ISTANBUL TECHNICAL UNIVERSITY ★ GRADUATE SCHOOL OF SCIENCE ENGINEERING AND TECHNOLOGY

TRAJECTORY BASED UAV COORDINATION

M.Sc. THESIS

Halise TURKMEN

Department of Mechatronics Engineering

Mechatronics Engineering Programme

Thesis Advisor: Assis. Prof. Dr. Emre Koyuncu

MAY 2019

ISTANBUL TECHNICAL UNIVERSITY ★ GRADUATE SCHOOL OF SCIENCE ENGINEERING AND TECHNOLOGY

TRAJECTORY BASED UAV COORDINATION

M.Sc. THESIS

Halise TURKMEN
518161051

Department of Mechatronics Engineering

Mechatronics Engineering Programme

Thesis Advisor: Asst. Prof. Emre Koyuncu

Date of Submission: 03 May 2019

Halise Turkmen, a M.Sc. student of ITU Graduate School of Science Engineering and Technology, student ID 518161051 successfully defended the thesis entitled "TRAJECTORY BASED UAV COORDINATION", which she prepared after fulfilling the requirements specified in the associated legislations, before the jury whose signatures are below.

Thesis Advisor :	**Asst. Prof. Emre KOYUNCU** Istanbul Technical University	…………………..
Jury Members :	**Asst. Prof. Ramazan YENICERI** Istanbul Technical University	…………………..
	Dr. Umut DEMIREZEN Roketsan	…………………..

Date of Submission : 03 May 2019
Date of Defense : 14 June 2019

To my family.

FOREWORD

Firstly, I would like to thank my family for supporting me throughout the duration of my Masters'.

I would also like to thank my thesis advisor Asst. Prof. Emre Koyuncu for guiding me through my research.

I cannot forget my colleagues at the Aerospace Research Center at Istanbul Technical University, whose discussions and inputs provided insight to my research.

Lastly, I would like to thank all of the professors at the Aerospace Research Center, the affiliated institutions who provided me with the opportunity to benefit from their resources for my research and TUBITAK for its scholarship aid.

May 2019 Halise TURKMEN

TABLE OF CONTENTS

Page

ABBREVIATIONS

UAV	**:** Unmanned Aerial Vehicle
PRM	**:** Probabilistic Road Map

SYMBOLS

$\boldsymbol{F_i}$	: Thrust force on rotor i
$\boldsymbol{k_F}$	: Rotor constant for force
$\boldsymbol{\omega_i}$	: Angular velocity of rotor i
$\boldsymbol{M_i}$	: Moment due to rotor i
$\boldsymbol{k_M}$	: Rotor constant for moment
$\boldsymbol{m}$	: Mass of Quadrotor
$\boldsymbol{g}$	: Gravitational acceleration
$\boldsymbol{r_i}$	: Distance of rotor i from the Quadrotor center
F	: Total force
M	: Total Moment
$\boldsymbol{z_B}$	: Unit vector on z – axis on the body frame
${}^{\mathrm{E_I}}\boldsymbol{\omega}^{\mathrm{E_B}}$	: Angular velocity from inertial frame to body frame
$\boldsymbol{R}$	: Rotation matrix from body to world frame
$\boldsymbol{L}$	: Distance between rotors and body frame
$\boldsymbol{p}$	: Angular velocity about the x - axis
$\boldsymbol{q}$	: Angular velocity about the y - axis
$\boldsymbol{r}$	: Angular velocity about the z – axis
$\boldsymbol{\phi}$	: Roll Angle
$\boldsymbol{\theta}$	: Pitch Angle
$\boldsymbol{\psi}$	: Yaw Angle
$\vec{\mathbf{u}}$	: Control input
x	: State vector
$\boldsymbol{C(u)}$	: B-Spline trajectory at u
$\mathbf{P}_i$	: Control Point i
$\boldsymbol{B_{i,n}}$	: Bernstein polynomial
$\boldsymbol{n}$	: Degree of B-Spline curve
$\boldsymbol{N_{i,n}(u)}$	: n^{th} degree B-Spline basis function
$\boldsymbol{U}$	: B-Spline knot vector

LIST OF TABLES

Page

LIST OF FIGURES

Page

TRAJECTORY BASED UAV COORDINATION

SUMMARY

With the increase of availability of technology, Unmanned Aerial Vehicles have made their way from military applications to commercial and civilian use. Many sectors, such as law enforcement, emergency response, agriculture and photography have the potential to heavily rely on the use of UAVs. However, due to regulations and qualifications, civilians are not able to operate specialized commercial drones without some sort of training. Furthermore, most applications in these sectors would require the piloting of a few drones at a time, such as crowd surveillance or missing person searches, which take up resources that might otherwise be better allocated elsewhere. In order to remove these barriers, autonomous UAV swarms could be used. Such a system would only require an operator to define a mission, therefore removing the technology barrier.

In this thesis, an autonomous, decentralized UAV swarm algorithm is designed for navigating through narrow corridors in a known environment with static and dynamic obstacles. A multi start, single goal scenario is assumed, similar to that of emergency response teams searching for survivors in damaged buildings or cave explorations. Possible methods in the literature are mentioned and suitable methods and algorithms are suggested. Similar systems to the suggested one are reviewed.

As a result, a Probabilistic Road Map path planner is used with Dijkstra's shortest path planning algorithm to navigate through the static obstacles. Path smoothening is applied to shorten the path and reduce the number of unnecessary maneuvers. A viable trajectory is generated by passing a cubic B-Spline through the waypoints. Pseudo control points are calculated from the waypoints and are placed appropriately among the waypoints so as to ensure that the trajectory generated by the cubic B-Spline passes through the waypoints. This prevents too much deviation from the path and assures first and second derivative continuity in case the path is changed after a certain waypoint due to new obstacles. Collisions between UAVs are avoided by calculating neighboring UAV's trajectory and adjusting the B-Splines' knot vector, thus adjusting the velocity of the UAV. This is necessary as the application is for a multi start, single goal scenario, hence the assumption that UAV trajectories will converge after a certain point is not unreasonable. Kinodynamic path planning is used if a dynamic obstacle is detected and a collision is expected. The B-Spline knot vector is updated. In this manner, viable trajectories are generated and the UAVs follow the leader in single file.

YÖRÜNGE TABANLI İHA KORDİNASYONU

ÖZET

Teknolojiye erişilebilirliğin artışı ile İnsansız Hava Araçları (İHA) çoğunlukla askeri uygulamalardan sivil ve ticari uygulamalara kadar geniş bir yelpazede kullanılmaya başlamıştır. Askeri uygulamalar olarak başlıca alan tarama, hedef bulma ve takip etme ve saldırı görevleri yer almaktadır. Sivil ve ticari uygulamaları ise ziraat, arama ve kurtarma, afet yönetimi, fotoğrafçılık gibi alanlarda kullanılmaktadır. Ziraat sektöründe İHAlar ekin ve sürülerin uzaktan gözetimi yapılarak zararlı durumlara hızlı müdahale etme imkânı sunar. Arama ve kurtarma ekiplerine ve afet yönetiminde ise hayati malzemelerin ulaşımı, geniş alanların aranması, güvenilir ulaşım yollarının tespit edilmesinde yardımcı olurlar.

Bahsedilen uygulamaların çoğunda İHA eğitilmiş bir personel tarafından uzaktan kontrol edilerek gerçekleştirilir. Bu durum hem İHA kullanımın maliyetini arttırır hem de eğitilmiş personelin bulunmadığı yerlerde İHAlardan faydalanmayı kısıtlamaktadır.

Son zamanların popüler araştırma konusu olan çoklu robotik sistemlerin koordinasyonu İHAlar üzerinde uygulanması ise insan - İHA etkileşimini minimuma indirerek İHA görevlerini otonom hale getirmeyi hedeflemektedir. Çoklu robotik sistemler üç veya daha fazla hücresel veya çeşitli görevlere göre özelleştirilmiş robotların insan müdahalesiz ve birbirleriyle etkileşimli bir şekilde verilen görevleri icra eden sistemlerdir. Çoklu robotik sistemlerde bulunması gerek alt birimler ise görev dağıtıcı, rota planlayıcı, yörünge planlayıcı ve haberleşme olarak listelenebilir. Görev dağıtıcı birimi verilen görevi uygun olan robota veya gerektiği durumlarda görevi robotlara bölen birimdir. Rota planlayıcı ise robotun bulunduğu ortamı haritalandırarak engellerden kaçınarak robotun ilk konumundan görevin tamamlanması gerektiği konuma rota bulur. Nispeten yeni bir alan olan yörünge planlama ise robotun dinamik kısıtlarını da dikkate alarak rota üzerinde robotun takip edebileceği bir yörünge planlamaktadır. Haberleşme birimi ise robotlar arası haberleşmeden sorumludur.

Çoklu İHA sistemlerinde takip edilebilir yörünge üretimi çok araştırılmayan bir konudur. Yapılan uygulamalarda Quadrotor İHAlar için parçalı lineer rotalarda her lineer rota parçasının sonunda İHAlar durmakta ve tekrar başlamakta. Sabit kanat İHAlar için ise minimum dönüş yarıçapı ile linear rota parçasının sonunda İHA dönüş yapmakta. Her iki durum görev başı uçuş süresini arttırarak verimi düşürmektedir. Yörünge üretimi literatürde üç yöntem ile sıklıkla yapılır: B-Spline eğrileri kullanarak, İHAnın dinamik kısıtlarını rota planlama esnasında uygulayarak ve (Quadrotorlar için) diferansiyel düz çıktıları parçalı polinom fonksiyonlar olarak yazıp baz olarak kullanarak minimum snap yörüngeler için optimize eden değerleri bulmak.

Yapılan bu çalışmada deprem gibi bina ve yapılara zarar veren afetler sonrası acil müdahale ekiplerine yardımcı olacak çoklu İHA sistemi için yörünge koordinasyonu sağlanması hedeflenmektedir. Bu senaryolarda binalar insanların giremeyeceği veya güvenliklerini tehdit edecek derecede hasar görmektedir, lakin kurtarma

operasyonlarında aranmaları gerekmektedir. Hasar seviyesine göre binada dar ve yüksek geçitler bulunabilir. Bu durumlarda kara robotları hareket engellerinden dolayı ideal değildir. Çoklu İHA sistemleri kullanılabilir. Dar geçitlerden tek şerit halinde ve atik bir şekilde geçmeleri gerekmektedir. Bunun için en büyük engel rota bulunduktan sonra İHAların bu rotayı takip etme sırasında birbirleriyle çarpışmalarıdır. Bunu engellemek için İHA yörüngelerini aralarında paylaşılması ve İHAların birbirleriyle çarpışmayı önleyecek rotalar bulmalı. Kablosuz haberleşme kısıtlarından dolayı İHA ların yörünge paylaşımları aksaklığa uğraya bilir, transfer edilen verilerin bütünlüğünü sağlamak ise veri sayısı arttıkça zorlaşmaktadır. Ayrıca, tekrar tekrar rota ve yörünge hesaplamak verimli değildir. Bu sebeplerden dolayı geliştirilen algoritma kriterlerinde veri transferini minimuma indirmek ve tek seferde rota hesaplamak bulunmaktadır.

Bu koşulu sağlayan çoklu İHA sistemlerde ve üç boyutlu ortamlarda kullanılmak üzere bir rota ve yörünge planlama algoritması geliştirilmiştir. Sistemdeki İHAlar ortak bir Rastlantısal Yol Haritasını (Probabilistic Road Map – PRM) kullanılarak bilinen engellere çarpmadan bulundukları konumdan hedef konuma doğru bir rota bulur. Bulunan rotaların konum noktalarından geçen bir kübik B-Spline eğrisi oluşturulur. Ortak PRM kullanıldığından dolayı bulunan rotaların ve dolayısı ile yörüngelerin hedefe yaklaştıkça birleşmesi beklenmektedir. Çoğu durumda bu durum gerçekleşir. Yörüngelerin birleşme noktası veya ortak noktası üzerinde İHAların birbirleriyle çarpışma durumu olursa tespit edilir. Bu durumlarda B-Spline'ların parametreleri optimize edilerek yörüngelere 'zamanda ölçeklendirme' işlemi uygulanır. İHAların yörüngeleri değişmez iken bu yöntem sayesinde yörüngelerin birinci ve ikinci türevleri, yani hız ve ivmeleri, değişir ve çarpışmalar önlenir. Yeni bir engelin tanımlanması veya tespit edilmesi durumunda yörüngeler olası çakışmalar için kontrol edilir ve PRM güncellenir. Orijinal rotada çakışmadan önce ve sonra gelen konum noktaları tespit edilir ve engeli atlatan yeni bir rota bulunur, yörüngeye eklenir.

PRM ilk defa 90larda robotların iki boyutlu düzlemde rota bulmaları ve çok boyutlu robot (örneğin robotik kollar) eklemlerinin engellere çarpmayan bir hareket dizisi bulması için kullanılmıştır. Bu algoritmanın en büyük avantajları rastgele örnekleme yapmalarından dolayı ortamı eşit bir şekilde kontrol etmeleri ve örnekleme sayısına göre çok boyutlu uygulamalarda diğer yöntemlerden daha az işlem gerektirdiğidir. Algoritma iki süreçten oluşur: Öğrenme süreci ve Sorgulama süreci. Öğrenme sürecinde ortam rastgele örneklenir, engellerin içinde olamayan örnekler nod olarak saklanır. En yakın n nod doğrular ile birleştirilmeye çalışılır. Eğer nodlar arasında bir engel bulunmuyor ise birleştirilirler. Bu şekilde harita oluşturulur. Sorgulama sürecinde ise başlangıç ve hedef nokta verilir. Bu noktalar geçici olarak en yakın nodlara doğrular ile birleştirilir. Daha sonra buluşsal rota planlama yöntemleri ile başlangıç noktasından hedef noktasına giden nodlar üzerinden rota bulunur.

B-Spline eğrileri tasarım ve animasyon alanında az sayıda parametre ile eğri ve yüzeyleri tanımlama özelliklerinden dolayı sıklıkla kullanılır. Bu parametreler kontrol noktaları, knot vektörü ve b-spline nın derecesi olmaktadır. Kontrol noktaları eğrinin şeklini etkiler. Eğri sadece belli şartlar sağlanırsa kontrol noktaların üzerinden geçer, ama her zaman kontrol noktalarını takip eder. Knot vektörü ise artan gerçek değerlerden oluşmaktadır. Knot vektöründe eleman sayısı b-spline nın derecesi ve kontrol nokta sayısına bağlı olarak değişiklik gösterir. B-spline nın derecesi bir eğri üretmek için gerekli minimum kontrol nokta sayısını belirler. Aynı zamanda, b-pline nın derecesi arttıkça oluşan eğri veya yüzey kontrol noktalarından uzaklaşır ve eğri türevlerin derecesi artar. n. dereceden bir b-spline eğrisinin n-1. Dereceye kadar türevi süreklidir.

