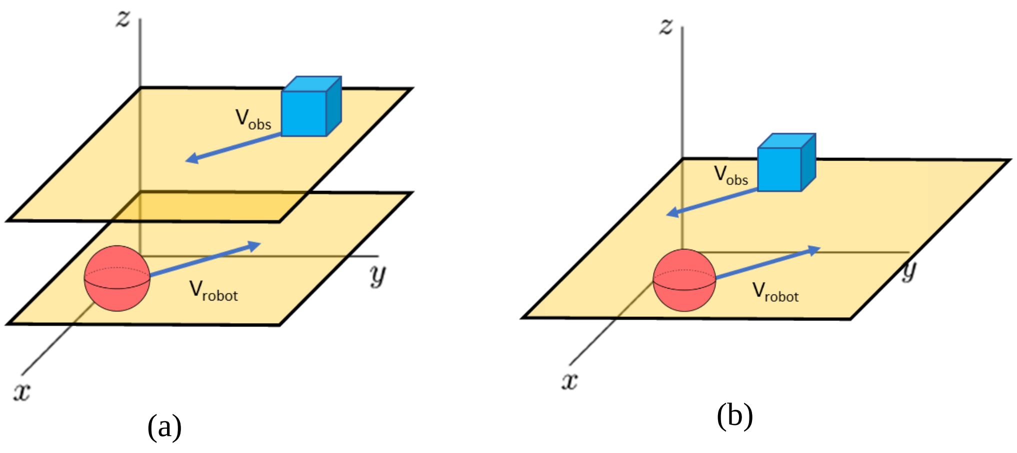


Figure 4-4: Second auxiliary case of case 1 where the robot and obstacle velocity vectors are parallel and opposite. (a) lie on different planes, (b) lie on the same plane

Given this ambiguity, the first case is split into two auxiliary cases, moving in similar directions and moving in opposite directions. Only the first auxiliary case is certain and will be caught (line 7 of algorithm 6), the robot and obstacle are moving in similar directions and will not collide if they continue on the same path. The robot will ignore the obstacle and continue following its preplanned path unless it gets too close. A replan event is triggered when the proximity between them is below a specified distance (line 8 and 9 of algorithm 6). However, if the obstacle or path of the robot changes direction, the robot will have to recalculate accordingly and choose a course of action based on any of the other two cases. The second auxiliary case will be ignored initially but will be caught later on if the robot and obstacle get too close (lines 14 and 15 of algorithm 6).

For the second case, since the path trajectory of the robot and obstacle intersect at a point in space. Assuming that this intersection point is found in front of the robot, the robot and obstacle are moving towards each other and there will be an impending collision if nothing is done. The robot immediately triggers a replan event to find another path to avoid the collision (lines 12 and 13 of algorithm 6). This intersection point can be calculated by equating both the path trajectory line of the robot and obstacle and solving it in a system of linear equation in matrix form A**x** = **b**.

$$l_{robot\_trajectory} = (X_{robot}, Y_{robot}, Z_{robot}) + s * V_{robot}$$

$$l_{obs\_trajectory} = (X_{obs}, Y_{obs}, Z_{obs}) + t * V_{obs}$$

$$l_{obs\_trajectory} = l_{robot\_trajectory}$$

$$\begin{bmatrix} X_{obs} + t * V_{x\_obs} \\ Y_{obs} + t * V_{y\_obs} \\ Z_{obs} + t * V_{z\_obs} \end{bmatrix} = \begin{bmatrix} X_{robot} + s * V_{x\_robot} \\ Y_{robot} + s * V_{y\_robot} \\ Z_{robot} + s * V_{z\_robot} \end{bmatrix}$$

$$\begin{bmatrix} t * V_{x_{obs}} - s * V_{x\_robot} \\ t * V_{y\_obs} - s * V_{y\_robot} \\ t * V_{z\_obs} - s * V_{z\_robot} \end{bmatrix} = \begin{bmatrix} X_{robot} - X_{obs} \\ Y_{robot} - Y_{obs} \\ Z_{robot} - Z_{obs} \end{bmatrix}$$

$$\begin{bmatrix} V_{x_{obs}} & -V_{x\_robot} \\ V_{y\_obs} & -V_{y\_robot} \\ V_{z\_obs} & -V_{z\_robot} \end{bmatrix} \begin{bmatrix} t \\ s \end{bmatrix} = \begin{bmatrix} X_{robot} - X_{obs} \\ Y_{robot} - Y_{obs} \\ Z_{robot} - Z_{obs} \end{bmatrix} \Leftrightarrow A\boldsymbol{x} = \boldsymbol{b}$$

$$\begin{bmatrix} t \\ s \end{bmatrix} = \begin{bmatrix} V_{x_{obs}} & -V_{x\_robot} \\ V_{y\_obs} & -V_{y\_robot} \\ V_{z\_obs} & -V_{z\_robot} \end{bmatrix}^{-1} \begin{bmatrix} X_{robot} - X_{obs} \\ Y_{robot} - Y_{obs} \\ Z_{robot} - Z_{obs} \end{bmatrix} \Leftrightarrow \boldsymbol{x} = A^{-1}\boldsymbol{b}$$

Since there are 3 equations and only 2 unknown, m = 3 > n = 2, this is an overdetermined system. If the solution found is not unique, the least squares solution will be returned. Substitute the least square *s* and *t* into the equation of path trajectory for the robot and obstacle to obtain the two estimated intersection point. If the two estimated intersection points are within a specific tolerance, the intersection point is deemed to be legitimate. Since the equation of path trajectories are lines with infinite length, there may be situations that the intersection point is found behind the robot or too far away from the current robot position. Figure 4-5 below will illustrate some examples. To ensure that the legitimate intersection point is found ahead of the robot and is not located too far away, the intersection point has to be located within a specified distance from the next node of the robot's current path for the replan event to be triggered (line 12 of algorithm 6).

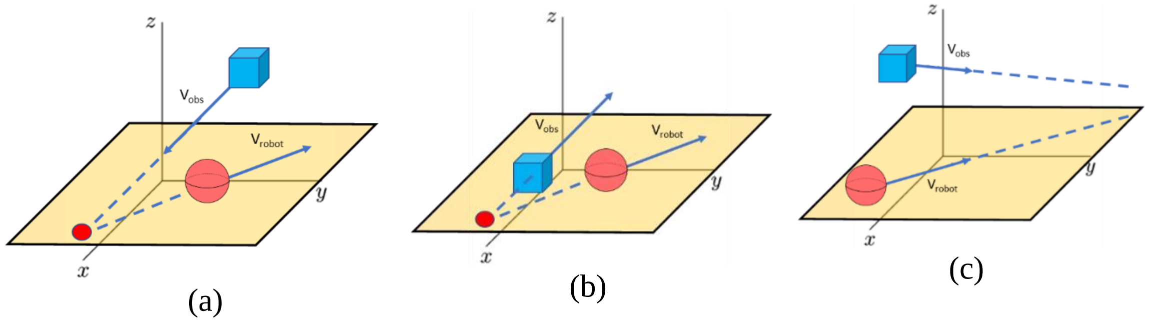


Figure 4-5:(a) & (b) Path trajectories intersect behind the robot, (c) Intersection point too far away from current robot position

Finally, the third and last case occurs when the path trajectory of the robot and obstacle are skew lines. Since skew lines are non-parallel and non-intersecting, it will fall through the parallel check in case one and intersection point check in case two. Similar to the first auxiliary case of case one, it is unlikely for the robot to collide with the moving obstacle if both robot and obstacle continue on their path trajectory. The robot will ignore the obstacle and continue following its preplanned path unless it gets too close. A replan event is triggered when the proximity is below a specified distance (lines 14 and 15 of algorithm 6). Figure 4-6 is an example where the robot and obstacle are moving along skew lines.

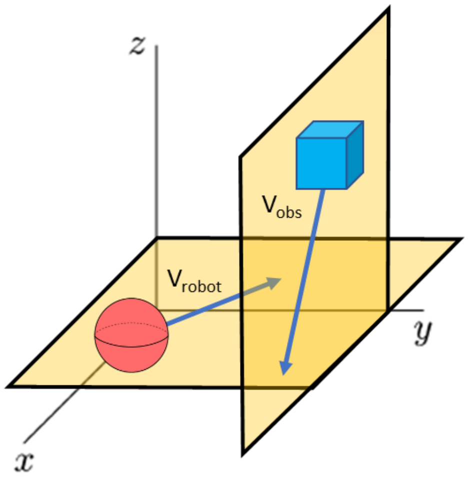


Figure 4-6: Example where the path trajectories of robot and obstacle are skew lines

4.5 Planning and Rewiring Operations

The replanning algorithm in 3-D largely follow the steps for 2-D detailed in algorithm 5. When a replan event is triggered in the section above, the replanning and rewiring process will run. Otherwise, the robot will skip this algorithm and continue on its preplanned path.