Bu şekilde Quadrotor İHAların çoklu sistem uygulamalarında yörünge manipülasyonu ile koordinasyonu sağlanmıştır.

1. INTRODUCTION

Unmanned Aerial Vehicles (UAVs) have been a hot topic in both the academic field and in the private sector for the past two decades. Initially being used in the military for reconnaissance and attack missions, UAVs now have a wide variety of applications in the civil sector as well – in photography, emergency response, and agriculture, to name a few.

UAV drones have gained popularity in the emergency response services. This is largely due to their ability to reach remote or otherwise dangerous places. Delivery of medical supplies after natural disasters by UAVs take a relatively shorter amount of time, especially in cases of obstructed roads, thus increasing the chances of saving lives (Judy E. Scott, 2017). Search and rescue teams are able to survey areas and situations faster, saving time and resources. Law enforcement and firemen alike use UAVs for surveillance of crowds and burning structures respectively, considerably reducing expended manpower and allowing them to prevent dangerous situations (Overwatch, 2017; Karpowics, 2017).

Likewise, the agriculture sector has also adopted the use of UAVs to increase crop production and estimate yield. According to a report published by the Food and Agricultural Organization of the United Nations (Food and Agriculture Organization of the United Nations, 2018), UAV surveillance of farmland is used farmers to identify and correct irrigation problems, pest and fungal infections, and other problems. Insurance companies also use such information to accurately predict the crops worth.

However, the widespread use of such specialized UAVs is relatively hindered by the cost of not only the UAVs themselves, but also the piloting, training, and/or software required. As most countries rightly have strict regulations on civilian operated UAVs in urban area, most customers either have to get a certified license, permission from the local authority, or hire a trained operator.

Such problems are only multiplied when considering applications in larger areas and with more than one UAV – such as surveying of farmland – thus generating the need for UAV swarms and autonomy. Swarm robotics entered the academic field the late eighties with Beni detailing the conditions of Cellular Robotic Systems (Beni, 1988). He later went on to address the nature of their interactions and behaviors as Swarm Intelligence, defining it as the method of manifesting of intelligent behavior of a collection of robots through interaction with each other and the environment (Gerardo Beni, 1989). Şahin, 2004 expanded this definition by adding that the robots should be simple and unitary and should act towards a certain goal (Erol Şahin, 2004). He goes on to stress the autonomous nature of the robots and their number in the swarm.

Autonomy in swarms entails task division between swarm members, finding collision free paths in the given environments, generating feasible trajectories from said paths and completing the given tasks without human interference. This thesis will focus on realizing these objectives in an optimal manner. The rest of the thesis is organized as follows: Section 1 will declare the purpose of the thesis, explore the relevant literature and suggest a solution, Section 2 will give information on the methods used and their integration with each other, Section 3 will discuss the experiments and results and Section 4 will consist of the conclusions and consider future works.

1.1 Purpose of Thesis

As discussed previously, the demand for UAV swarms in the emergency response field is high – though hindered by lack of autonomy, especially in the case of very specific scenarios – such as searching for survivors in demolished buildings. The most common natural disasters leave destroyed infrastructure in their wake, which are unstable and the search for survivors is tedious work. One needs to be able to traverse narrow and partially obstructed passages, a painless task for UAVs. Therefore, this thesis aims to develop algorithms to generate feasible and changeable trajectories through narrow corridors for a UAV swarm while taking into account static and dynamic obstacles.

The proposed algorithms is as follows. Each UAV finds its own path and calculates its own trajectory. A Probabilistic Road Map is generated once. Start and goal positions of queries are added to the PRM and the shortest path is found using Dijkstra's Shortest Path Algorithm. Obstacles in the environment and samples for the PRM are stored in a KD-Tree, which simplifies nearest neighbor and collision checks. Next, a spline trajectory is generated, and the spline parameters are shared with nearby UAVs and a priority ID is assigned in the event of possible collisions. The UAV with lower priority ID adjusts its velocity to prevent collisions. For dynamic obstacles, the UAV presumes it has a linear trajectory and adjusts its velocity to avoid it. If new stationary obstacles are detected on the UAV path, the nodes of the PRM falling in the obstacles are deleted and a new PRM is generated. The nearest node to the collision and the node after the collision is found and a path is generated to avoid the obstacle. This path is then added to the original path.

1.2 Literature Review

Robotic swarm coordination and control have been a hot topic in academic research since the term was first coined. The development of UAVs has only increased the popularity of swarm and path planning algorithms. From surveying the literature, it would be appropriate to classify main objectives of autonomous UAVs within swarms as finding the shortest viable path, the maximum coverage path, and searching for targets. Due to the purpose of this thesis, research on finding the shortest viable paths will be focused on. Among the numerous literature, current and relevant works and algorithms will be presented. First, the important methods in the literature will be briefly argued. Then, solutions presented in the literature to the path planning problem will be presented. Where possible, the environment representation, path finding and trajectory generation methods of the solutions will be evaluated.

1.2.1 Overview of available methods

There are many algorithms in the literature which can be applied to path planning problems. Environment representations for path planning can be divided into three groups: the roadmap approach, the cell decomposition approach, the grid based approach and the potential field approach (Pehlivanoglu, 2012).

The Roadmap approach includes the Probabilistic Road Map (PRM) – where the environment is sampled randomly and edges are formed between neighboring samples if there are no obstructions (L. E. Kavraki, 1996; N. M. Amato, 1996) – and Voronoi Partitioning – where nodes in the environment are either defined or sampled and edges are formed equidistant to two neighboring nodes (Aurenhammer, 1991; Voronoi, 1908). Both are frequently used in path planning problems along with an optimization based or a graph based search algorithm (Liwei Huang, 2016; Sai Chen, 2017; Hrabar, 2008; Debnath S.K., 2018; Mehmet Korkmaz, 2018). The PRM is especially suited for

3D environments as its computational burden relies on the number of samples chosen rather than the environment dimensions. In the case of Voronoi partitioning, 3D environments bring about separating planes rather than separating edges.

The Cell Decomposition approach reduces the environment into labeled cells. The cells are either labeled by their vertices and edges or by their (*x, y, z*) positions. These regions are then marked according to whether they contain obstacles or not (Seda, 2007). The cell size and shape do not need to be identical. Afterwards, an optimization algorithm is generally used to find a path that minimizes a cost function representing the objectives of the mission (P. B. Sujit, 2007). For the Cell Decomposition method, modeling a 3D environment may bring about complex 3D shapes or many maps representing different altitudes, both of which would increase the number of computations.

Grid based approach is when a grid is superimposed on the environment. The areas corresponding to obstacles, start, goal and free regions are generally marked uniquely. The regions in the grid are identical in terms of shape and size and must be chosen carefully, as too large regions leads to a coarse grid and therefore loss of information, and too small regions requires more computational power. With grid based environment representations, the direction of motion is limited to the number of edges adjacent to one grid, therefore randomized path planners cannot be used and the search is performed such that regions surrounding the current region are explored first (Yap, 2002). Similar to the Cell Decomposition method, representing a 3D environment would also increase the dimensions of the grid used.

The artificial potential field approach is similar to the grid based approach as it also involves superimposing a grid on the environment. However, in this method, each region in the grid contains a force value which represents the cost of moving to that region. The value is generated by repulsive and attractive forces. Repulsive forces compel the UAV to move away from a region and attractive forces compel the UAV to move towards a region. Depending on the problem statement, repulsive forces may represent distance to an obstacle or no flight zone and increase as the regions approach it and attractive forces may represent the distance to the goal and decrease as the regions approach the goal region. In such a scenario, a gradient descent algorithm could be used to follow the adjacent regions to the current region with the minimum force value (YongBo Chen, 2015; Yaohong Qu, 2015). A drawback of this method is that it is susceptible to local minima, although several improvements have been made to overcome this (Warren, 1989; P. Vadakkepat, 2000). Artificial potential fields have also been applied to dynamic motion planning problems in 2D by adding time to the dimensions (K. Fujimura, 1989) and including the relative positions and velocities of the dynamic target and obstacle in the attractive and repulsive forces respectively (S. S. Ge, 2002) to name a few. The main drawback of these methods is the computational burden of recalculating the artificial potential field matrix at each cycle. This drawback was reduced by the use of General Purpose Graphical Processing Units to accelerate a parallel APF computation algorithm to find real time paths in an unknown environment in (Omer Cetin, 2016).

There are many algorithms used in path planning. They are divided by Yang et al. to five categories: Sampling Based Algorithms, Node Based Optimal Algorithms, Mathematical Model Based Algorithms, Bio-inspired Algorithms and Multi-fusion Based Algorithms (Liang Yang, 2016). The algorithms will be discussed under these categories.

The sampling based algorithms were mostly discussed in the previous paragraphs, with the exception of the Rapidly-Exploring Random Tree path planning algorithm and its variations. The main idea behind the RRT algorithm is randomly sampling the environment and connecting the sampled point to the nearest point present, given that there is a collision free path between them. This is done in a loop until the goal position is found or the maximum number of points has been reached. Like other sampling based algorithms, the RRT algorithm does not guarantee finding the optimal path and it is probabilistically complete.

Node Based Optimal Algorithms are called as such because they evaluate the nodes' position and weight information within a cost function in order to find the optimal path. Such algorithms include Dijkstra's shortest path algorithm, A* algorithm and D* algorithm. These algorithms use grids or nodes to represent the configuration space.

Mathematical Model Based Algorithms only consider the accelerations and velocity constraints of the point robot. Because they consider the dynamics of the robot in order to find the optimum path, they are also considered by some to be trajectory generation algorithms. Due to the fact that robot states and environment variables have to be defined at all times, they can produce better results in terms of shortest path and obstacle evasion in highly cluttered environments. However, since most robot models are complex, these algorithms have a high computational cost.

Bio-inspired Algorithms are those which are inspired by biological systems or interactions of species with each other and their surroundings. The aim of Bio-inspired Algorithms is for robots to be able to accomplish their missions without human interaction or supervision. These algorithms can further be divided into two: Evolutionary Algorithms and Neural Network Algorithm. Evolutionary algorithms are designed by observing the behavior and interactions of a certain species, e.g.: ants for Ant Colony Optimization Algorithm, grey wolves for Grey Wolf Optimization Algorithm, etc. Neural Network Algorithms resemble the workings of biological neurons to process information. The efficiency of Bio-inspired Algorithms is largely dependent on the optimizing rules and the models proposed. Complex models and cluttered environments can greatly reduce the performance of these algorithms.

1.2.2 Literature survey

Judd et al. proposed a semi on-line path planning and trajectory generation with the use of a Voronoi diagram with two edge costs - threat costs and fuel costs (Kevin Judd, 2001). They modeled the obstacles as points for the creation of the Voronoi diagram. They defined the threat cost as the sum of inverses of the distance of three points on the edge from the threat weighted by the length of the edge. The three points were chosen at one sixth, half, and five-sixths of the total length of the edge. The fuel cost is proportional to the length of the edge under the assumption that the UAV flies at constant speed. They found the total cost as a weighted sum of the costs mentioned above. The weights in this sum were used to allow the prioritization of shorter paths or less exposure to threats based on the user's wishes. For path queries, the start and goal locations are connected to the nearest three nodes. Then, the final path was found by running Dijkstra's Shortest Path algorithm and smoothened by joining the edges on the path with cubic splines. The splines are generated from the midpoint of an edge to the midpoint of the edge directly after it in the path. The intersection of the two consecutive edges is taken as a temporary control point. Judd et al. finds the knots for the spline through an optimization process by enforcing C^0, C^1, and C^2 continuity and the condition to pass through the beginning and end control points. The start and goal locations are connected to the rest of the spline trajectory in a specific manner. For the joining the start location, the initial heading of the UAV is specified. The end of the spline is set to be the midpoint of the next edge and the heading to be tangent to it. Then, the midpoint is optimized for minimal threat exposure with maximum curvature constraints arising from the UAV's dynamic constraints. Joining the goal location is done similarly. The splines generated for the other edges are optimized by minimizing the distance of the spline from the original path while keeping in mind the curvature constraints. Because this is a repetitive and computationally taxing process, Judd et al. defined new variables describing the location of the middle control point. These variables are related to the length of the edges and the angle between them. The values of these variables are numerically computed beforehand and store in a look-up table, thereby reducing computations during flight. If there are no corresponding values to specific edge lengths and angle, the algorithm assumes that there are no feasible paths. Because the spline trajectories for edges near to the UAV are calculated first, if the UAV detects new obstacles further away, the procedure is followed from the beginning except for taking the end of the current spline trajectory as the starting point. Therefore, by the time the UAV reaches the end of the original segment, the first segment of the new path is calculated and the UAV successfully avoids the obstacles. Judd et. al. reported that it took 1.3 seconds to calculate the new trajectory.

Wzorek et. al. compared a combination of PRM with A* search algorithm and RRT to realize a fully autonomous UAV (Mariusz Wzorek, 2006). They integrated the control kernel with the motion planners in order to produce dynamically feasible paths. When building the PRM, they joined collision free nodes with cubic curves, thus smoothing the flight. If a cubic curve cannot be used due to obstacles, then the UAV is forced to fly these segments as they are. At the query stage of the PRM, additional constraints such as altitude requirements, obstacles, no-fly zones, velocity limits, etc. can be implemented and dealt with by the A* search algorithm. They found that the mean elapsed time for planning is below 1000 ms - including addition of new constraints. After the offline map generation stage, the algorithm proposed is as follows. The query is made and if a suitable path is found, each trajectory segment is defined via its start and goal locations, headings and velocities and sent to the controller, where it is executed. Once completed, the new segment is sent. To avoid collisions with dynamic obstacles, Wzorek et. al. defines four strategies which can be called upon depending on the parameters of the detected collision – re-planning from the next waypoint (the beginning of the next trajectory segment), re-planning from the last collision free waypoint, re-planning for only the trajectory segments with collisions, methods to improve the re-planning. The strategy is chosen by the strategy selector, if more than one strategy is selected, it also chooses the optimal one. They recorded the results of running RRT and PRM and using Strategy 1 and Strategy 3 for different number of randomly added obstacles during the flight. They found that while re-planning with Strategy 3, the maximum re-planning time is shorter for both RRT and PRM. However, there are more segments in the path than for re-planning with Strategy 1, which lowers the velocity of the UAV.

A combination of algorithms is introduced by Amin et. al. for offline preplanning and online re-planning of paths (Jayesh N. Amin, 2006). They implemented their algorithm for 2D and 3D space representations. They used a Region Quad-tree data structure to store obstacles. This is a hierarchical data structure which recursively divides spatial regions into quadrants while retaining symmetry among the new regions. They assumed that the location of all the obstacles are not known to the UAVs until they are within a certain proximity and that all obstacles, known and unknown are stationary. For the path planning, they used a Dual RRT algorithm, where two trees are expanded simultaneously - one from the start position and one from the goal position - with a distance estimator instead of the Euclidean distance to reduce computation. The path found by the RRT algorithm is not always optimal in terms of minimal distance, therefore Amin et. al. applied Dijkstra’s Shortest Path algorithm to remove unnecessary nodes. They applied this method to 3-D space and found that the resulting path in most cases flew over the obstacles to the goal position instead of around them. To prevent this they added a penalty for high altitudes in the distance approximation function for the RRT algorithm. Corridor constraints were added to ensure that the path segments are flyable by the UAV. They implemented their algorithms as C-libraries and ran simulations in Matlab. The results of their experiments show that their algorithm was successfully able to find offline paths as well as re-plan them online when they sensed static obstacles.