The obstacle trajectory prediction multiplier will also be used for the 3-D implementation given its contribution to a more effective replan path in the 2-D simulations. This multiplier should be proportional to the current distance between the obstacle and the robot. The further the obstacle is from the robot, the larger the multiplier and vice visa. The multiplier will help bring the predicted position of the obstacle closer to cause a more intentional replan path while the obstacle is still beyond the safe distance.

Similar to equation 5, the predicted obstacle location can be found following the equation below.

$$Predicted\ obstacle\ coordinates = (X_{obs}, Y_{obs}, Z_{obs}) + multiplier * V_{obs} \qquad (8)$$

Using the predicted obstacle coordinates, any nodes that are within a specified range around it are considered to be in collision with the obstacle and are invalidated.

The assumption made in 2-D will be carried followed to 3-D implementation as well. It was assumed that the preplanned path is the most optimal path leading to the goal location and the robot should return to this path after avoiding the moving obstacle. Furthermore, the goal node selection will follow the same criteria with slight changes to the specified distance and account for the third dimension. The configuration space in 3-D will also be determined by the limits of the start and goal node but with a larger allowance than in 2-D. After the required inputs have been calculated, the path planning algorithm will run. To account for prediction error and allowance for the size of the robot, the size of the obstacle will be artificially increased during the path replanning process such that the replan path generated will avoid the area around the obstacle as well. After the replan path is generated and rewired to the preplanned path, the robot will then travel on this new replan path. If the robot encounters another obstacle and determines that another replan is required, the replan process will be repeated.

In 3-D space, the third dimension makes it much more computationally expensive to generate a collision-free path. RRT* path planning will be employed during the replanning process. Since RRT does not have the additional path optimizing steps that is characteristic of RRT*, it is computationally faster than RRT*. RRT will also be run in a different set of simulation for comparison on path replanning time. In addition, as collision checking is usually the primary computational bottleneck of sampling-based motion planning algorithm [26], special attention will be given to the collision checking function within the path planning algorithm.

While the replanning algorithm may be logically sound, MATLAB simulation results in section 4.6 will highlight the various inadequacies the algorithm have and methods to improve its effectiveness will be suggested.

4.6 MATLAB Simulation in 3-D

Based on the algorithm described in sections 4.2 to 4.5 above, a MATLAB program has been written to simulate and evaluate its effectiveness. Simulations are conducted using a laptop with Intel i7 2.4GHz processor and 8GB RAM. This section will cover the implementation details in MATLAB, the testing and evaluation methodology followed by the modifications and improvements done and the results from different simulation set.

4.6.1 Implementation Details

A three-dimensional configuration space of 1000 x 1000 x 1000 pixels is created for this simulation. Within the environment, three large static obstacles, a wall, a cylindrical pillar and a wall with an opening, are generated. The start node is set to be at (50, 50, 50) and the goal location at (900, 900, 900). Using the RRT path planning algorithm MATLAB program adopted from NTU PhD student Wang Yuan Han together with the 3-D geometry toolbox developed by David Legland [25], a preplanned path has been generated in red in Figure 4-7 below. The preplanned path is then saved as a file to be used for future simulations. All simulations will have the maintain the same static environment.

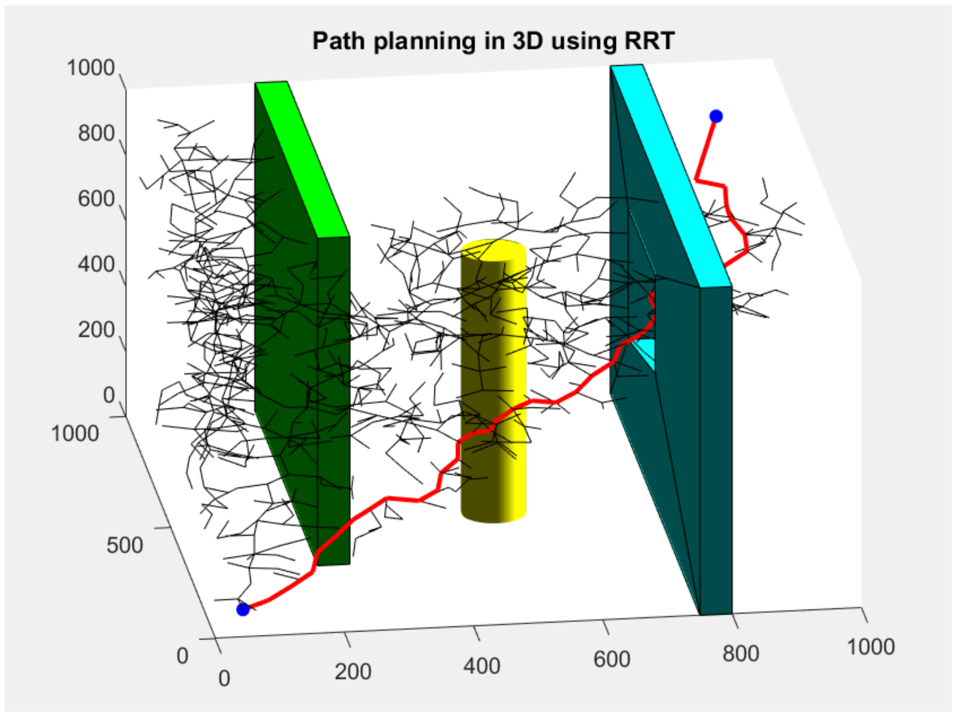


Figure 4-7: RRT path planning in 3-D

Table 4-1 below list the important parameters to set in each simulation.

Table 4-1: List of parameters to set for simulation

Type	**Parameters**	**Value**
Constant Parameters	List of moving obstacle destinations vertices	Obs_D_x = [100, 355, 400, 100, 285, 355, 160, 650, 650, 235, 715, 400, 300, 250]; Obs_D_y = [100, 275, 200, 100, 230, 275, 100, 450, 520, 200, 560, 300, 250, 200]; Obs_D_z = [200, 180, 500, 100, 200, 180, 135, 450, 400, 170, 410, 300, 200, 170];
	Moving obstacle shape and size	cube, 30 pixels
	Obstacle speed	8
	Robot speed	8
	Size of robot	20 pixels radius
	Shape of robot	Sphere
Variable Parameters	Detection sphere threshold	150 pixels radius
	Safety distance to trigger replan	120 pixels radius
	Minimum distance between start and goal node	100 pixel
	Multiplier	Distance between obstacle and robot / Obstacle speed
	Sampling range allowance	+- 30 pixels from start and goal position
	Size of enlarged moving obstacle	40 pixels (200% increase in size)

All subsequent simulations will follow these constant parameters in the same environment with three static obstacles. Variable parameters are subjected to refinement between sets of simulation trials. Within the same set of simulation, the main difference between every simulation is how the obstacle move. There are 14 different vertices that the obstacle can travel between. Figure 4-8 displays the position of all the possible vertices that can be randomly selected to form a path for the obstacle. The initial preplanned path is plotted in black while the vertices are plotted in red asterisks for illustration. There are ample opportunities for the moving obstacle to randomly cross path with the robot in various directions to test the replanning algorithm.

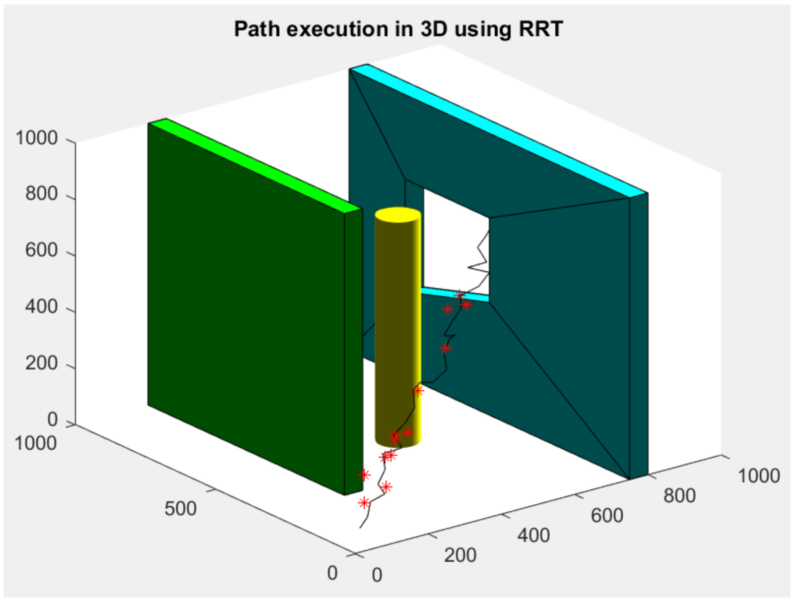


Figure 4-8: Red asterisks represent all possible vertices for obstacle path in the simulated environment

Every movement of the robot and moving obstacle is done frame by frame. Due to the serial processing nature in MATLAB, whenever a replan event occur, the frame will pause and wait for the replanning to be done before rewiring the new path and continuing. For all simulations, the initial preplanned path is plotted in black with replanned paths plotted in different colours for easier visualization as seen in Figure 4-9.