Lopez et. al. presented a real-time 2D path planner for UAVs in dynamic environments (Jose Luis Sanchez-Lopez, 2017). They modelled the environment and obstacles as a set of geometric primitives so as to reduce loss of resolution in other grid or cell based representations. Dynamic obstacles were also modelled as geometric primitives with accounting for their heading in the shape of the representation. They are also considered temporary. Only dynamic obstacles within a threshold proximity to the UAV are modelled. A PRM is then constructed, but collisions with dynamic obstacles are not taken in to account. This is done during the query stage to allow dynamic mapping. A Potential Field Map is used as a cost function where the weighted sum of distance to the goal and distance from obstacles is used. The cost of traversing a segment is the line integral along that curve, which is found using the potential field map. A* is then used to find a probabilistically optimal collision free path. A shortening algorithm is applied to remove redundant waypoints and velocity and acceleration planning is done by joining corners and sharp turns with circumference arcs where possible. If a new static obstacle is defined or a dynamic obstacle is detected, the potential field map is updated and the cost of the current path is checked. If it is below a threshold, the path is still viable. Otherwise, a new path is generated. Simulation results for a single UAV in highly cluttered environment without prior knowledge of obstacles show that the algorithm is able to find a collision free path to the goal without falling in to local minima. A simulation of a search and rescue mission involving a swarm of 3 UAVs shows that the UAVs are able to successfully carry out their missions while avoiding collisions with each other and obstacles in the environment.

Xue et. al. proposed an Improved PRM used with D* lite heuristic search algorithm for offline and online path planning (Qian Xue, 2014). They compute a “failure weight” for nodes in the PRM corresponding to the number of neighbors a node fails to connect to. In the learning stage of the PRM, they only sample from a circle with the center being the current point. By doing this, they reduce the computational effort in finding nearest neighbors among all of the samples. The best edge fitting the criteria is stored and the new center becomes the other vertex of the edge. The radius is chosen with respect of the distance to the nearest obstacle, therefore guaranteeing that there will be no collisions within the circle. This is done until the goal position is reached. For sampling in 3D, the environment altitude is divided and the radius of the obstacle at the sampled altitude is considered when calculating the radius of the safety circle. Through simulations using the standard PRM and the Improved PRM, Xue et. al. shows that IPRM finds shorter paths in a significantly shorter amount of time than PRM. Furthermore, IPRM is maintains stability in its results when the number of obstacles is changed. If obstacles are detected on the current path, the process is repeated to re-plan the path from the current location of the UAV.

Yan et. al. proposes a 3D UAV path planner using PRM and the A* Search algorithm (Fei Yan, 2013). The main contributions of their work is the use of Octrees for modelling the 3D environment and incorporating overlapping bounding boxes to find paths through narrow corridors. Nodes in an Octree represents 3D volumes called voxels. The environment is recursively divided into smaller voxels if an obstacle is present in it. To ensure that the UAV can pass through the paths, the smallest voxel is larger than the dimensions of the UAV. Bounding boxes are sub-regions containing voxels of different sizes and are labelled as empty, full, or mixed. Empty bounding boxes contain only empty voxels, therefore they do not contain any obstacles and are sampled randomly. Full bounding boxes have full voxels and therefore are not sampled. Mixed bounding boxes contain both empty and full voxels and represent the boundaries between the environment and the obstacles, including narrow corridor regions. In these boxes, the sampling density is increased with respect to the density of full voxels in order to increase the chances of finding a path. Compared with the random sampling method, Yan et. al. has shown that this method guarantees finding a path through narrow corridors if there is one, whereas the random sampling does not. Instead of connecting *k*-nearest neighbors in the bounding box, the connectivity between neighboring voxels are evaluated. This is done by checking if they share a surface. If a free voxel shares a surface with another free voxel, there is a path between them. In this way, connectivity between the sampled nodes is checked and a path is formed by joining the centers of all of the voxels in between them. Local paths found in the bounding boxes are joined by defining another bounding box overlapping the voxels lining the shared surface of the original bounding boxes. Thus, connectivity between bounding boxes is satisfied. Once the road map is generated, a path is found using A* search algorithm. Yan et. al. performed indoor and outdoor tests using a laser scanner to sense the obstacles proving their algorithm's ability to find a path in unknown cluttered environments.

Farooq et. al. used PRM to generate paths for UAV swarms in the presence of obstacles - the dangerous zone - and in the absence of obstacles - the safe zone (Muhammad Umer Farooq, 2017). In the safe zone, UAVs fly in formation with same velocities and distances between the nearest UAV. In the dangerous zone - when obstacles are detected - each UAV generates a PRM and finds an obstacle free path with time series to avoid collision with other UAVs. They performed simulations for evenly spaced, rectangular obstacles and randomly placed, irregular shaped obstacles. The results showed that the UAVs break formation in the dangerous zone and find collision free paths. Once they reach the safe zone they fall back into formation.

Petersson et. al. proposed a path planning algorithm for autonomous UAV surveillance of urban areas (Per Olof Pettersson, 2005). They made some modifications to the standard PRM algorithm to allow for dynamic constraints. UAV configurations are samples in the free space and are connected with splines if there are no obstacles present. The spline connections are made during the query phase and take into account new constraints. Obstacles are represented in an OBB Tree. Collisions are checked for by checking for intersections of the spline with an OBB. The path is refined by removing redundant nodes. They constrained the parameters of the splines so as to make them C^1 continuous.

2. METHOD DEVELOPMENT

In this section, the relevant theories and algorithms are explained in detail.

2.1 Differential Flatness and the Quadrotor Model

A system is said to be differentially flat if there exists a set of output variables (flat outputs) from which all states and actions can be calculated without integration. This permits the control of non-linear model of the Quadrotor. Using the differential flatness property of the Quadrotor, it is possible to check the feasibility of the calculated trajectory and modify it to be follow able by the Quadrotor. This is done by putting bounds on the derivatives of the flat outputs.

In this section, the derivation of the Quadrotor model is explained.

2.1.1 Quadrotor Model

Figure 2.1 shows the free body diagram of a Quadrotor with the Thrust forces, F_i, angular velocity of the rotors, ω_i, and the body and world frame origins, E_B and E_I. The rotational velocity around the x, y, z axis of the body frame are also shown as p, q, r (for ϕ, θ, ψ Euler angles) respectively.

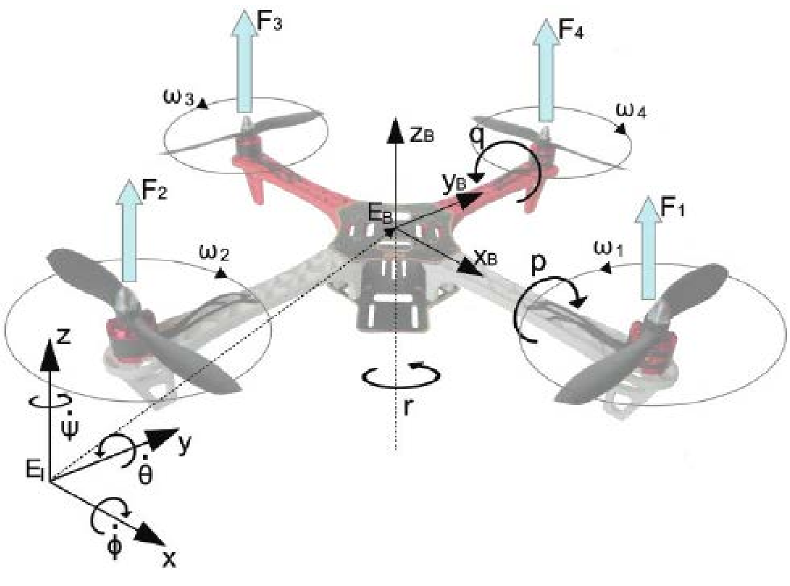

Figure 2.1: Free body diagram of a Quadrotor (Anežka Chovancová, 2014).

The force and moment equations are given in Eq. (2.1) and (2.1), where k_F and k_M are taken as constants dependent on the propeller properties.

$$F_i = k_F \omega_i^2 \tag{2.1}$$

$$M_i = k_M \omega_i^2 \tag{2.1}$$

Therefore the net force and moment is as in Eq. (2.2) and (2.3). Here, m is the mass of the Quadrotor, g is the gravitational acceleration and r_i is the distance from the rotors to the center of the body axis.

$$\mathbf{F} = \sum F_i - mg\boldsymbol{z_B} \tag{2.2}$$

$$\mathbf{M} = \sum (r_i \times F_i) + \sum M_i \tag{2.3}$$

The angular velocity ω can be written in the body frame as given in Eq. (2.4) and Euler's rotation equation is given in Eq. (2.5).

$${}^{\mathrm{E_I}}\omega^{\mathrm{E_B}} = px_B + qy_B + rz_B \tag{2.4}$$

$$M = \mathrm{I}\dot{\omega} + \omega \times (\mathrm{I}\omega) \tag{2.5}$$

Using these along with Eq. (2.2) and (2.3), the Newton-Euler equations are then found as Eq. (2.6) and (2.7), where R the rotation matrix from body to world frame and L is the length between the rotors and the body frame, taken to be equal for all of the rotors.

$$m\ddot{\mathbf{r}} = \begin{bmatrix} 0 \\ 0 \\ -mg \end{bmatrix} + \boldsymbol{R} \begin{bmatrix} 0 \\ 0 \\ \sum F_i \end{bmatrix} \tag{2.6}$$

$$\mathrm{I} \begin{bmatrix} \dot{p} \\ \dot{q} \\ \dot{r} \end{bmatrix} = \begin{bmatrix} L(F_2 - F_4) \\ L(F_3 - F_1) \\ M_1 - M_2 + M_3 - M_4 \end{bmatrix} - \begin{bmatrix} p \\ q \\ r \end{bmatrix} \times \mathrm{I} \begin{bmatrix} p \\ q \\ r \end{bmatrix} \tag{2.7}$$

After simplifying Eq. (2.7) and from Eq. (2.6), we can define the control inputs in terms of angular velocity as in (2.8). The states are given in (2.9) and are the coordinates, the orientation, the velocity, and the angular velocity of Quadrotor in the world coordinate system.

Given the flat outputs, $\sigma = [x, y, z, \psi]^T$, the Quadrotor dynamics are shown to be differentially flat in (Daniel Mellinger, 2011).

$$\vec{\mathbf{u}} = \begin{bmatrix} \mathrm{u}_1 \\ \mathrm{u}_2 \\ \mathrm{u}_3 \\ \mathrm{u}_4 \end{bmatrix} = \begin{bmatrix} k_F & k_F & k_F & k_F \\ 0 & k_F L & 0 & -k_F L \\ -k_F L & 0 & k_F L & 0 \\ k_M & -k_M & k_M & -k_M \end{bmatrix} \begin{bmatrix} \omega_1^2 \\ \omega_2^2 \\ \omega_3^2 \\ \omega_4^2 \end{bmatrix} \tag{2.8}$$

$$\mathbf{x} = [x, y, z, \phi, \theta, \psi, \dot{x}, \dot{y}, \dot{z}, p, q, r]^T \tag{2.9}$$

2.2 KD-Trees

KD-Trees or "k" dimensional trees were first introduced by Bentley in 1975 (Bentley, 1987) as a multidimensional binary search tree in order to store and query for data. It has gained popularity in artificial intelligence applications due to the efficiency of specific searches, such as for the nearest neighbors over the original K Nearest Neighbor algorithm, which relied on a brute force search.

A KD-Tree is essentially a binary tree where the leaves are *k* dimensional data points. Every node has two pointers, either null or pointing to another node in the tree. In the case of constructing binary trees, data to the left of the root node form the left sub-tree and data to the right form the right sub-tree. For KD-Trees, however, the dividing nodes can be thought of as dividing hyper planes on a chosen axis that split the space into two sub-spaces. All the following points are placed based on their values on the chosen axis. If the new node has the same axis value as the root node, then their values on the next axis are compared and the new point is placed accordingly.

A conventional KD-Tree is constructed by sequentially choosing the splitting plane axis. For example, to meet the purpose of this thesis, a three dimensional KD-Tree needs to be constructed. Therefore, the first hyper plane axis would be the *x* axis and the children of the root node would be divided based on their x value. The second hyper plane would be the *y* axis and the children of these nodes would be divided based on their y values, and so on. Assuming that all the points for the tree are given at once and not fed one by one, the next point to be placed is selected by choosing the median of all the points that go in the sub-tree. Table 2.1 contains randomly generated x, y, and *z* values and Figure 2.2 shows the KD-Tree constructed from them. If the KD-Tree were to be plotted in a three dimensional graph, the result would be similar to the results shown in Figure 2.3.

Table 2-1: Randomly generated x, y, z, values.

x	81	91	27	96	95	14	79	3	67	39	05
y	90	63	54	15	48	42	95	84	45	65	80
z	12	9	95	97	80	91	65	93	74	17	63

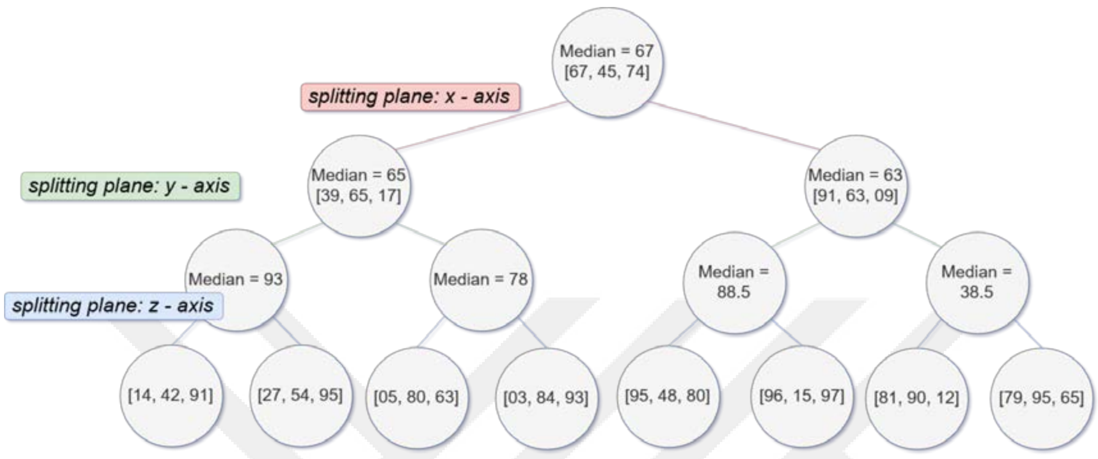

Figure 2.2: The resulting KD-Tree from the randomly generated x, y, and z values.