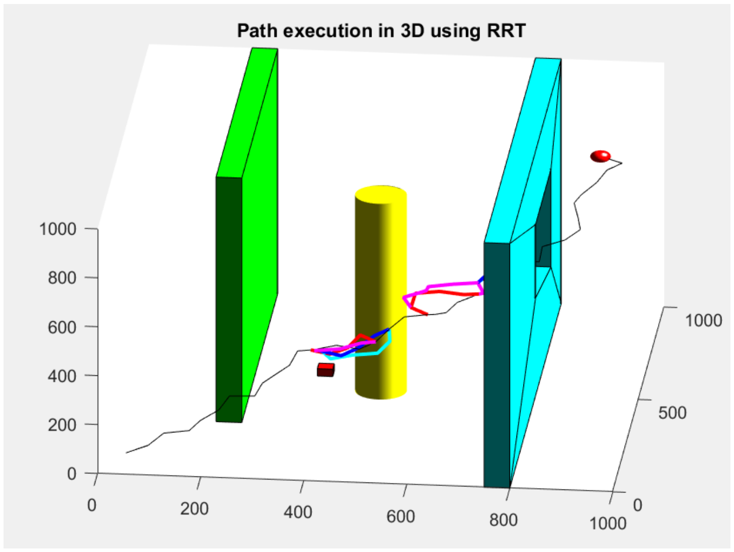


Figure 4-9: Example of a completed simulation run with replanned paths plotted in different colours

4.6.2 Testing and Evaluation Methodology

As the path of the moving obstacle is completely randomized, there are multiple ways the robot may encounter the obstacle. Furthermore, there may also be situations that the robot and obstacle may not encounter each other for the entire run. A sufficiently large number of runs are required to objectively evaluate the robustness of the replanning algorithm. 100 runs will be done per set of simulation and crucial data will be recorded in each run.

For a path planning method, path length and path planning time are the most common metrics in the measurement of performance [27]. However, in the case of dynamic replanning with random moving obstacles, path length or path optimality should not be used as a metric for evaluation as the localized path planning environment is not consistent. The localized path planning environment changes according to the start and goal node of the replan, the predicted obstacle location and the presence of static obstacles (if any). Path planning time will be used as a performance criterion due to the real-time requirement in dynamic path replanning. Since the number of replan in a simulation run is unpredictable, an average will be taken across all replanning time in all 100 simulation runs.

The percentage of successful replan within a specified time limit is another performance criterion to be considered. A literature review done by Robert J. Kosinski stated that the choice reaction time for average human reaction time is the longest as compared to simple and recognition reaction time. It was discovered that recognition reaction times averaged 384 milliseconds [28]. Since the path planning process can be seen as a subset of choice reaction, a conservative estimate of 400 milliseconds will be set as the real-time requirement of the path planning process.

In the simulation, a successful replan path may not necessarily mean a successful avoidance between the robot and moving obstacle due to multiple factors. This could be due to various factors such as trajectory prediction error, insufficient clearance between replan path and obstacle, a sudden change in obstacle path direction or other uncertainties due to the dynamic motion of both robot and moving obstacle. To ascertain the robustness of the replanning algorithm, the number of collisions out of 100 runs should be monitored. This can be converted into a collision rate per set of trials. In order to determine whether the robot and the obstacle are in collision, the distance between their centroids is calculated and compared against the minimum distance between them when the surface of the robot and obstacle are in contact. Given that the sphere representing the robot is 20 pixels in radius and the obstacle is a cube of 30 pixels length, the minimum distance required between them can be calculated by equation 9 and rounded up to be 42 pixels.

$$Minimum\ distance\ before\ colliding = Radius_{robot} + \sqrt{(\frac{1}{2} * length_{obs})^2 * 2} \qquad (9)$$

With perfect knowledge of the position and obstacle during the simulation, the relative distance between their centroids can be consistently tracked and compared with the result of equation 9. When the relative distance between the centroid falls below the minimum distance, the robot and obstacle are deemed to have collided and the simulation will end abruptly. The run will be recorded as a collided trial and the subsequent run will proceed.

Since RRT and its variants are probabilistically complete methods, there are chances of replan failure, where no feasible replan path could be found. It is expected that replan failures will occur rarely but will be noted if it occurs.

The type of encounter (4 cases elaborated in section 4.4: similar direction, impending collision, skew lines, too close) and prediction multiplier will also be recorded during the simulation for monitoring and debugging purposes.

4.6.3 Modifications and Improvements

There were multiple problems faced during the simulation of the proposed replanning algorithm. Many solutions were attempted and the replanning algorithm has gone through iterative improvement after each set of runs and results. Initially, the objective of the path replanning process was to generate an effective replan path to avoid the moving obstacle. It was set as an infinite loop which only breaks when a solution is generated. After the replanning process was determined to be sufficiently effective, the replanning algorithm was only allowed to run once to meet real-time requirements. If no solution could be generated, it will be recorded as a replan failure and the robot will continue on its existing path.

Problem 1: In some runs, the path replanning process was taking a prolonged period of time before generating a replan path.

Problem 2: In worse cases, no replan path was generated even after several minutes.

With regards to problem 1, it was hypothesised that the predicted obstacle position was close enough to the robot such that the artificially enlarged obstacle blocks out almost all the possible paths that could be generated resulting in a few replan failures before being successful in generating a replan path. For problem 2, it was concluded that the predicted obstacle position may have been too close to the robot such that the artificially enlarged obstacle completely overlaps with the robot's current position, resulting in an unsolvable solution.

Attempted solution:

Several variables were adjusted to try to rectify this issue.

1. The trajectory prediction multiplier was tuned to ensure that the overlapping does not happen. This was done by adjusting the multiplier until the predicted obstacle position is at a specific distance away from the robot. However, prolonged path planning time still occurs sometimes. The case where no solution path was generated became rare.
2. The start and goal node distance requirement and sampling range allowance were also increased to enlarge the configuration space for a greater probability of generating a collision-free path.
3. Reducing the size of the artificially enlarged obstacle at its predicted location helped to lower the occurrence rate of both problems but made the replan path much more ineffective in avoiding the moving obstacle.

It was later discovered that the path collision check code within the RRT path planning algorithm from the geom3d toolbox [25] was not functioning at it should be. The path collision check code converts the line segment between the new node and near node into an infinite line and checks if the infinite line intersects with the surface of both the moving and static obstacles. This invalidated most or all of the nodes that were supposed to be valid, resulting in a prolonged check or no solution being found. Since only intersections between the line segment and the obstacle surfaces are important, bounds were set to ignore intersections beyond the line segment. As a result, more randomly generated nodes are now validated in the correct manner, significantly speeding up the replanning process and increasing chances of finding a solution.

After implementing the fixes above, the occurrence of problem 2 was greatly reduced but not eliminated. It was discovered that this was due to the rare cases where the obstacle reaches its destination vertex and the next random vertex chosen happens to be the same vertex. In such cases, the obstacle will remain stationary for the two observed timesteps and its calculated velocity vector is zero. If the obstacle is sufficiently close to the robot, the overlap situation will occur when the obstacle is artificially enlarged. However, the prediction multiplier will not be able to move the obstacle away since the multiplier only affects the predicted location when the obstacle velocity vector is non-zero. Hence, the unsolvable situation occurs and no replan path could be generated. The solution was to observe the obstacle for another time step if both the stored coordinates of the obstacle were identical. Only if the velocity vector of the obstacle is non-zero, will the replan event be triggered.

Alternatively, the moving obstacle could be plotted as its actual size when it is static similar to how other static obstacles are plotted during the replanning stage.

However, this may not provide sufficient spatial clearance for the robot if the obstacle starts moving again in the next instant, leading to the next problem of high collision rate.