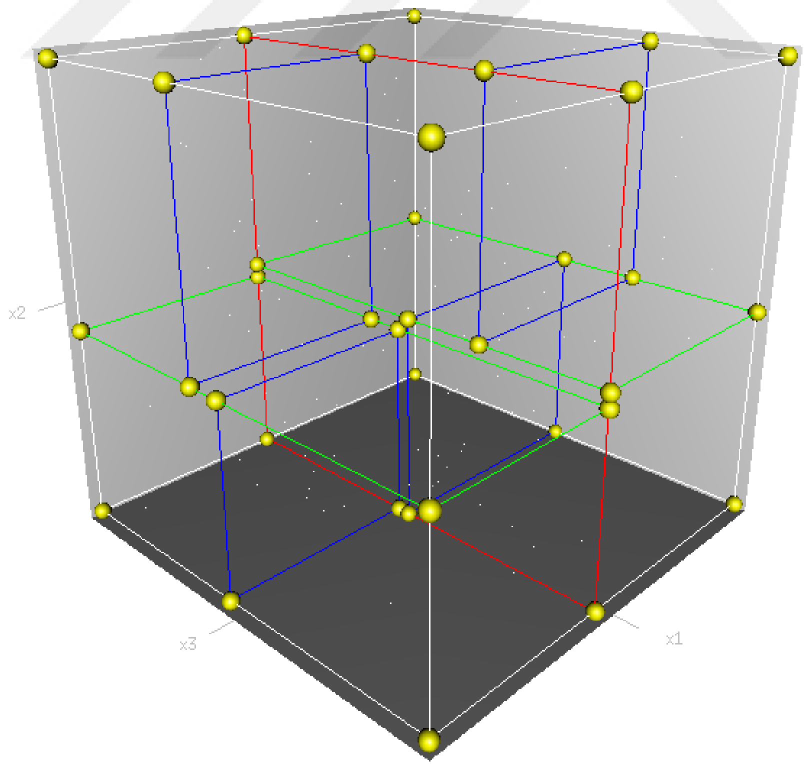

Figure 2.3: An example of space division with hyper planes in a KD-Tree (Tyner, 2006). The red plane is divides the x axis, the green planes divide the y axis, and the blue planes divide the z axis. The nodes are shown as white dots.

Since points which are closer to each other are placed as nodes pointing to each other in KD-Trees, the computational cost is significantly reduced to between $O[\log N]$ and $O[k * N * \log N]$, where k is the dimension and N is the number of data points.

One limitation for KD-Trees is that as the dimensions increase, the efficiency is reduced as most of the points in the tree will need to be evaluated. As a rule, the equality $N \gg 2^k$ must be satisfied for the search to be efficient (compared to other methods) (Jacob E. Goodman, 2004).

For a sample size in the hundreds and a three dimensional space, this equality is satisfied, making this method an efficient one to sample the environment and store the obstacle locations in this thesis. The Scikit Learn implementation was used for the simulations (F. Pedregosa, 2011).

2.3 Probabilistic Road Map (PRM)

The PRM is a path planning method suitable for bigger dimensions and large areas. It is stored as an undirected graph with nodes being collision free configurations and edges being viable paths between these configurations. Construction of a PRM consists of two stages – the learning stage and the query stage.

In the learning stage, the configuration space C is sampled n times randomly. If any of the samples lie in obstacle regions, they are discarded. Depending on the nature of the obstacles in the environment, some checks and additions may be made to the algorithm to ensure uniform samples – such as in the case of Amato et al. where they placed a point at the center of obstacles and extended rays directed outwards uniformly to find sample points on the surfaces of the obstacles. They suggested this method due to the highly cluttered environment and showed it to be able to find viable paths in narrows corridors (N. M. Amato, 1996). Next, a local planner is used to check for collision free paths between the sampled points. In this step, in order to reduce calculations, the nearest k neighbors are checked providing they are within distance d from the current point. While any distance metric could be used, e.g.: Manhattan distance, because of its relevance to UAV flight and its implementation in the KD-Tree class available in the Scikit-Learn library Euclidean distance is used to find the nearest neighbors in this thesis. As a result, a graph $G = (N, E)$ is formed, where N is a list of the samples and

E contains the edges that can be connected to N. The pseudo code of the learning stage is given in Algorithm 2.3.1.

Algorithm 2.3.1: Learning stage algorithm for Probabilistic Road Map

```
function LearningStage (n, k, max_edge_length, obstacles)
input:   n: number of samples
         k: number of nearest neighbors
         max_edge_length: maximum edge length between two nodes
         obstacles: obstacle points
output:  G = (N, E)
while i < n
    random_xyz = random_values()
    if dist(random_xyz, obstacles) < radius
        N ← random_xyz
        i + +
for i = 0, i < n, i + +
    Neighb ← nearest neighbors within max_edge_length distance of N(i, :)
    for j = 0, j < min(length(Neighb), k) , j + +
        if is_collision(Neighb(j), N(i))  == 0
            E ← [N(i), Neighb(j)]
return G = (N, E)
```

In the query stage, the start and goal positions are added as nodes in the graph and are connected to their nearest neighbors, similar to the adding of edges in the learning stage. If a connection for either of the start or goal position is not possible, it results in an automatic failure to find a path. Actions to prevent this will be discussed in the following paragraphs. Once the start and goal nodes are connected to the road map, it becomes possible to find a collision free path that connects them. Depending on the requirements and processing capabilities, Random Walks, Dijkstra, A*, or other path planning algorithms can be used. Usually heuristic path planning algorithms are used with the distance between two nodes or the distance traversed as the cost function.

An example PRM in a three dimensional environment is given in Figure 2.4. In the figure, the red triangle represents the goal position and the other triangles represent the individual UAV start positions. The blue dots are the random samples taken during the learning stage and the yellow lines show the edges connecting the samples. The black dots show the obstacles. As can be seen, most nodes have multiple edges connecting them to other nodes, whereas a few have no edges at all. This is due to the limit on the maximum distance of a possible neighbor node.

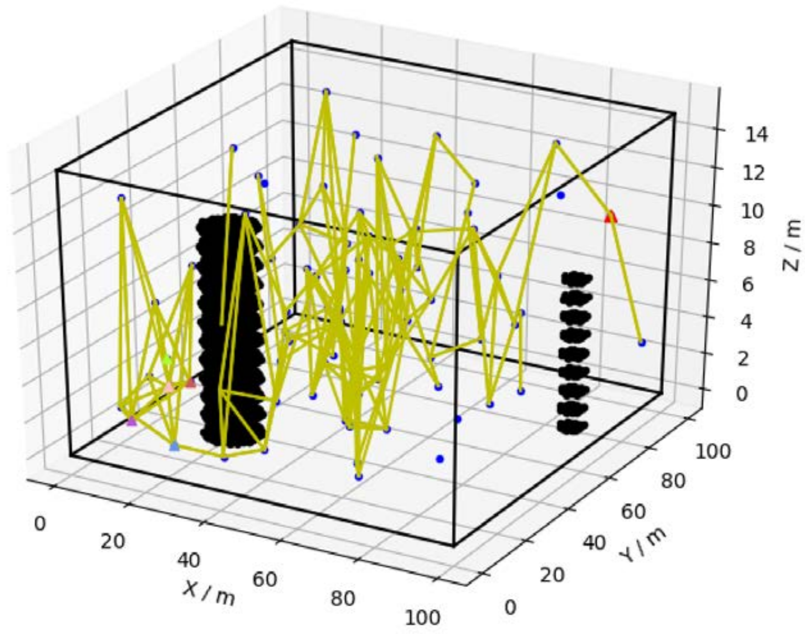

Figure 2.4: PRM in 3D.

Since the sampling of the environment is done randomly in the learning stage, the resulting path may not be the optimal one in terms of distance or efficiency. This can be remedied by running a smoothing algorithm on the path. An example of a smoothening algorithm is one which tries to remove the middle nodes in the path by checking if there is a collision between the previous node and the next node.

If a path cannot be found, it does not necessarily mean that there is no path. The PRM path planner is not optimal but it is proven to be probabilistically complete (David Hsu, 2006) – which means that given enough time or samples, a path will be found if there is one. Therefore, it is important to evaluate the environment and adjust the factors affecting the resulting PRM. These factors are number of samples and the maximum edge length between two nodes.

Increasing the number of samples will increase the chances of finding a more optimal path in terms of smoothness and minimum distance traveled, however it will also increase the computation time spent on the learning stage.

Similarly, increasing the maximum edge length also increases the smoothness of the path and the chances of finding neighbors when the number of samples is low, but it will also increase the number of iterations needed to check for obstacles along the path. While there are some algorithms to shorten edge and obstacle collision checking or to intuitively choose samples closer to obstacles to ensure finding paths around them, eventually, a tradeoff must be made when deciding these parameters.

One main advantage of PRM over other methods is that while the learning stage takes a comparatively longer time than the query stage, in most cases it only needs to be done once, after which many different queries can be made.

2.4 Dijkstra Search Algorithm

Dijkstra’s shortest path algorithm finds the shortest path between nodes in a graph. The original algorithm that was published by Dijkstra found a path by finding the shortest path between two nodes (Dijkstra, 1959). The pseudo code for Dijkstra’s shortest path planner is given in Algorithm 2.4.1

Figure 2.5 shows Dijkstra’s Shortest Path algorithm applied on a three dimensional Probabilistic Roadmap. There are five startıng points in this application show by the different colored triangle markers. While the initial pieces of the found path show differences, it can be seen that the paths eventually converge and the pink edges show the path that is found.

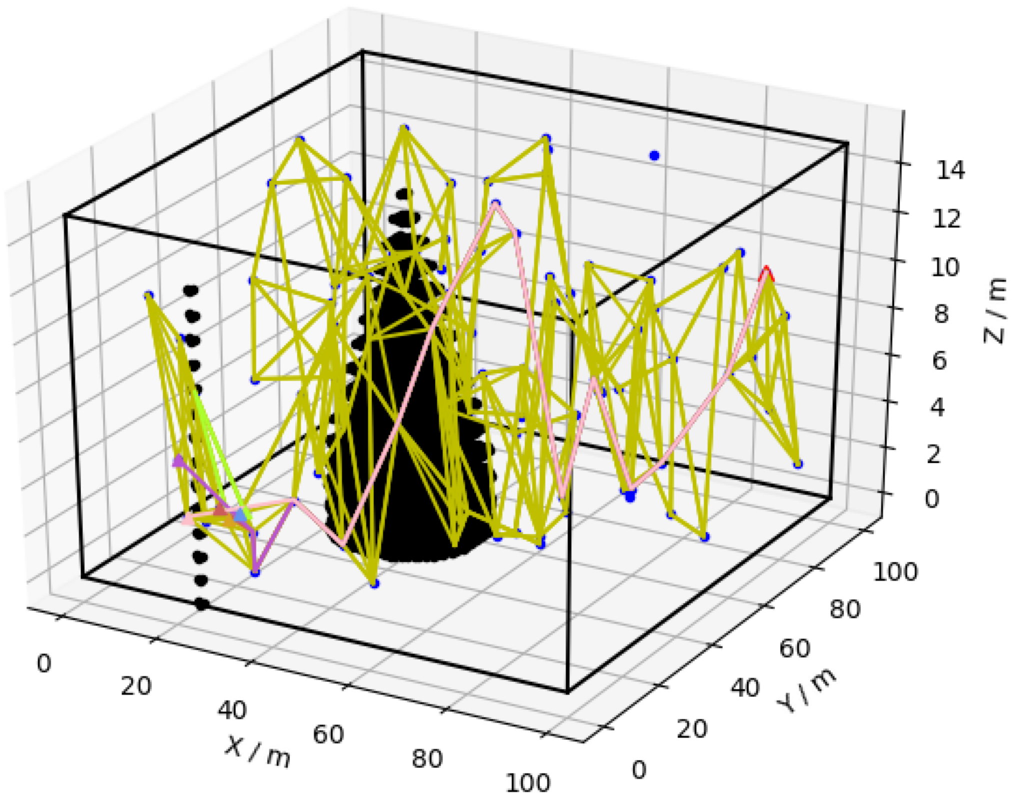

Figure 2.5: Dijkstra’s Shortest Path algorithm applied on a 3D PRM.

Initially, all of the nodes are copied to a temporary set called $openset$. This is the set containing all unexplored nodes. The distance to a node from the start node is stored in the $distance$ array.

Algorithm 2.4.1: Dijkstra's Shortest Path Algorithm

```
function Dijkstra (G, l, start, goal)
input:    G: (V, E)
          l : edge lengths
          start: start node
          goal : goal node
output:   P: (V)
openset ← V
for v in V:
    distance[v] ← ∞
    previous[v] ← ∅
    openset ← v
distance[start] ← 0
P ← ∅
while openset != empty:
    current ← vertex in openset with minimum distance[current]
    Remove current from openset
    if current == goal:
        if previous[current] != ∅ or current = start:
            while current != ∅:
                P ← current
                current = previous[current]
        break
    else:
        for each neighbor v of current:
            if v ∈ openset:
                temp ← distance[current] + l(current, v)
                if temp < distance[v] :
                    distance[v] = temp
                    previous[v] = current
return P
```

Unexplored nodes are assigned a distance of infinity. The node being evaluated at an iteration is called the $current$ node. The first $current$ node is the start node and its distance value is zero. For the rest of the iterations, the $current$ node is always the closest unexplored node to the start node. At the $current$ node, the distance between the $current$ node and its neighbor, which is $l(current, v)$, and the distance from the start node to the $current$ node, $distance[current]$, is added. If this sum is of the distance between an unexplored node and the start node is less than the assigned value, then it is updated. After all of the neighbors of the $current$ have been explored, it is removed from the $openset$ and the next $current$ node becomes neighboring node with the smallest distance value. The $previous$ pointer points to the previous current node. In this manner, all of the paths in the graph are traversed and the shortest paths to a node are labeled until the $goal$ point is reached. Once the $goal$ point is reached, it is inserted in the beginning of the P stack, which contains the shortest path. The $previous$ pointers of the $current$ node are traversed until the $start$ node is reached.

2.5 Bèzier Curves and B-Splines

Bèzier curves and B-Splines have primarily been used in the animation and design sector because of the fact that they can define curved lines and surfaces with a few points in the geometric space. These points are called "control points". Because of the properties of Bèzier curves, the beginning and end of the curve passes through the first and last control points. The control points in between are adjusted to change the shape of the curve as desired. This section will attempt to detail the workings of Bèzier curves and B-Splines and draw attention to the properties which make them suitable for trajectory generation for UAV swarms.

A Bèzier curve is defined as parametric curve which uses Bernstein polynomials as a basis. The equation for the curve is given in Eq. (2.10):

$$\boldsymbol{C}(u) = \sum_{\boldsymbol{i=0}}^{\boldsymbol{n}} \mathbf{P}_{\boldsymbol{i}} B_{i,n}(u), \qquad 0 \leq u \leq 1 \tag{2.10}$$

where n is the degree of the curve, $\mathbf{b}_{\boldsymbol{i}}$ are the control points, $B_{i,n}(t)$ are the basis functions and u is the instance at which the curve is being evaluated.