Problem 3: High collision rate of approximately 30% in 100 runs.

There are 3 main reasons for this problem. First is the poor prediction of the obstacle future location. Next is the case where the obstacle suddenly changes direction after the replan and the robot does not have enough spatial allowance to travel on its new replan path. Finally, would be the obstacle approaching the robot from behind. The path obstruction check process is able to confirm that the obstacle is approaching but the replanning algorithm could not account for it unless its predicted location is found within the localised configuration space in front of the robot.

Attempted solution:

Several variables were adjusted to try to rectify this issue.

1. Artificially increasing the size of the predicted obstacle during path replanning. Although this helps to account for the prediction error of the obstacle, it restricted the free configuration space, resulting in prolonged path planning time or in worse cases, result in problem 2 where no solution could be found.
2. Prediction multiplier fine tuning. In section 4.5, it was mentioned that the prediction multiplier was to be proportional to the distance between the robot and the current obstacle position. Equation 8 shows how the multiplier is being used to calculate the predicted obstacle location. There were also some attempts to use a fixed predictor multiplier value.

After some observation of the plotting of the obstacle in its predicted location, it was noted that the prevention of overlap between predicted obstacle location and robot was the main reason for the ineffective avoidance of the replan. When the obstacle approaches the robot from the front, the overlap prevention works well in diverting the new replan path around the obstacle. However, when the obstacle approaches from its sides or from behind, the overlap prevention will result in the predicted obstacle location being pushed back. This prevents the predicted obstacle position from influencing the path planning process, thus resulting in a path that is ineffective in avoiding the obstacle in such situations.

A new method of collision check is proposed: bounding sphere collision detection. This method is faster than checking the intersection of the path with every single side of the cubic obstacle. During the replanning process, the artificially enlarged obstacle has been replaced with a bounding sphere is around the predicted obstacle position. All newly generated nodes in the path replanning process must lie a safe distance beyond the bounding sphere to be valid. Only the current position of the robot, the initial node of the replan path can lie within the sphere. This allows for more flexibility of overlap between the robot and the predicted obstacle bounding sphere without causing an unsolvable situation. In addition, since the bounding volume around the obstacle is invalid for new nodes, a more effective replan path to avoid the obstacle will be generated. Variables such as the minimum distance between the robot and predicted obstacle location, and size of bounding sphere were adjusted to see its effects. Simulation results showed that collision rate drops to about 15-20%, showing the improvement in the algorithm.

Penultimate Replanning Algorithm

In the previous method, since only newly generated nodes are checked to be beyond the bounding sphere, there was a possibility that the path between the nearest node and the newly generated node could still pass through the bounding sphere. In such cases, the replan path will not be as effective in avoiding the obstacle, increasing the chance of collision. Figure 4-10(a) illustrates the issue. Using 3-D geometry concepts, the shortest distance from a point to a line segment could be determined. By ensuring that the shortest distance was larger than the radius of the sphere, it can be ascertained that the path does not intersect with the bounding sphere as seen in Figure 4-10(b). Collision rate drops to fell to 5-15% after implementing this additional check but path planning time increased as every new node must undergo the check. In order to speed up path replanning, a second larger bounding sphere was added to set a maximum range in which the path to the new node will be checked. Only new nodes that are generated between the smaller sphere and the larger sphere will be checked. This corresponds to the yellow shaded region in Figure 4-10. New nodes generated in the smaller sphere in white are invalid and will be rejected while those outside the larger sphere are deemed to be safe enough to be left unchecked.

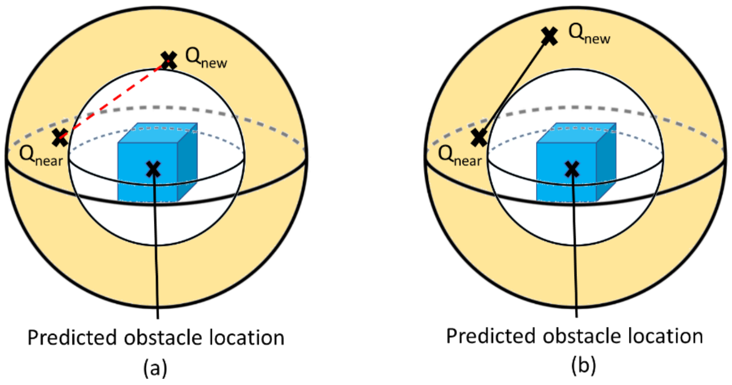


Figure 4-10: Bounding sphere collision check method. (a) Ensuring only newly generated points are beyond the bounding sphere (b) Ensuring newly generated points and path are beyond the sphere

If the nearest node to the new node is the current robot position that lie within the bounding sphere, the path connecting them will be diverted away from the obstacle position by a minimum of 90 degrees. This will deflect the initial portion of the replan path away from the predicted obstacle location while the additional checks above ensure that the rest of the path does not intersect with the bounding sphere around the obstacle position.

Final Replanning Algorithm

The penultimate replanning method has been mostly effective in avoiding the moving obstacle but there are still a few cases where the replan path moves towards incoming obstacle while moving around the bounding sphere located on the predicted obstacle location. This is mainly due to the modelling of a specific future position based on the direction of travel without accounting for the time dimension. The final improvement to the algorithm is the addition of another collision check method to account for the direction of travel of the obstacle with respect to time. A space-time volume representation of the moving obstacle is implemented together with the existing two bounding sphere check. The velocity vector of the obstacle will be modelled as a line and all newly generated nodes found between the smaller and larger bounding spheres are to be a specific distance away from the line for them to be valid. This is done with the exception of one or a few newly generated nodes whose near node is the current robot position that lies within the smaller sphere. An illustration of the concept is shown in Figure 4-11. The light blue shaded area is the space-time volume representation of the moving obstacle plotted at the predicted obstacle location but restricted to the size of the larger sphere. Similar to the penultimate algorithm, all newly generated nodes must lie beyond the smaller bounding sphere to be valid. In the final improvement, there is an additional criterion that newly generated nodes, other than the few exceptions, must also lie outside the light blue volume to be valid.

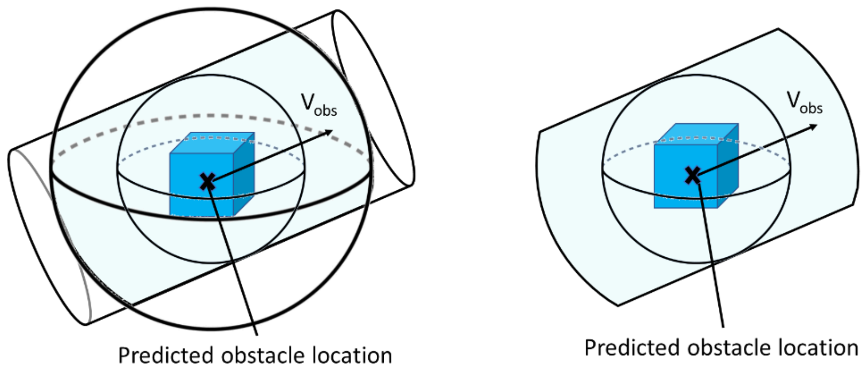


Figure 4-11: Collision check with space-time volume representation of moving obstacle

4.6.4 Results and Analysis

Penultimate Replanning Algorithm Simulation with RRT*

Important variables set:

Table 4-2: Critical variables set for penultimate algorithm simulation

Variable	Value
Prediction Multiplier	Distance between centroids / (obstacle speed*1.15)
Minimum distance between centroids of robot and predicted obstacle location	42 pixels
Changes to prediction multiplier if minimum distance between centroids is not met	-0.5 until it is resolved
Size of larger bounding sphere	100 pixels radius
Size of smaller bounding sphere	60 pixels radius
Minimum distance required between centroid of predicted obstacle location and goal node	80 pixels

- Collision rate of **8%** (8 collisions in 100 runs)
- Total of 1291 replan events, average path replanning time = **99.82 ms**
- Percentage of replanning time exceeding the acceptable real-time limit (400ms) = 55/1291 * 100% = **4.26%**