The n^{th} degree Bernstein polynomial for a n^{th} degree Bèzier curve is given by the formula in Eq. (2.11):

$$B_{i,n}(u) = \frac{n!}{i!\,(n-i)!} * u^i (1-u)^{n-i} \tag{2.11}$$

Evaluating Eq. (2.10) for a cubic Bèzier curve, we get the following equation:

$$\mathbf{C}(\boldsymbol{u}) = (1-u)^3\mathbf{P_0} + 3u(1-u)^2\mathbf{P_1} + 3u^2(1-u)\mathbf{P_2} + u^3\mathbf{P_3} \tag{2.12}$$

Depending on the values of the control points P_i, a curve similar to the one shown in Figure 2.6. From this figure, some properties of Bèzier curves relevant to the trajectory generation of UAVs are visible:

1. The polygons formed by the control points approximate the curve's shape.
2. The curves are within the convex hulls defined by their control points.
3. As the degree of the Bèzier curve increases, the curve deviates more from the polygon defined by its control points, and therefore from the control points themselves.

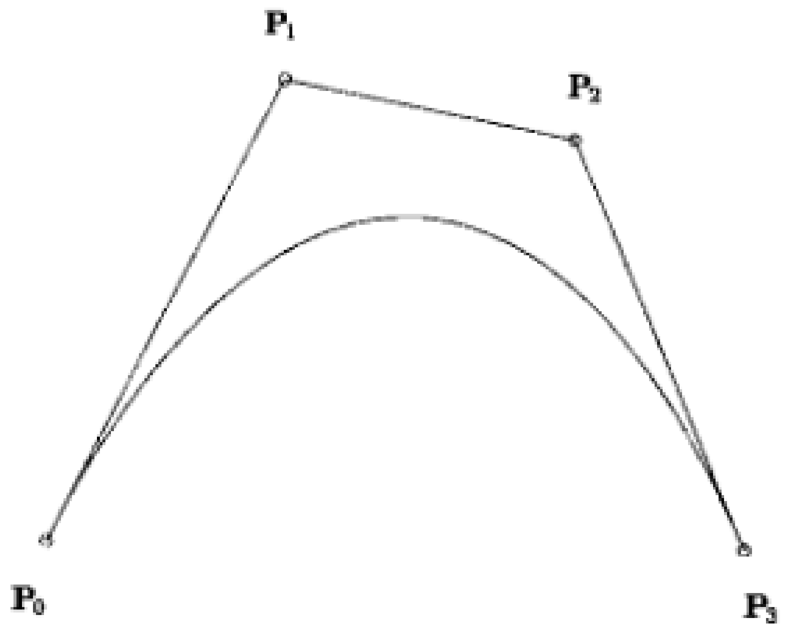

Figure 2.6: An example cubic Bèzier curve (Les Piegl, 1996).

From the viewpoint of trajectory generation from waypoints, some limitations of Bèzier curves become obvious. Firstly, the number of waypoints found during the path planning stage cannot be limited or known beforehand, therefore more than four waypoints are likely to be found. This not only increases the computing time to generate the trajectory, but the resulting trajectory is generally unstable due to round-off errors and other numerical uncertainties during the computations. Furthermore, it will not follow the control points closely, which may lead to collisions, as the distance between the found safe path and the trajectory will increase. Secondly, if the path were to change midway due to changed missions or dynamic obstacles, the new trajectory would not be continuous. To solve these problems, B-Splines are implemented.

B-Splines are piecewise polynomial curves, basically they are a two or more Bèzier curves placed end to end. Figure 2.7 shows an example cubic B-Spline.

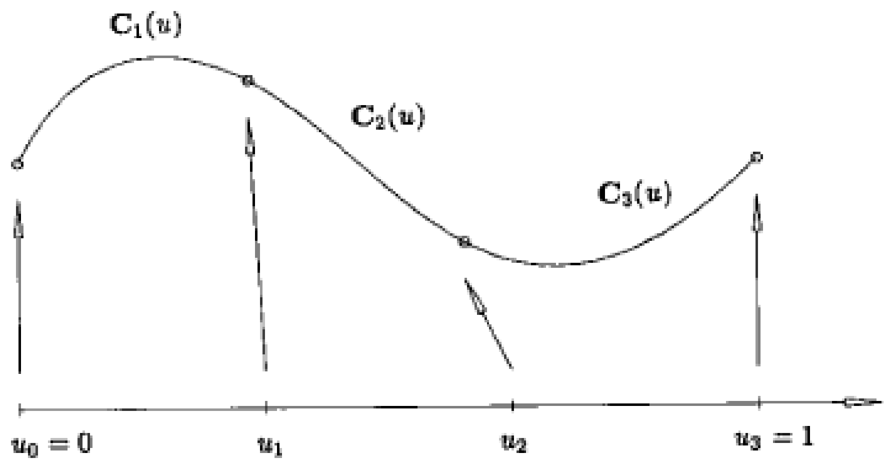

Figure 2.7: An example of a B-Spline curve consisting of three polynomials (Les Piegl, 1996).

A B-Spline is defined by its control points, knot vector and degree. The equation defining a B-Spline curve is given in Eq. (2.13).

$$\boldsymbol{C}(u) = \sum_{\boldsymbol{i=0}}^{\boldsymbol{n}} \mathbf{P}_{\boldsymbol{i}} N_{i,n}(u), \qquad a \leq u \leq b \tag{2.13}$$

Here P_i are the control points, a and b are the minimum and maximum values of the knot vector and $N_{i,n}(u)$, given in Eq. (2.14) and Eq. (2.15), is the n^{th} degree B-Spline basis functions defined on the knot vector.

$$N_{i,0}(u) = \begin{Bmatrix} 1, & if\ u_i \leq u < u_i + 1 \\ 0, & if\ otherwise \end{Bmatrix} \tag{2.14}$$

$$N_{i,n}(u) = \frac{u - u_i}{u_{i+n} - u_i} N_{i,n-1}(u) + \frac{u_{i+n+1} - u}{u_{i+n+1} - u_{i+1}} N_{i+1,n-1}(u) \tag{2.15}$$

The knot vector $U = \{u_0, u_1, \dots \dots, u_m\}$ consists real numbers - $u_0, u_1, \dots \dots, u_m$, which are called the knots, and $u_i \leq u_{i+1} \leq u_{i+2}$ must be satisfied. An important feature of knot vectors is whether they are clamped or not. In B-Splines, the end points of the curve do not necessarily pass through the first and last control points. A clamped knot vector forces the ends of the B-Spline to pass through these points and must contain $p + 1$ equal knots at the beginning and the end, like shown in the knot vector U in (2.16).

Another important relation is the relationship between the number of elements in a knot vector with the degree of the B-Spline and the number of control points. This is given in Eq. (2.17) with p as the number of control points given.

$$U = \{u_0, u_1 = u_0, \ldots, u_n = u_0, u_{n+1}, \ldots u_{m-n-1}, u_{m-n} = u_m, u_{m-n+1} = u_m, .., u_m\} \tag{2.16}$$

$$m = n + p + 1 \tag{2.17}$$

The knots in the knot vector do not affect the shape of the B-Spline in any other way, however, they do affect the derivatives. A smaller value of $b - a$ leads to a "faster trajectory", as the interpolation steps decrease and the positions found are further apart in order to pass through the first control point at $u = u_0$ and the last control point at $u = u_m$.

There are some important priorities of B-Splines which make them especially useful for generating the trajectories of Quadrotor Swarms. They are discussed in the following paragraphs.

The fact that the entire trajectory of a UAV can be stored as a few control points and basis functions reduces the amount of operations regarding storing and deleting trajectories. Also, as lags in data transmission and ensuring the integrity of a transmitted message between UAVs in a swarm is a genuine problem, reducing the frequency and size of these messages is necessary. Instead of constantly broadcasting to and listening for nearby UAV positions or sending the entire trajectory in (x, y, z, t) coordinates, it is simpler and more secure to just send a few control points. The trajectories of respective UAVs can then be calculated on-board.

As mentioned before, B-Splines are continuous, therefore not just ensuring a smooth trajectory, but also allowing velocity controllers to be used – thus taking advantage of the on-board accelerometers of the UAV. For position controllers, either GPS needs to be used or the output of the accelerometers have to be integrated twice, which is not reasonable. And since this implementation of UAV swarm is for indoor or underground environments, using GPS is not an option.

Another feature of B-Splines – the strong convex hull property – states that the curve is contained in the convex hull of its control polygon. This feature allows the aligning of control points such that the related segment of the B-Spline is a straight line. In this case, three control points are collinear and the B-Spline is forced to pass through the center control point. This is important because it means that if pseudo control points are placed in the appropriate locations, the B-Spline will be forced to pass through the original control points, thus ensuring minimal deviation from the path found by the planner.

The ability to change the derivative of the B-Spline by changing the knot vector minimum and maximum values basically means that the velocities of the UAVs can be changed by simply changing these values. This makes it easier to prevent collisions between the UAV and other moving objects, such as other UAVs in the swarm or dynamic obstacles, as there is no need to find new paths or generate different trajectories. This is the principle which is used to prevent collisions between UAVs, while allowing them to travel the same trajectory.

3. IMPLEMENTATION

In this thesis, a decentralized trajectory generation algorithm for multi UAV systems flying in a cluttered environment and through narrow corridors is designed. For each UAV, the obstacles in the environment are broken down and modelled as points in the space and stored in a KD-Tree for fast querying. The free space in the environment is sampled and a PRM is generated. Afterwards, the shortest path to the goal point is found using Dijkstra's Shortest Path algorithm. Path shortening is applied to remove redundant nodes and smoothen the path. Pseudo control points are calculated for each point on the UAV's path. A cubic B-Spline is fitted on the path using the original path waypoints and the pseudo control points. The pseudo control points ensure that the Spline passes through all of the control points and is continuous in its first and second derivatives. Since the goal point is the same for all of the UAVs, their paths are expected to converge at some point. The B-Spline control points are shared with nearby UAVs, therefore allowing them to calculate the trajectories of other UAVs in the swarm. This allows them to predict the point of collision, if there is one. A priority ID is assigned to the UAVs by each other based on their closeness to the point that the trajectories intersect. Then, the UAVs intuitively adjust their own knot vector to pass the intersection point after those that have higher priority ID. Similarly, if a dynamic obstacle is detected throughout the flight, the UAVs calculate the predicted trajectory of the obstacle and use time scaling to avoid it. If new static obstacles are detected, the PRM is updated and a new path is found from the nearest unobstructed waypoint to the goal position. The new path is then joined to the UAV's spline trajectory at an appropriate node such that the first and second derivatives of the trajectory are continuous.

The algorithm scheme is given in Figure 3.1.

The remainder of this section will discuss the implementation of the methods mentioned above and will detail the algorithms for path shortening, pseudo-control point generation, time-scaling of trajectories and joining spline trajectories.

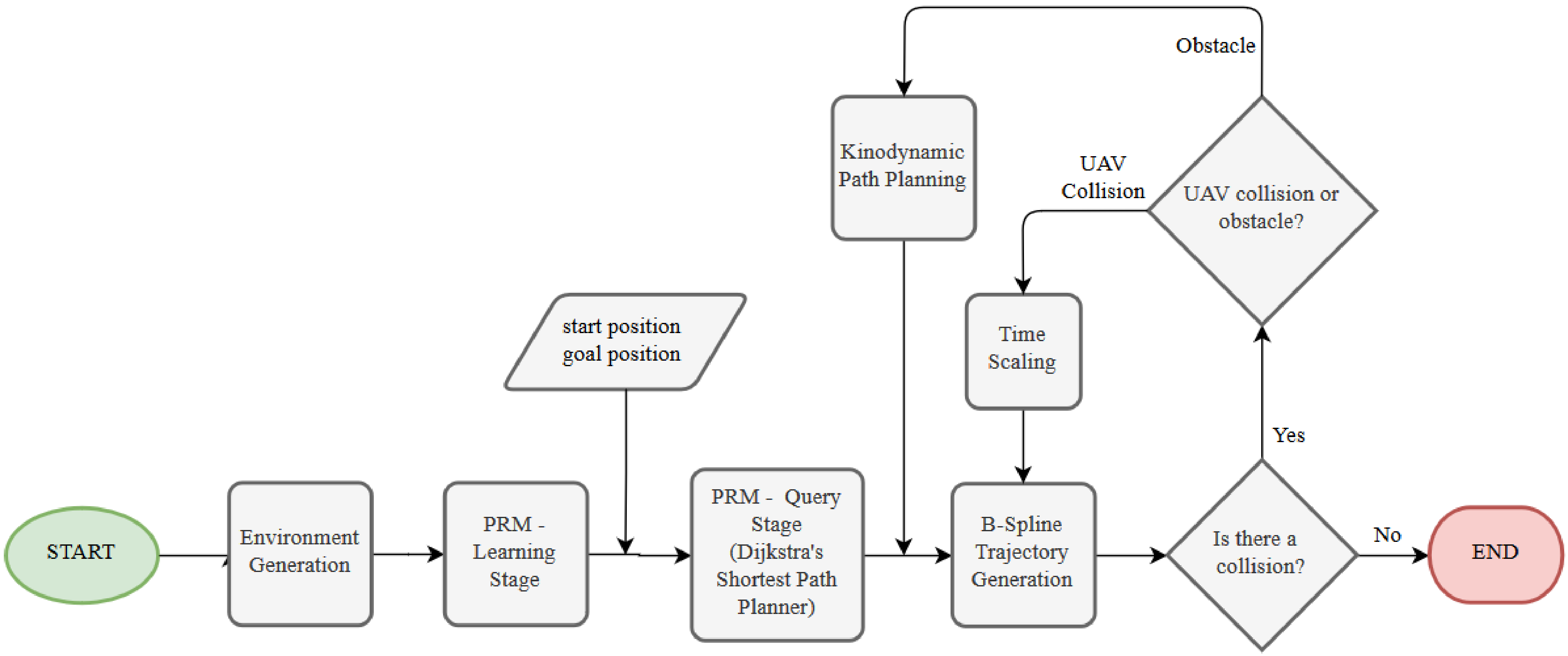

Figure 3.1: The general algorithm design for the multi-UAV system.

3.1 Path Shortening Algorithm

The main principle behind the path shortening algorithm is to sequentially traverse through the waypoints in the path and remove next waypoint if there is no obstacle between the waypoint after it and the current waypoint until the last waypoint is reached.

This algorithm is run after the query stage of the PRM. Since this process is not very efficient in terms of computational operations (due to step-wise checking of long path segments for obstacles), a trade-off between path smoothness and time taken to find a path was made. As a result, this algorithm is only run if there are more than five linear path segments in the found path. Furthermore, dynamic obstacles are not taken into account while running the path shortening algorithm.

The algorithm is given in Algorithm 3.1.1.