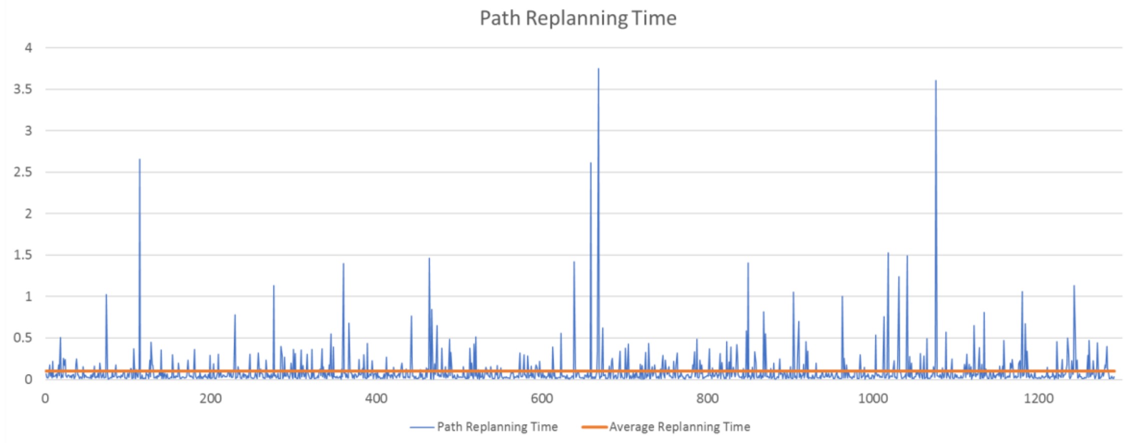


Figure 4-12: Graph plot of replanning time in 100 runs

Final Algorithm Simulation

RRT* based replanning with varying parameters

Important variables set:

Table 4-3: Critical variables set for final algorithm simulation

Variable	**Value**	
	Simulation Set A	Simulation Set B
Prediction Multiplier	Distance between centroids / obstacle speed	
Minimum distance between centroids of robot and predicted obstacle location	42 pixels	
Changes to prediction multiplier if minimum distance between centroids is not met	-0.5 until it is resolved	
Size of larger bounding sphere	100 pixels radius	
Size of smaller bounding sphere	60 pixels radius	
Minimum distance required between centroid of predicted obstacle location and goal node	70 pixels	80 pixels
Line swept sphere (space-time volume representation of moving obstacle)	50 pixels radius	60 pixels

Results for simulation set A:

- Collision rate of **1.4%** (7 collisions in 500 runs)
- Total of 6479 replan events, average path replanning time = **154.42 ms**
- Percentage of replanning time exceeding the acceptable real-time limit (400ms) = 566/6479 * 100% = **8.74%**
- Percentage of replan failures = 2/6479 * 100% = **0.0309%**

Results for simulation set B:

- Collision rate of **0.8%** (4 collisions in 500 runs)
- Total of 6367 replan events, average path replanning time = **175.62 ms**
- Percentage of replanning time exceeding the acceptable real-time limit (400ms) = 632/6367 * 100% = **9.93%**
- Percentage of replan failures = 38/6367 * 100% = **0.597%**

The results from simulation set A and B highlights the efficiency and effectiveness of the developed 3-D dynamic path replanning algorithm. Comparing both results, simulation set B seemed to be more favourable. Both sets are capable of achieving very low collision rates with collision rate in set B being below 1%. Since collision rates were hovering at about 1%, 500 runs were done instead for a more accurate representation of the replanning algorithm performance. The average planning time in both sets are safely within the real-time limit with less than 10% of replan time exceeding the limit. Although simulation set B has a much higher replan failure rate than simulation set A, the failure rate is still very low and can be mitigated.

Further analysis of the collision between robot and obstacle provides two major reasons. First, approximately half the collision is due to a replan failure during localised RRT* planning. The robot is unable to generate a new replan path and thus continued on its path resulting in a collision. This could be attributed to the weakness of sampling-based path planning algorithms in navigating narrow configuration spaces. The narrow configuration space is a result of implementing the space-time volume representation of the moving obstacle. A safety strategy such as stop and wait, backtracking or another safety path can be employed to reduce the occurrence of collision due to replan failure. Alternatively, increasing the detection range and safety threshold to trigger a replan event will allow for more opportunities for replan, mitigating the issue of replan failure. Second, would be the abrupt change in direction of the obstacle velocity towards the robot when the robot and obstacle are in relatively close proximity of each other. In such cases, the robot is already attempting to transverse a replan path based on the obstacle's previous velocity. While the robot is able to replan another new path to avoid the approaching obstacle, it does not have the spatial allowance to maneuver and is unable to move fast enough to fully traverse the new replan path before getting hit by the obstacle. This is mainly due to the requirement of at least two timesteps of observation and a fixed mobile robot speed throughout the simulation. The randomness of the replan path also contributes to the issue. Even though the replan path may be effective in avoiding the obstacle, there may be another path which is more effective in creating a larger distance between the robot and the obstacle. Increasing the detection radius around the robot together with a larger bounding sphere and space-time volume representation of the obstacle will help reduce collisions caused by abrupt changes in obstacle velocity. The robot will be able to react to the incoming obstacle earlier and the replan paths will be even more deliberate in avoiding the area about the moving obstacle. However, caution must be exercised as there is an inherent tradeoff between increasing the size of the space-time volume representation of the obstacle and replan failure rate. Allowing variable speed of the robot will also help reduce such collisions.

Performance comparison between RRT and RRT* based replanning

As simulation set B was able to achieve promising collision rates of below 1%, future simulations will now follow the variables set in simulation set B. Another two sets of 500 runs were done to compare the performance of the replanning algorithm when it is based on different variants of similar path planning methods. Since RRT* algorithm was used in the previous simulation, running the replanning process with the basic RRT algorithm will be a good comparison. RRT* is known to sacrifice computational time for optimality. Hence, it is expected RRT will be able to perform better in terms of meeting the real-time requirement.

Results for simulation running with RRT*:

- Collision rate of **1.2%** (6 collisions in 500 runs)
- Total of 6308 replan events, average path replanning time = **157.25 ms**
- Percentage of planning time exceeding the acceptable real-time limit (400 ms) = 576/6308 * 100% = **9.13%**
- Percentage of replan failures = 27/6308 * 100% = **0.428%**

Results for simulation running with RRT:

- Collision rate of **0.4%** (2 collisions in 500 runs)
- Total of 7121 replan events, average path replanning time = **57.96 ms**
- Percentage of planning time exceeding the acceptable real-time limit (400ms) = 123/7121 * 100% = **1.73%**
- Percentage of replan failures = 14/7121 * 100% = **0.197%**

Both RRT and RRT* based replanning are able to achieve promising results with both having very low collision rates of 0.4% and 1.2% respectively. The exceptionally low collision rate from RRT based replanning might just be a random occurrence. Alternatively, it could be attributed to the lack of optimality in the path resulting in the path taking a longer detour around the obstacle, providing larger spatial allowance between the obstacle and robot. Efficiency is good in RRT* but better in RRT with both replanning time averages being safely below the real-time requirement of 400 milliseconds. Replan within the real-time limit accounts for more than 90% and 98% all replans in RRT* and RRT respectively and only a very small portion of replans, below 0.5%, actually fail to generate a solution.

While the average planning path replanning time shows that RRT is generally faster RRT*, it does not provide information on consistency. The cumulative replanning time plot in Figure 4-13 shows that RRT based replanning is consistently faster than RRT* based replanning. From the best fit gradient of each line, it is estimated that RRT based replanning is approximately 2.6 times faster than the RRT* counterpart.

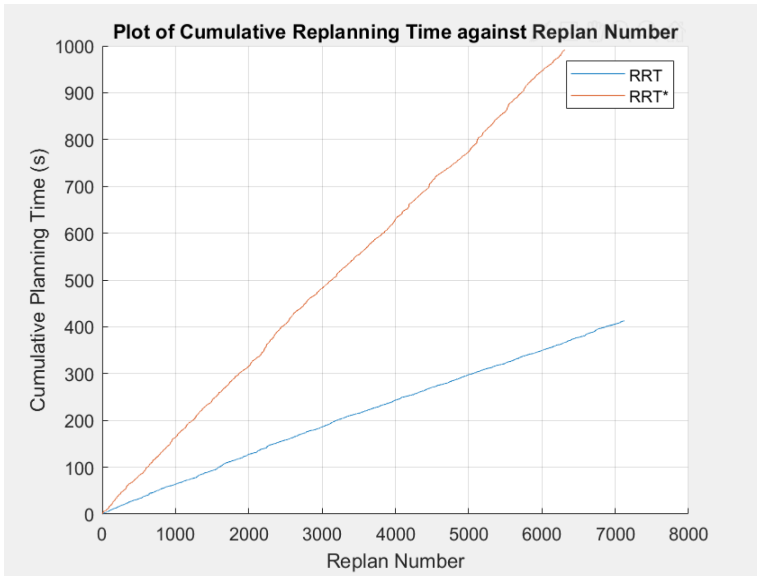


Figure 4-13: Plot of Cumulative Replanning Time against Replan Number

Another important comparison metric between RRT and RRT* based replanning would be the path cost. As seen from Figure 4-14, the average replan path cost for every type of robot-obstacle interaction is higher in RRT based replanning. Furthermore, the range and variance of path cost is much larger in RRT based replanning than RRT* star based replanning. Given that 90.87% of all replanning time falls within the real-time requirement, RRT* is more preferable since RRT replan paths are more jagged and costly.

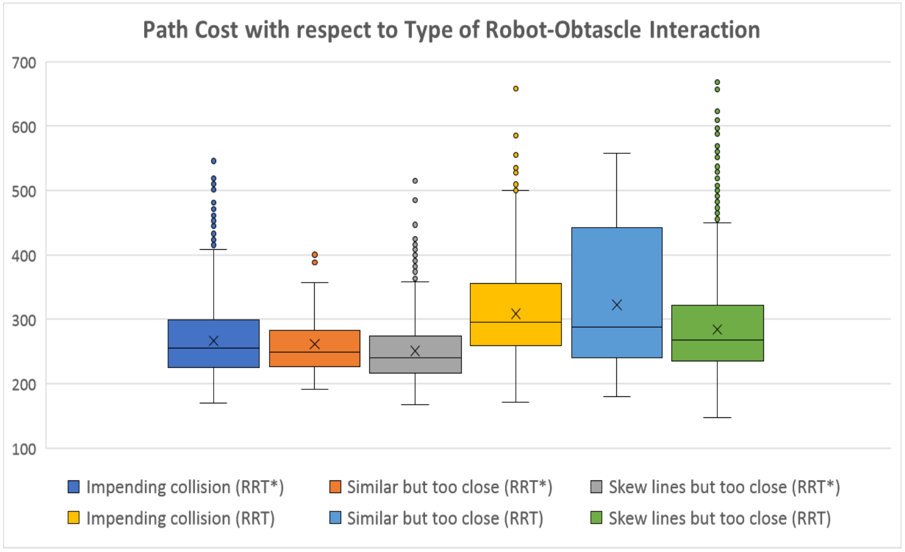


Figure 4-14: Box and Whiskers plot of Path Cost with respect to Type of Robot-Obstacle Interaction

From Figures 4-15 and 4-16, it is interesting to note that a large proportion of long replanning times (above one second) are due to replan events triggered when the robot and obstacle are moving along skew lines at close proximity. On the other hand, replan events caused by an impending collision at a point generally explore more nodes but require much lesser time. This is likely due to the smaller bounding sphere being plotted closer to the robot current position, limiting the free configuration space during replanning. As a result, more nodes are being rejected by the smaller bounding sphere before further calculations are done. A replan event trigger by an impending collision may lead to a replan failure while a replan event caused by skew lines are more likely to exceed the real-time requirements. The issue of replan failure could be solved by allowing the process to sample more points, increasing the chance of success. A strict real-time limit with an escape or safety strategy in case of close proximity could be set to reduce the issues caused by long replanning times.

Given that both RRT* and RRT based replanning are able to satisfy the real-time requirement, optimality becomes a more important factor of comparison. For the application into multi-rotor UAVs, assuming the real-time requirement is met, optimality and smoothness of path are two other important considerations due to limited fuel and flight dynamics constraints. Hence, RRT* based replanning will be a more recommended approach for multi-rotor UAVs. Although the replanning time required for RRT is lower, the additional path smoothing required for RRT may erode its time efficiency advantage.

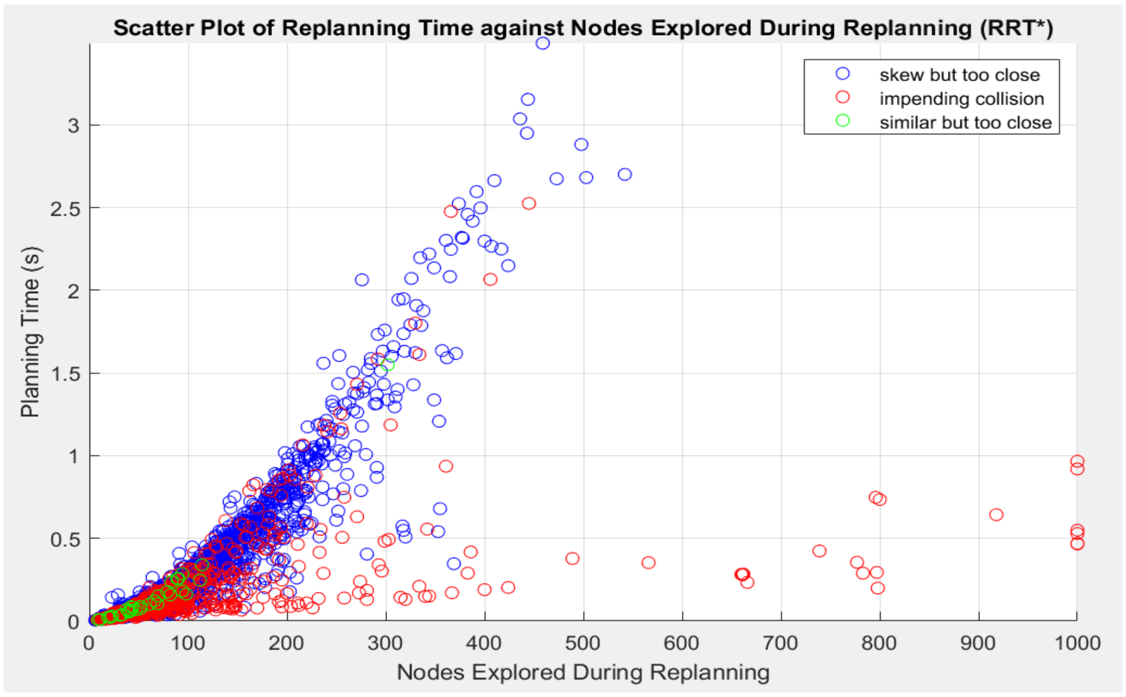


Figure 4-16: Scatter Plot of Replanning Time against Nodes Explored During Replanning (RRT)*

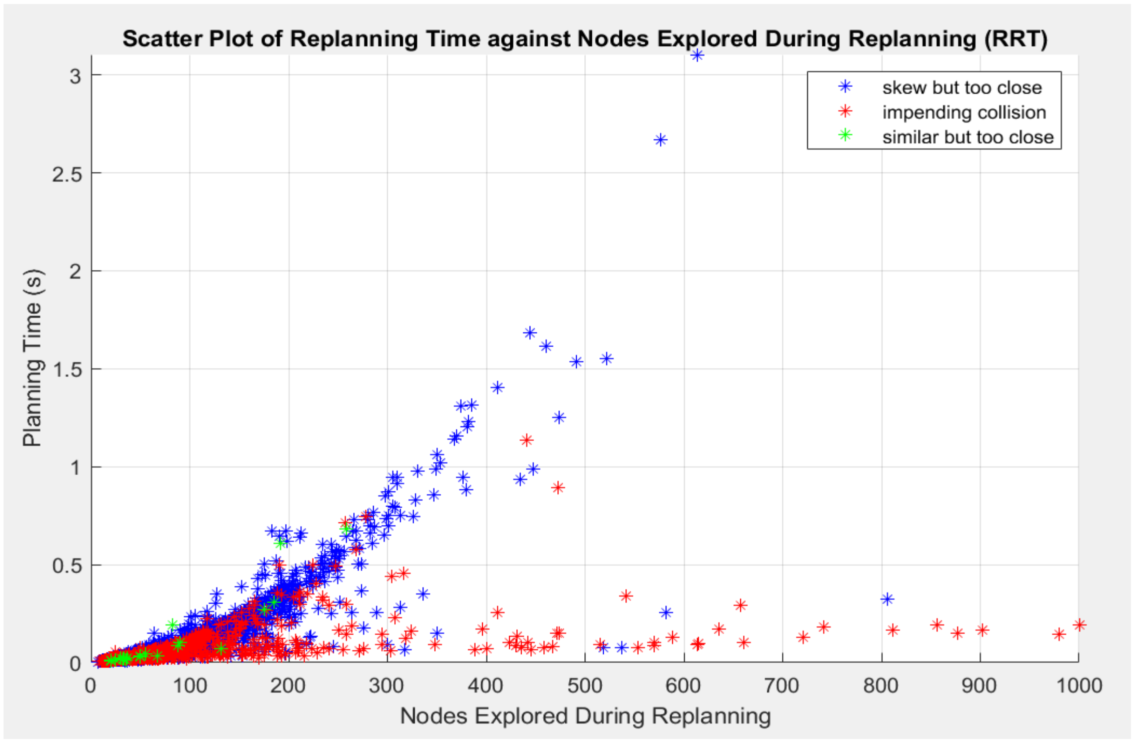


Figure 4-15: Scatter Plot of Replanning Time against Nodes Explored During Replanning (RRT)

4.7 Discussions on the Simulated Replanning Algorithm

The simulation of the replanning algorithm in MATLAB has been largely successful as seen in section 4.6. However, there are several important points that could be discussed further.