Algorithm 3.1.1: Path Shortening Algorithm

---

function ShortenPath ($waypoints, obstacles$)
input: $waypoints$: the coordinates of the points in the path
$obstacles$: obstacle points
output: $waypoints$
while $i < length(waypoints) - 2$
    for $j = i + 2,\ j < length(waypoints), j++$
        if $no\ collision(waypoints, obstacles)$
            $delete\ waypoint(j-1)$
return $waypoints$

3.2 Pseudo-Control Point Generation

The pseudo-control points are generated such that the resulting B-Spline trajectory passes through all of the original waypoints. This is done by relying on a property of B-Splines which says that if three control points are collinear then the spline will pass through the middle point. Another property is that if the points are placed equidistant, then the first derivative will be continuous.

For each $(n, n+1)$ waypoint pair, two pseudo control points are generated. Depending on the placement of the waypoints within the path, the pseudo-control points are calculated in four different ways.

3.2.1 For the first waypoint

In order to ensure first derivative continuity at the second waypoint, the distance between the pseudo-control points is taken as the third of the smaller of the lengths of the first and second linear segments, formed by the first and second waypoints and the second and third waypoints respectively. The second linear segment is extended towards the first waypoint and a pseudo-control point is found by iteratively moving along the extension until the distance requirement is met.

The second pseudo-control point is found by iteratively moving from the first waypoint to the first pseudo-control point until the distance requirement is, once again, met. The whole process is illustrated in Figure 3.2. The green points labeled p_1 and p_2 are the calculated pseudo-control points.

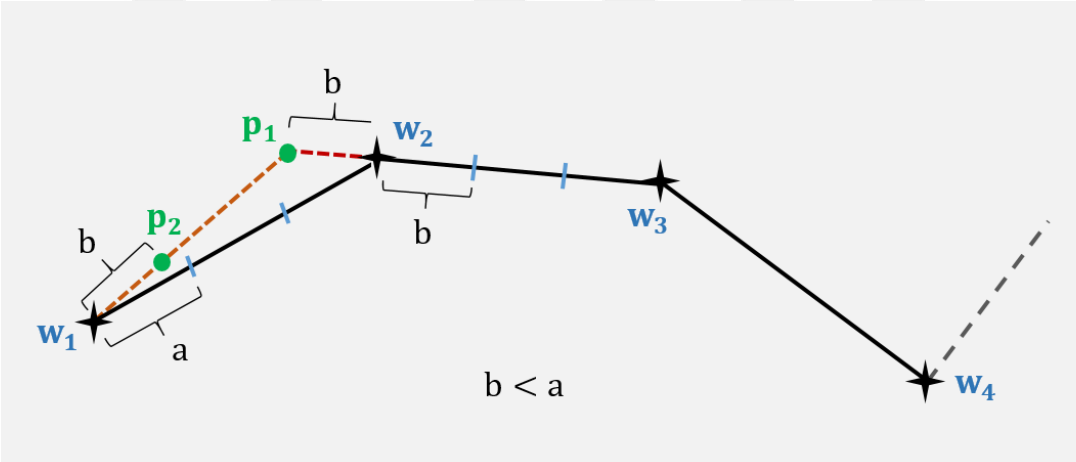

Figure 3.2: Calculation of pseudo-control points for the first linear segment in the path.

3.2.2 For even waypoints

Pseudo-control points for linear segments beginning with an evenly numbered waypoint are placed on top of the linear segment. For the first pseudo-control point, the lengths of the linear segments before and after the waypoint are considered. The pseudo-control point is placed at one third the minimum length from the evenly numbered waypoint.

For the second pseudo-control point, the lengths of the linear segment of the evenly numbered waypoint and the segment after it is considered. Once again, the pseudo-control point is placed at one third the minimum length from the oddly numbered waypoint (following the evenly numbered waypoint). The process is shown in Figure 3.3.

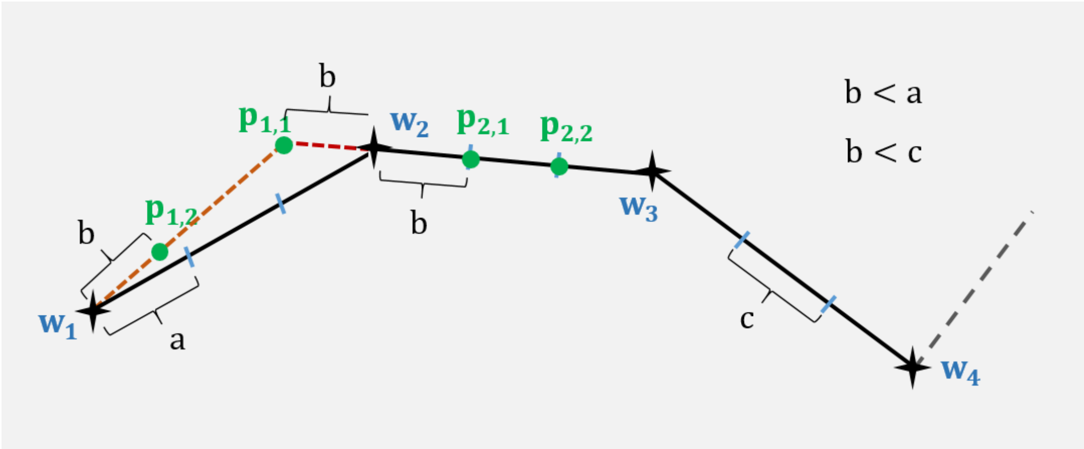

Figure 3.3: Calculation of pseudo-control points for evenly numbered waypoints.

3.2.3 For odd waypoints

For oddly numbered waypoints the pseudo-control points are placed as follows. The linear segment before the odd waypoint is extended towards the waypoint and a pseudo-control point is placed at an appropriate distance from it. The linear segment of the following even waypoint is also extended in the direction of the waypoint and the second pseudo-control point is placed at an appropriate distance from the even waypoint. The process is illustrated in Figure 3.4.

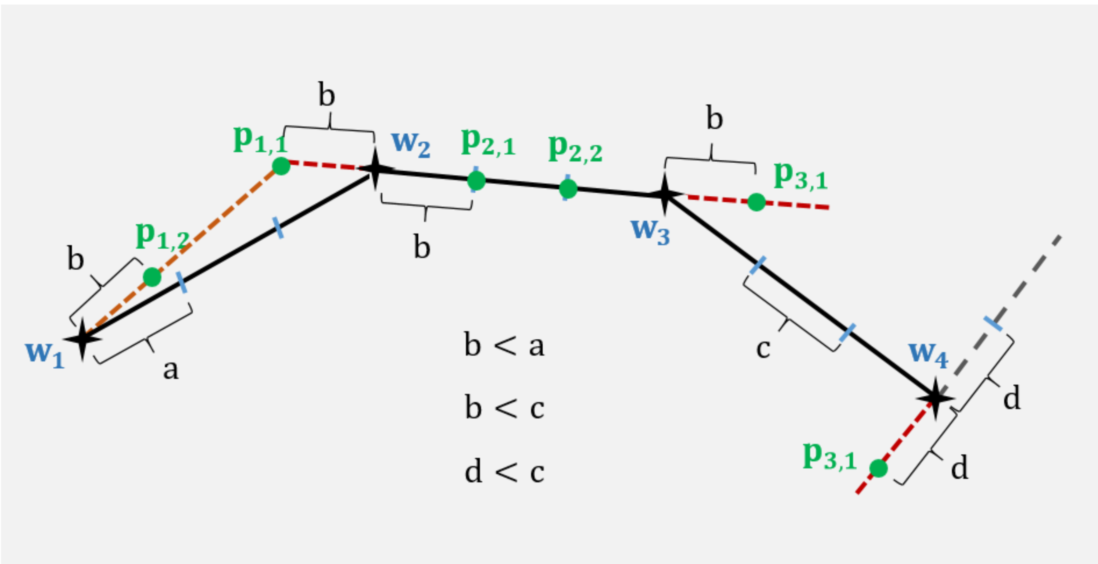

Figure 3.4: Calculation of pseudo-control points for oddly numbered waypoints.

3.2.4 For the second last waypoint

If the second last waypoint is evenly numbered, then the pseudo-control points are calculated as for the evenly numbered waypoints, except for while calculating the second pseudo-control point, instead of checking the length of the next line segment, the pseudo-control point is placed at one third the length of the current line segment from the last waypoint.

If the second last waypoint is oddly numbered, then the reverse of the process for the first waypoint is performed. Once again, the minimum length of the current and previous linear segment is found. The previous linear segment is extended in the direction of the last waypoint and a pseudo-control point is placed at an appropriate distance from the second last waypoint. The second pseudo-control point is placed at an appropriate distance from the last waypoint and towards the first pseudo-control point. The process is illustrated in Figure 3.5.

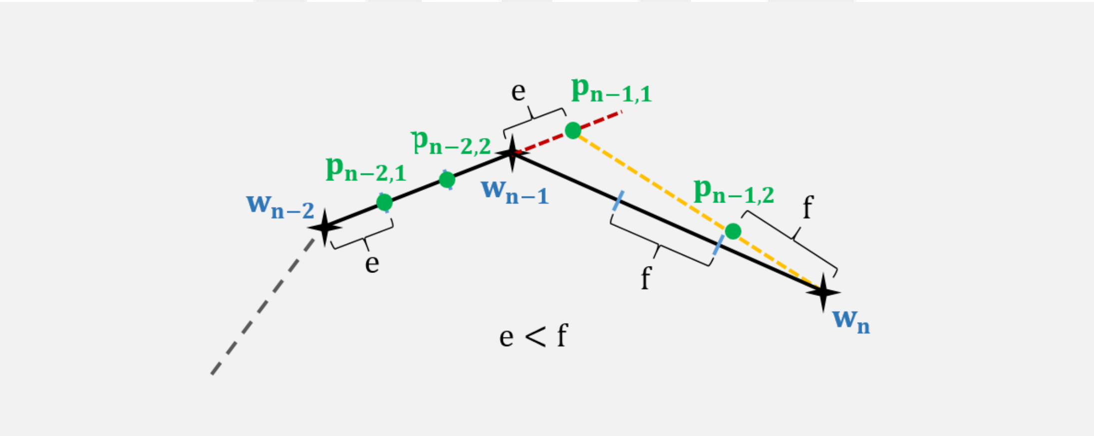

Figure 3.5: Calculation of pseudo-control points for the second last waypoint.

3.3 Time-Scaling of Trajectories

Trajectory time-scaling is possible because of the knot vector property in B-Spline curves. As mentioned in Section 2, the knot vector is a vector with non-decreasing values on which the basis functions of the B-Splines are defined. A clamped, uniform knot vector produces basis functions like the ones seen in Figure 3.6. The knot vector which produced this graph is given below. It is for a spline with 10 control points.

$$U_1 = [0 \quad 0 \quad 0 \quad 0 \quad 0.1428 \quad 0.2857 \quad 0.4285 \quad \cdots$$
$$\cdots \quad 0.5671 \quad 0.7142 \quad 0.8571 \quad 1 \quad 1 \quad 1 \quad 1]$$

Each curve on the graph is for an individual basis function to be used in the spline calculations. The knot vector is plotted on the x-axis and the values of the basis functions as the instant u is given on the y-axis.

In time-scaling, essentially the range of the knot vector is either increased or decreased by changing the last knot in the knot vector. This has the effect of "slowing down" the trajectory. While the overall trajectory does not change, the derivatives, velocity and acceleration, do. The reason for this is obvious in the B-Spline curve calculation equation given in Section 2 and in the basis function graphs given in Figure 3.6 and Figure 3.7.

In Figure 3.7, the basis functions for another clamped knot vector is shown. This knot vector is the same as U_1 except that its last knot has been increased and is given as U_2. The effect of increasing the last four knots can be seen as the last four curves are horizontally expanded, while the previous curves are compressed.

The effect of time-scaling is better seen in an example. For this, a B-Spline trajectory is generated for 10 control points using the knot vectors U_1 and U_2. The resulting trajectory is given in Figure 3.8.

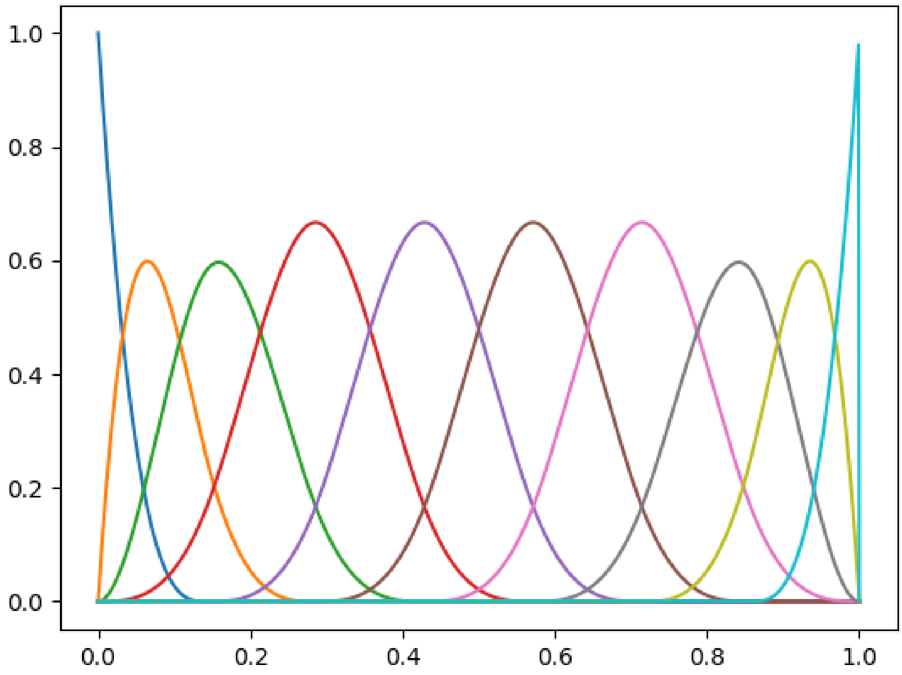

Figure 3.6: Basis functions for clamped, uniform knot vector U_1. The different colored plots are the different basis functions. The x-axis gives the range of the knot vector. The y-axis shows the values of the basis functions.

$$U_2 = [0 \quad 0 \quad 0 \quad 0 \quad 0.1428 \quad 0.2857 \quad 0.4285 \quad \cdots$$
$$\cdots \quad 0.5671 \quad 0.7142 \quad 0.8571 \quad 2 \quad 2 \quad 2 \quad 2]$$

The control points used for the trajectory are from a helix and are given in matrix C.