1. The predicted obstacle position based on a straight-line trajectory might overlap with the drone's current position, making the impossible to solve the localized path planning problem. However, removing the overlap may result in an inaccurate space-time volume representation of the moving obstacle. Hence, some overlap between the space-time volume representation of the obstacle and robot must be allowed. An exception has to be made such that only the initial branch from the current robot position to the next node is allowed to intersect the space-time volume representation of the moving obstacle.
2. The RRT* path planning algorithm in MATLAB creates a collision-free path for a point particle. It does not consider the size of the robot, making it difficult to completely avoid an incoming obstacle. The size of the obstacle bounding sphere and space-time volume representation were increased during localized path replanning to account for the volume of the mobile robot.
3. There are a limited number of different vertices that the obstacle can randomly select as its destination during the simulation. Given the infinite possibilities of interactions between the robot and the obstacle in 3-D space together with the additional time dimension due to the dynamic motion of both robot and obstacle,

there may be some robot and obstacle interactions that are not represented and verified in the simulation trials.

4. The orientation of the mobile robot during path execution is omitted in the MATLAB simulation but is extremely crucial in an actual multi-rotor UAV simulation due to flight dynamic constraints.
5. In the simulation, the movement of the mobile robot and obstacle are plotted in time slices due to the serial processing nature of MATLAB. When the replan event is triggered, the simulation is temporarily paused, allowing the path replanning algorithm to run. Only after the replanning algorithm has been fully executed, the next time slice is then plotted. However, in real life situations, the processing time for the path replanning is not negligible and should be accounted for as the obstacles will still move or change direction during the path planning time.
6. It is assumed that the drone sensor systems for localization and obstacle detection within a specified radius, are perfect during the simulation. Computation on collision trajectory and replanning are based on data from onboard sensors. Noise disturbances in a real-world environment may compromise the capability of the drone sensor systems to provide good data which will negatively impact the performance of the replanning algorithm.
7. A multi-rotor UAV may not be able to maneuver according to the replan path to avoid a moving obstacle if a sudden change in direction is required. This is due to inertia and other flight dynamics constraints. Further path smoothing technique may be required to remove sharp edges from the replan path.

8. Safety strategies [11] or escape trajectories [29] have not been considered in the dynamic path replanning algorithm in cases where the replan process fails to find a feasible replan path or fails to meet a real-time requirement.
9. There are several other factors that were not part of the project scope and therefore not considered in the MATLAB simulation. First, the multi-rotor UAV or obstacle could undergo a change in speed at any point in time. Second, the obstacle may not travel in straight line paths, making it more challenging to accurately predict its trajectory, Next, multiple obstacles could approach the mult-rotor UAV at the same time. Finally, both the moving obstacle and drone could come in different sizes. The range of obstacle size to drone size ratio was not considered in the simulation.

Chapter 5: Conclusion and Future Work

5.1 Conclusion

This report presented a dynamic path replanning algorithm for a mobile robot to navigate in an environment with both static and moving obstacles. The algorithm was first developed for 2-D configuration space before augmenting it to 3-D implementation. Trajectory prediction of the moving obstacle is the core concept of the algorithm. With the predicted trajectory path of the obstacle and its own estimated trajectory path, the robot is able to dynamically determine its next course of action to prevent a collision. In a 2-D configuration space, trigonometrical concepts together with proximity data are sufficient for the robot to make an informed decision. For the implementation in a 3-D configuration space, 3-D geometry concepts were used. The robot and obstacle velocity vectors are modelled as 3-D lines and the interaction between both lines allows the robot to classify the robot and obstacle interactions into three main categories (moving in similar directions, impending collision, moving along skew lines) to determine the course of action required. Together with the proximity data, this framework enables the mobile robot to make informed decisions in an efficient manner whenever a moving obstacle enters its detection range.

If the robot decides to carry out a replan, the predicted obstacle trajectory path and position also play an important role in the localized RRT* based replanning process. In order to adequately model a moving obstacle during the static RRT* based replanning process, a space-time volume representation was used. The space-time volume was created by sweeping the obstacle along its trajectory path about the predicted obstacle position. Simulation results from 3-D configuration space implementation in section 4.6.4 verified the performance of the dynamic replanning algorithm in generating an effective replan path to avoid the randomly moving obstacle. The final iteration of the dynamic replanning algorithm with RRT* was able to achieve promising results of 99% successful simulation runs without collisions. Thousands of replanning events were triggered during the entire simulation and more that 90% of replans are within an acceptable real-time limit of 400 milliseconds. Given that RRT* is probabilistically complete, there is still a possibility of replan failure where no replan path could be generated. Replan failures occurs at a very low rate, below 0.5%, and its impact could be easily minimised by increasing the detection range to allow for more replanning opportunities.

5.2 Recommendations for Future Work

There are ample opportunities for future work to improve the proposed dynamic path replanning algorithm. First would be the method of the planning and replanning process. The existing path planning process could be switched out for newer and more robust path planning techniques that are capable of planning and replanning in a partially known dynamic environment. Some examples of such techniques are dynamic path planning algorithm are D*-lite, Lifelong Planning A* (LPA*), Execution-extended RRT (ERRT), Dynamic RRT (DRRT), RRT^X and Real-Time RRT* (RT-RRT*). Second, the proposed algorithm is only able to account for one moving obstacle at a time. Additional work could be done to expand the current algorithm to account for multiple moving obstacles at the same time and to observe its performance in such situations. Next, more efforts can be directed to solve various limitations as highlighted in section 4.7 such as using a parallel processing platform for the simulation to determine a more realistic estimate on the effectiveness of the algorithm and the use of safety path trajectory for the robot in cases where replanning fail to meet a real-time requirement. Finally, more research can be done to go beyond the scope of the project and determine the effect of other factors on the performance of the algorithm. Some potential factors for further research would be the consideration of non-constant speed of both obstacle and robot, optimal detection threshold for the robot and the ratio of obstacle size to robot size for an effective replan path.

References

[1] W. T.R, C. H. and R. Earnshaw, "A Motion Constraint Dynamic Path Planning Algorithm for Multi-Agent Simulations," in *The 13-th International Conference in Central Europe on Computer Graphics, Visualization and Computer Vision'2005*, Czech Republic, 2005.

[2] Y. Liang, Q. Juntong, S. Dalei, X. Jizhong, H. Jianda and X. Yong, "Survey of Robot 3D Path Planning Algorithms," *Journal of Control Science and Engineering,* 2016.

[3] A. Giyenko and I. C. Young, "Intelligent Unmanned Aerial Vehicle Platform for Smart Cities," in *016 Joint 8th International Conference on Soft Computing and Intelligent Systems (SCIS) and 17th International Symposium on Advanced Intelligent Systems (ISIS)*, Sapporo, Japan, 2016.

[4] C. Devin and H. M. La, "Extended rapidly exploring random tree-based dynamic path planning and replanning for mobile robots," *International Journal of Advanced Robotic Systems,* pp. 1-15, 2018.

[5] C. Zammit and E.-J. Van Kampen, "Comparison between A* and RRT Algorithms for UAV Path Planning," in *2018 AIAA Guidance, Navigation, and Control Conference*, Kissimmee, Florida, 2018.

[6] N. Iram, K. Amna and H. Zulfiqar, "A comparison of RRT, RRT* and RRT*-Smart Path Planning Algorithms," *IJCSNS International Journal of Computer Science and Network Security, Vol 16, No.10,* Oct 2016.

[7] S. M. LaValle, "Rapidly-exploring random trees: A new tool for path planning," TR 98-11, Computer Science Dept., Iowa State University, Oct 1998.

[8] K. Sertac and E. Frazzoli, "Optimal Kinodynamic Motion Planning using Incremental Sampling-based Methods," in *49th IEEE Conference on Decision and Control*, Atlanta, GA, USA, 2010.

[9] K. Sertac and F. Emilio, "Sampling-based Algorithms for Optimal Motion Planning," *International Journal of Robotics Research,* vol. 30, pp. 846-894, 2011.