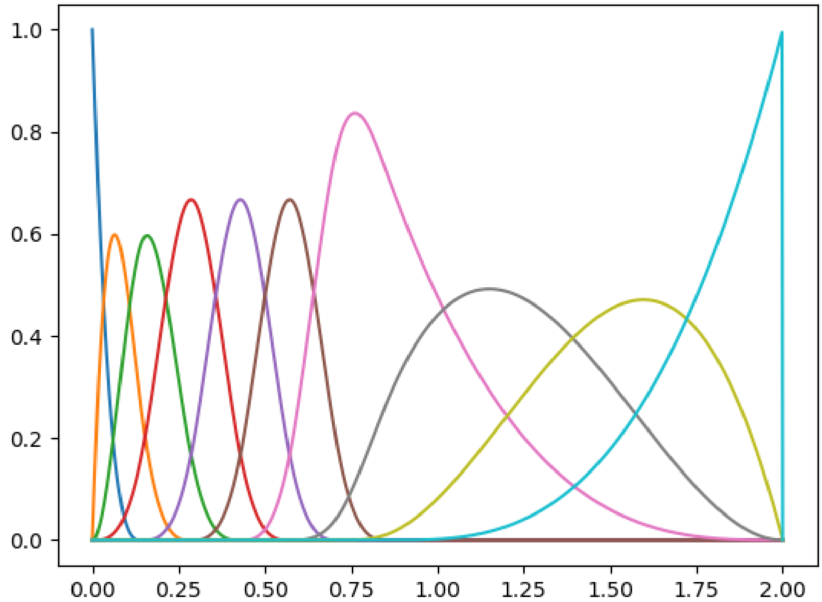

Figure 3.7: Basis functions for clamped, non-uniform knot vector U_2. The different colored plots are the different basis functions. The x-axis gives the range of the knot vector. The y-axis shows the values of the basis functions.

$$C = \begin{bmatrix} 0 & 0.64 & 0.98 & 0.86 & 0.34 & -0.34 & -0.86 & -0.98 & -0.64 & -0.24 \\ 1 & 0.76 & 0.17 & -0.5 & -0.93 & -0.93 & -0.5 & 0.17 & 0.76 & 1 \\ 0 & 0.69 & 1.39 & 2.09 & 2.79 & 3.49 & 4.18 & 4.88 & 5.58 & 6.28 \end{bmatrix}$$

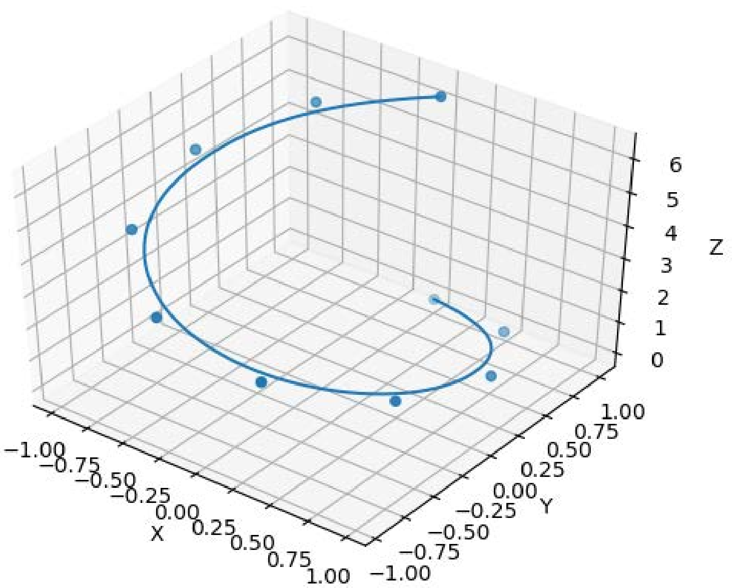

Figure 3.8: The B-Spline trajectory for the control points C.

The resulting first and second derivative graphs for knot vector U_1 is given in Figure 3.9 and Figure 3.10. The time or instance of the spline is given on the x-axis.

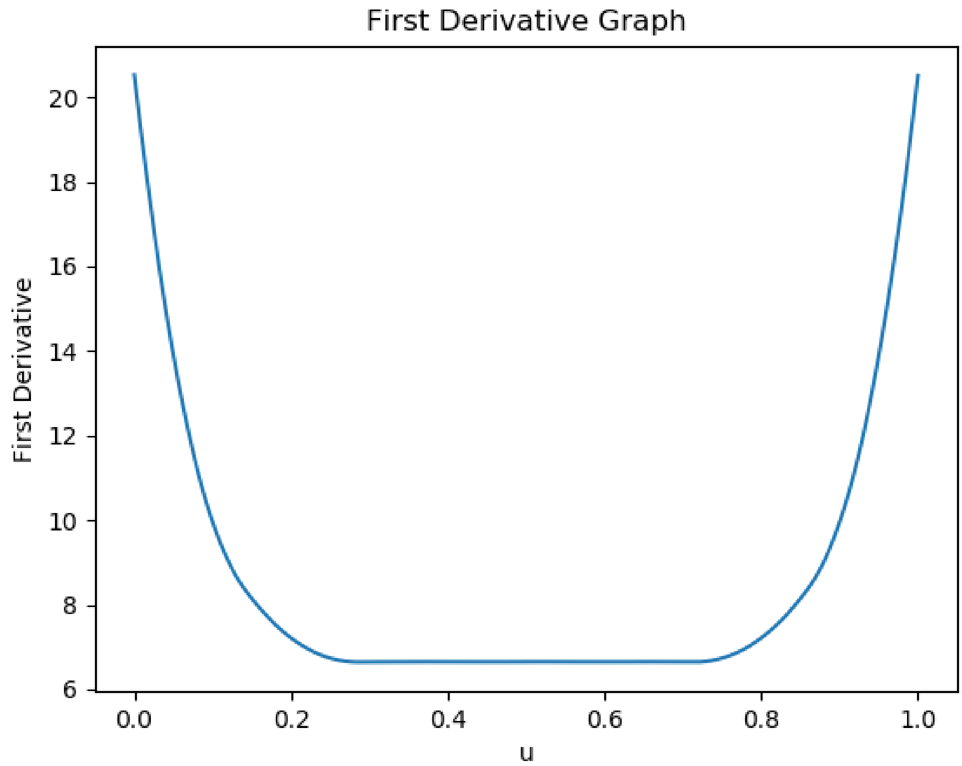

Figure 3.9: First derivative graph for knot vector U_1.

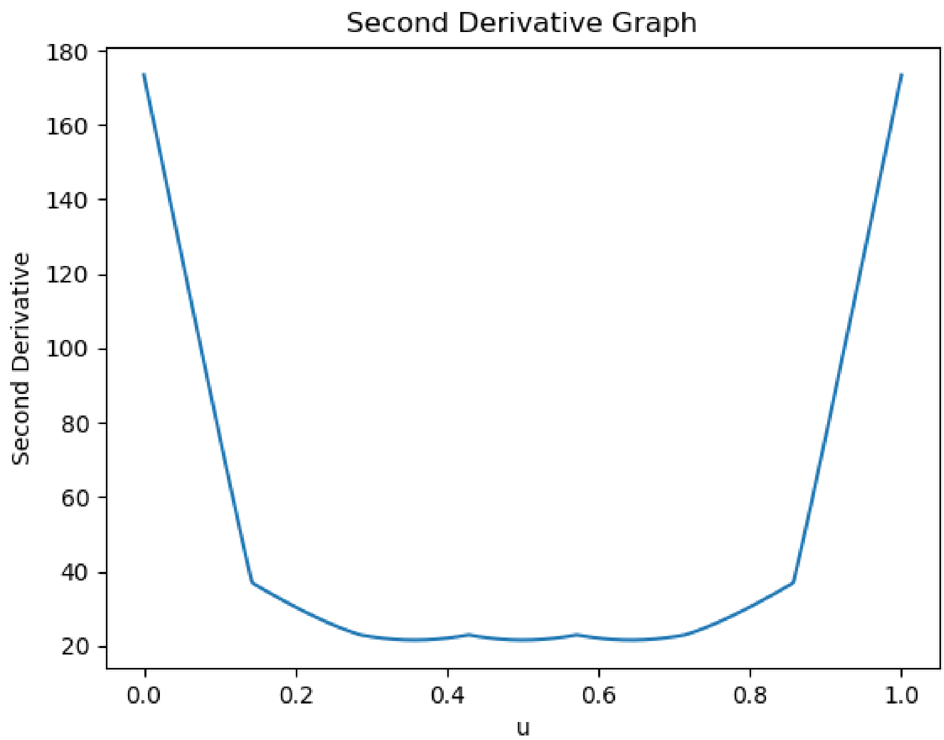

Figure 3.10: Second derivative graph for knot vector U_1.

The resulting first and second derivative graphs for knot vector U_2 is given in Figure 3.11 and Figure 3.12.

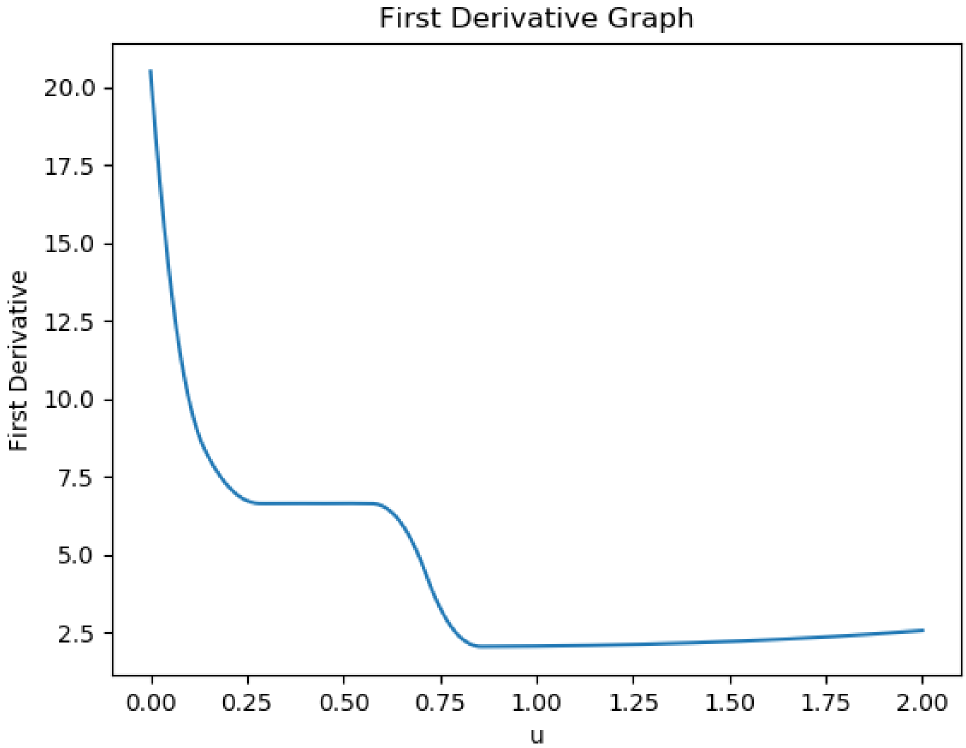

Figure 3.11: First derivative graph for knot vector U_2.

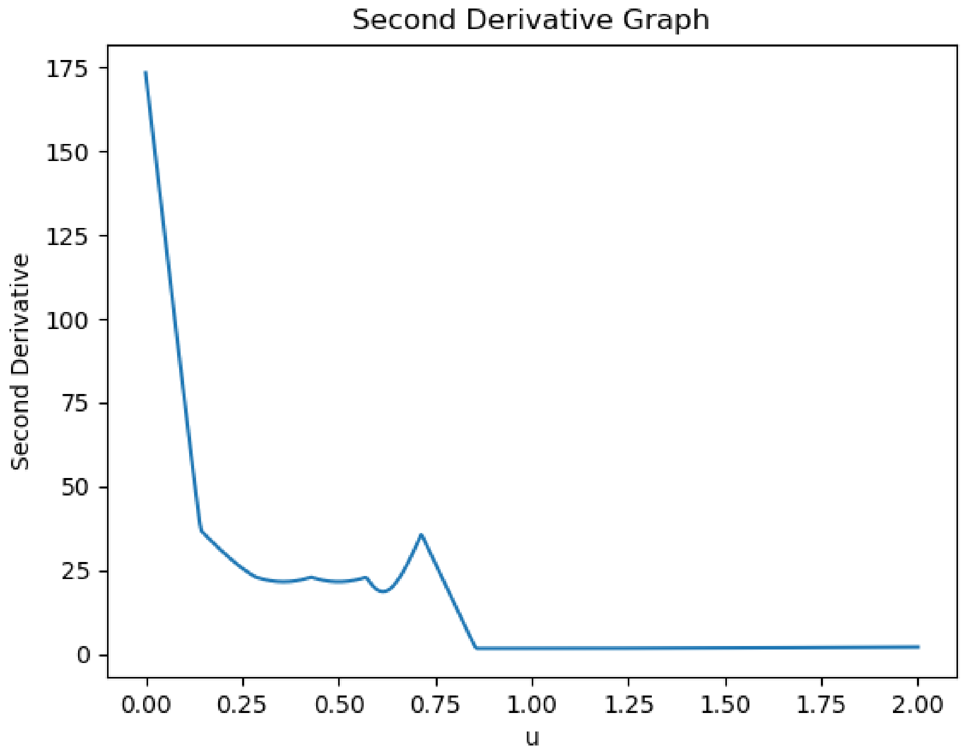

Figure 3.12: Second derivative graph for knot vector U_2.

3.4 New Static Obstacle Detection

If a new static obstacle is detected, firstly the control points used to generate the spline trajectory are checked if they lie within the obstacle. If they do, the nearest before and after way point of the path is noted. The PRM nodes which lie in the obstacle are removed from the map and anew road map is generated. Then a collision free path is queried using the previously noted way points as start and goal nodes. The found path is added in the appropriate place in the original path and the pseudo-control points are generated as detailed above.

3.5 Joining B-Splines

Joining of B-Splines is necessary when a new path needs to be calculated due to the detection of new obstacles. Either the entire trajectory must be calculated from the start or the three trajectories – the beginning, the new trajectory, and the end - must be joined. Both of the methods were implemented, however, recalculating the trajectory from the start gives better results in terms of continuity.

The reasons for this is discussed in detail in the piecewise generation of trajectories section and for the remainder of the thesis, trajectories are recalculated as a whole spline.

3.5.1 Regenerating the entire trajectory

In this method, a single b-spline is generated using all of the control points (the way points and the pseudo-control points).

In Figure 3.13, example trajectories of two adjacent UAVs heading to the same goal is shown. The markers on the graph show the three way points and pseudo-control points calculated for the generation of the spine. Figure 3.14 and Figure 3.15 shows the velocity of the generated trajectory. As can be seen both of the velocities are continuous.

Trajectories containing more waypoints are given in Figure 3.16. Their resulting continuous velocities are given in Figure 3.17 and Figure 3.18.