[10] J. W. Loeve, "Finding Time-Optimal Trajectories for Resonating Arm using the RRT* Algorithm," Delft University of Technology, Faculty Mechanical, Maritime and Materials Engineering, Delft, Aug 2012.

[11] M. Otte and E. Frazzoli, "RRTX: Asymptotically Optimal Single-Query Sampling-Based Motion Planning with Quick Replanning," *The International Journal of Robotics Research*, vol. 35, no. 7, p. 797–822, 2015.

[12] E. W. Dijkstra, "A note on two problems in connexion with graphs," *Numerische Mathematik*, vol. Volume 1 , no. Issue 1, pp. 269 - 271, December 1959 .

[13] 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, July 1968.

[14] A. Stentz, "Optimal and efficient path planning for partially-known environments," in *Proceedings of the 1994 IEEE International Conference on Robotics and Automation*, San Diego, California, USA, 1994.

[15] N. Kourosh, R. Joose and H. Perttu, "RT-RRT*: a real-time path planning algorithm based on RRT*," in *8th ACM SIGGRAPH Conference on Motion in Games*, New York, 2015.

[16] S. Ghandi and E. Masehian, "Review and taxonomies of assembly and disassembly path planning problems and approaches," *Computer-Aided Design*, Vols. 67 - 68, pp. 58 - 86, 2015.

[17] N. Correll, Introduction to Autonomous Robots, CreateSpace Independent Publishing Platform; 1st edition, 22 September 2014.

[18] D. Devaurs, T. Siméon and J. Cortés, "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, Apr 2016.

[19] J. v. d. Berg, D. Ferguson and J. Kuffner, "Anytime Path Planning and Replanning in Dynamic Environments," in *2006 IEEE International Conference on Robotics and Automation*, Orlando, Florida, USA, 2006.

[20] H. Choset, K. M. Lynch, S. Hutchinson, G. A. Kantor, W. Burgard, L. E. Kavraki and S. Thrun, Principles of Robot Motion: Theory, Algorithms, and Implementations, Cambridge: MIT Press, 2005.

[21] D. Fox, W. Burgard and S. Thrun, "The dynamic window approach to collision avoidance," *IEEE Robotics & Automation Magazine* , vol. 4, no. 1, pp. 23-33, Mar 1997.

[22] J. Bruce and M. Veloso, "Real-time randomized path planning for robot navigation," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, Lausanne, Switzerland, 2002.

[23] D. Ferguson, N. Kalra and A. Stentz, "Replanning with RRTs," in *IEEE International Conference on Robotics and Automation 2006*, Orlando, FL, USA, Jun 2006.

[24] S. Vemprala, "MATLAB Central File Exchange, 2D/3D RRT* algorithm," Jan 2017. [Online]. Available: https://www.mathworks.com/matlabcentral/fileexchange/60993-2d-3d-rrt-algorithm. [Accessed Aug 2018].

[25] D. Legland, "MATLAB Central File Exchange, geom3d v1.22, 3D geometry toolbox," [Online]. Available: https://www.mathworks.com/matlabcentral/fileexchange/24484-geom3d. [Accessed 16 Oct 2018].

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

[27] Z. Chen, M. Lin, S. Li and R. Liu, "Evaluation on Path Planning with a View Towards Application," in *2017 3rd International Conference on Control, Automation and Robotics*, Nagoya, Japan, 2017.

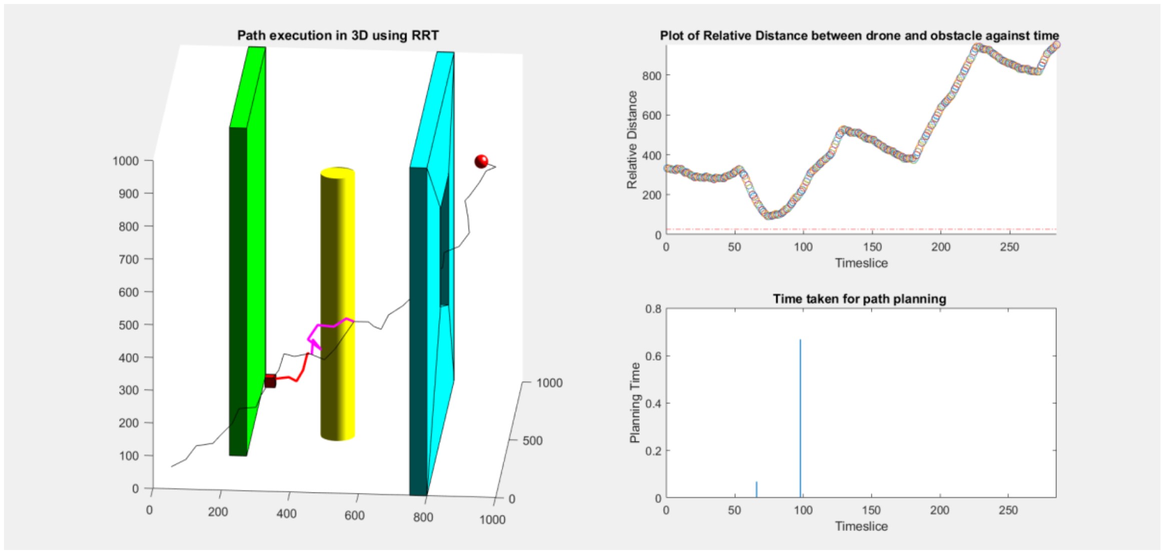
[28] R. J. Kosinski, "Semantic Scholar," 2012. [Online]. Available: https://www.semanticscholar.org/paper/A-Literature-Review-on-Reaction-Time-Kinds-of-Time-Kosinski/2cc72c884223f51be27e0e536d1fb19c1779f513. [Accessed 20 March 2019].

[29] H. David, K. Robert, J.-C. Latombe and S. Rock, "Randomized Kinodynamic Motion Planning," *The International Journal of Robotic Research,* vol. 21, no. 3, pp. 233-255, 2002.

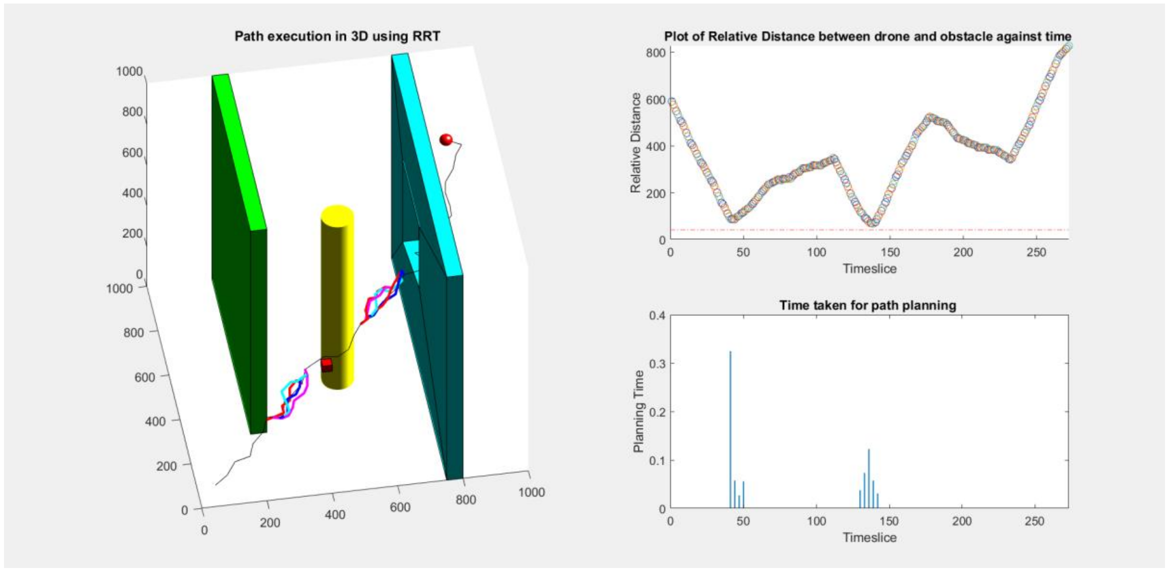
Appendices

Appendix A: MATLAB 3-D Implementation Simulation Interface

Example of simulation interface with RRT replanning.


Path execution in 3D using RRT
Plot of Relative Distance between drone and obstacle against time
Relative Distance
Timeslice
Time taken for path planning
Planning Time
Timeslice

Example of simulation interface with RRT* replanning.


Path execution in 3D using RRT
Plot of Relative Distance between drone and obstacle against time
Relative Distance
Timeslice
Time taken for path planning
Planning Time
Timeslice