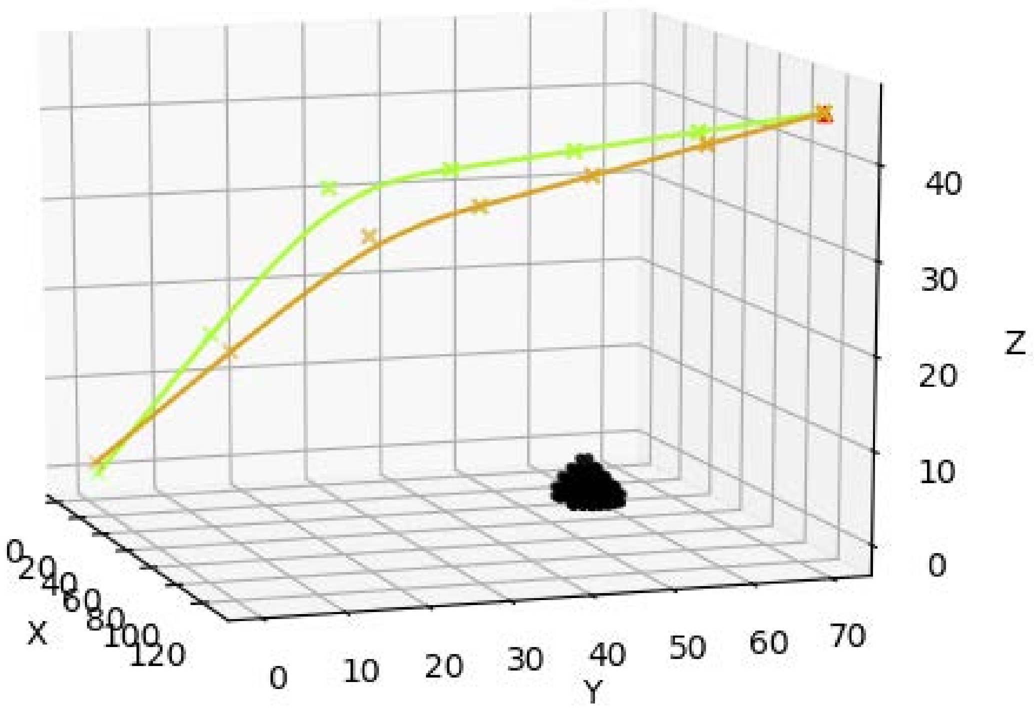

Figure 3.13: Example trajectories for two adjacent UAVs.

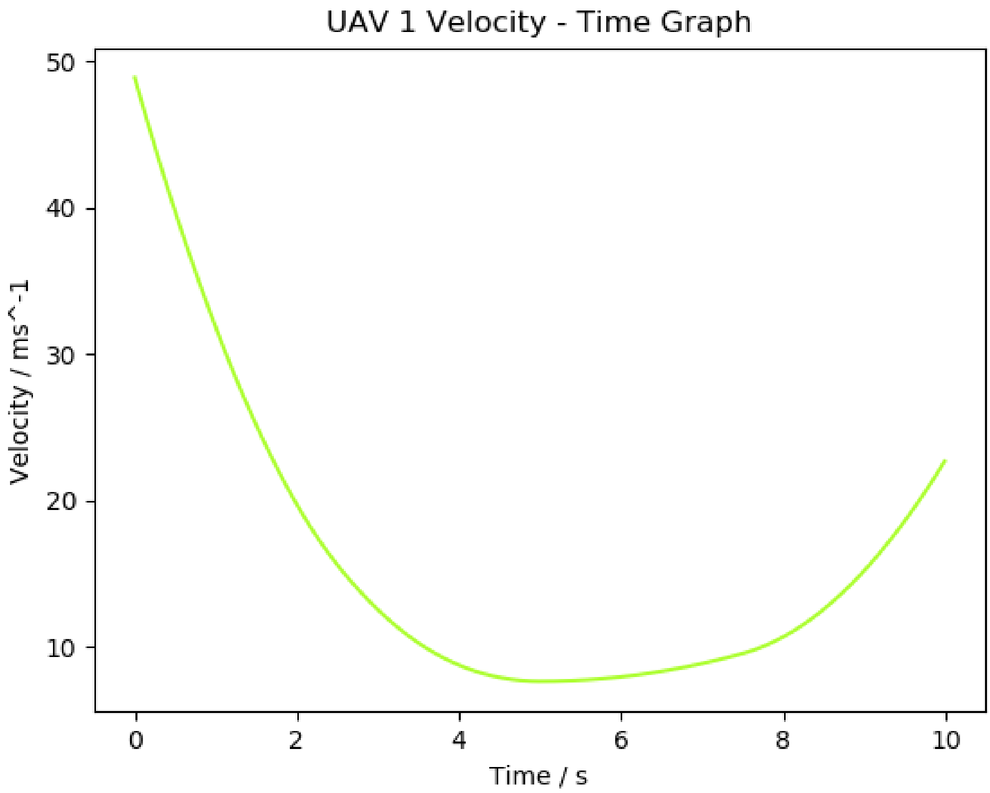

Figure 3.14: Velocity for UAV 1.

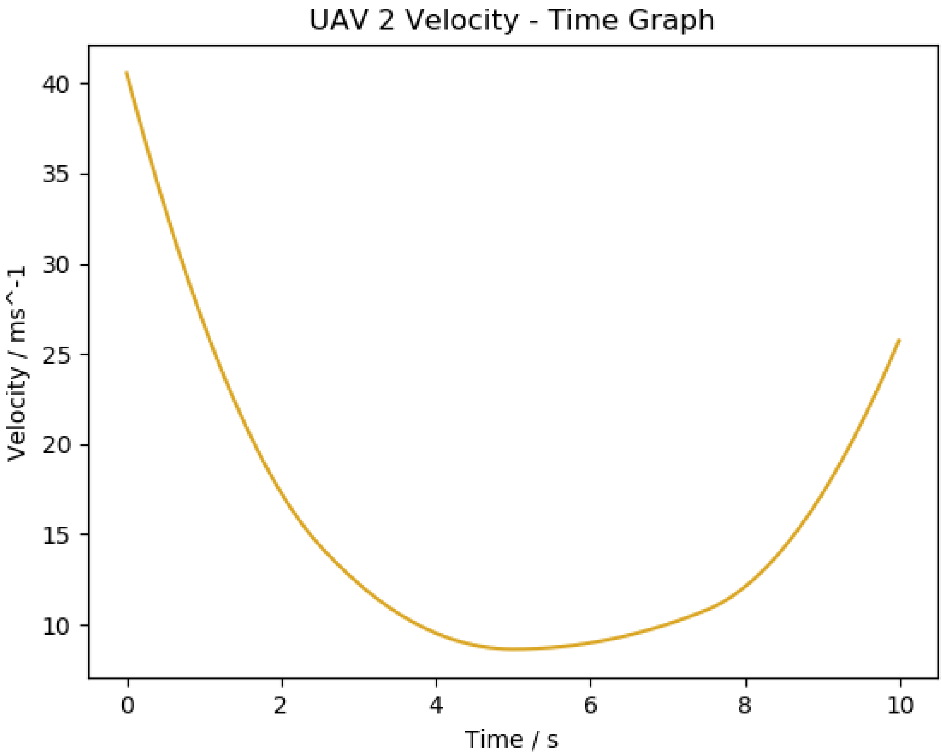

Figure 3.15: Velocity for UAV 2.

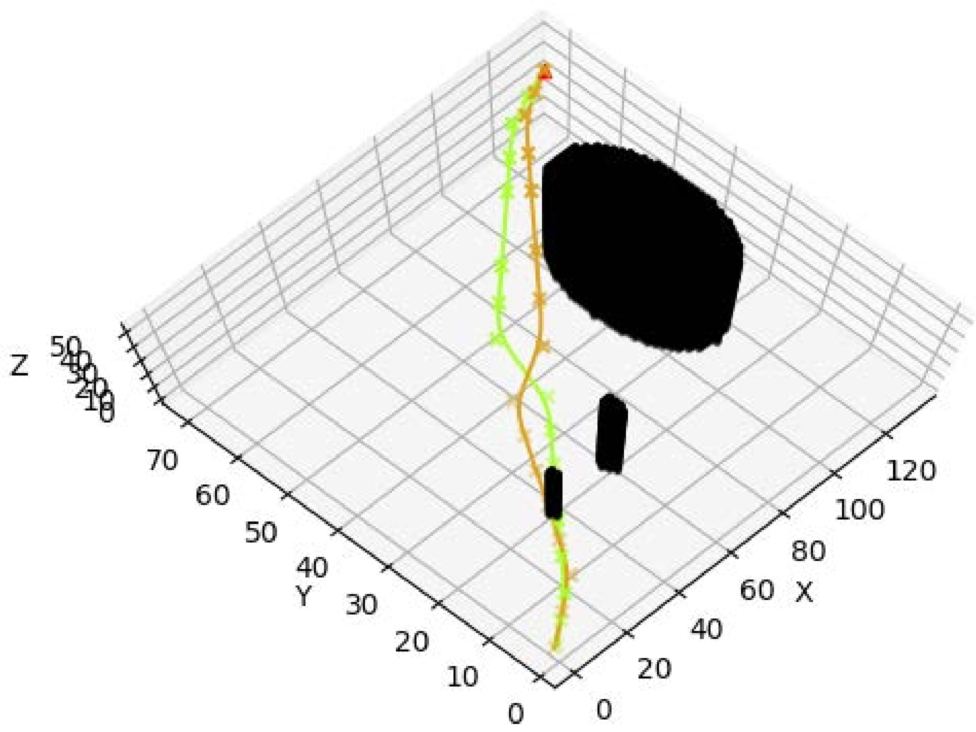

Figure 3.16: Example trajectories for two adjacent UAVs with more than three waypoints.

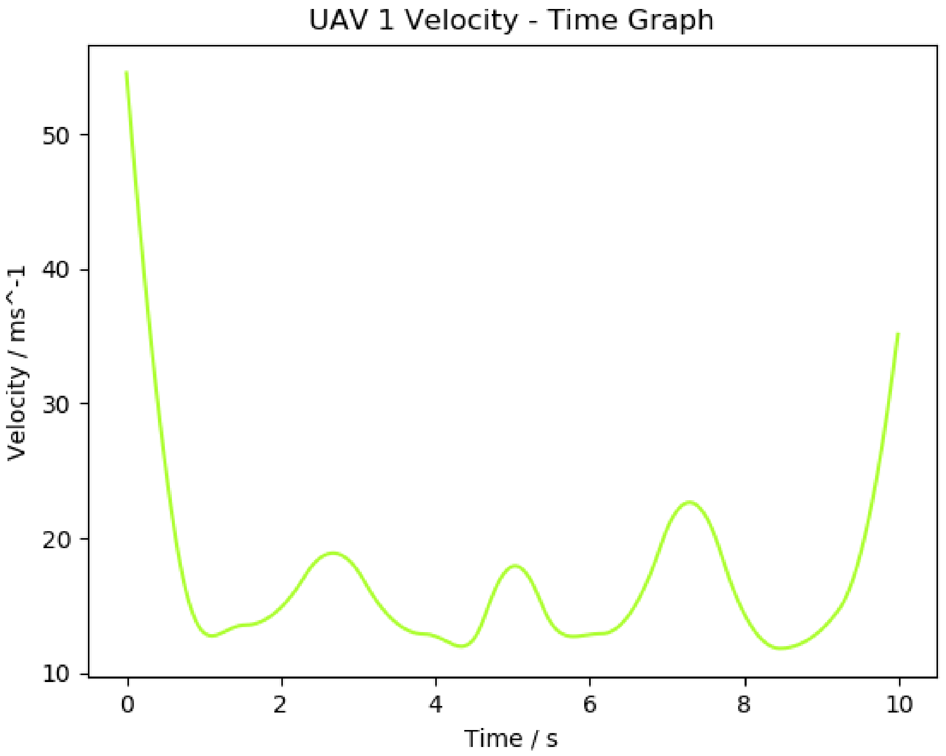

Figure 3.17: Velocity for UAV 1.

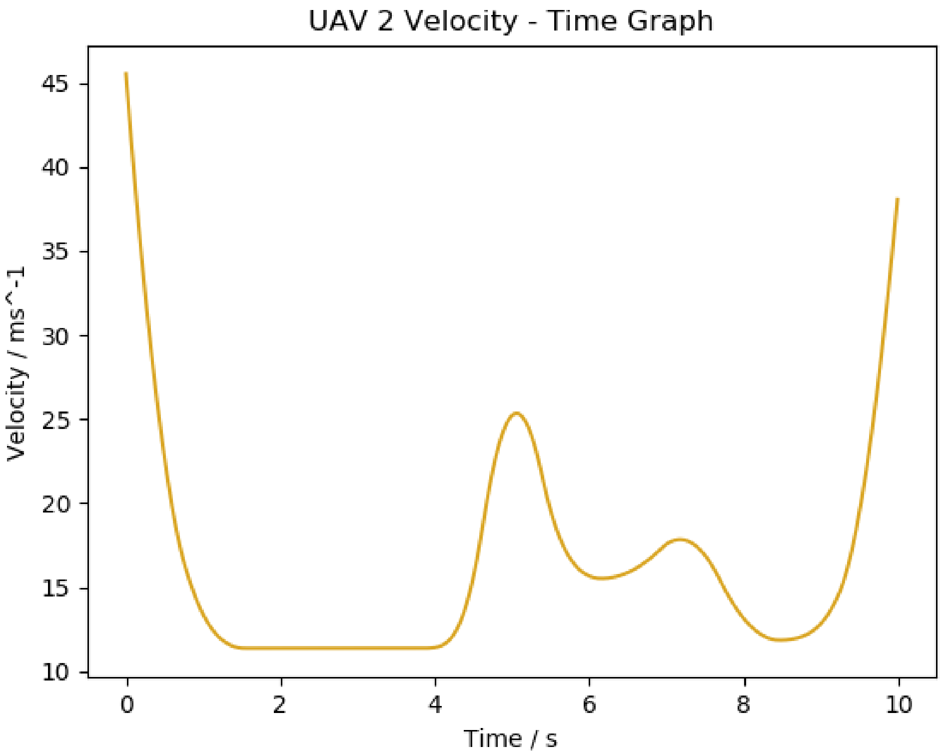

Figure 3.18: Velocity for UAV 2.

3.5.2 Piecewise generation of the trajectory

In this method, for each pair of consecutive waypoints and the pseudo-control points calculated between them a b-spline curve is generated. While this method is more efficient for joining trajectories, the pseudo-control points must be finely optimized to get equal distances before and after the way point.

The knots in the knot vectors must also be optimized to ensure the same initial velocity as the previous splines' ending velocity. Because these are optimization processes, there is a trade-off between the continuity of the velocities and the computational burden of choosing smaller increments. Furthermore, due to the nature of the algorithm of alternating curve and line trajectories (stemming from the collinear property of b-splines), the velocities are constant in the linear segments of the spline.

This problem is evident for the trajectory in Figure 3.19. Here, each consecutive pair of orange points on the blue trajectory is the starting and ending point of one B-Spline curve. While the trajectory is smooth, it can be seen that the velocity shown in Figure 3.20 is not continuous. The black vertical lines represent the duration of each B-Spline curve segment. The problem is clearly visible at the second, fifth, sixth, and eighth black vertical lines.

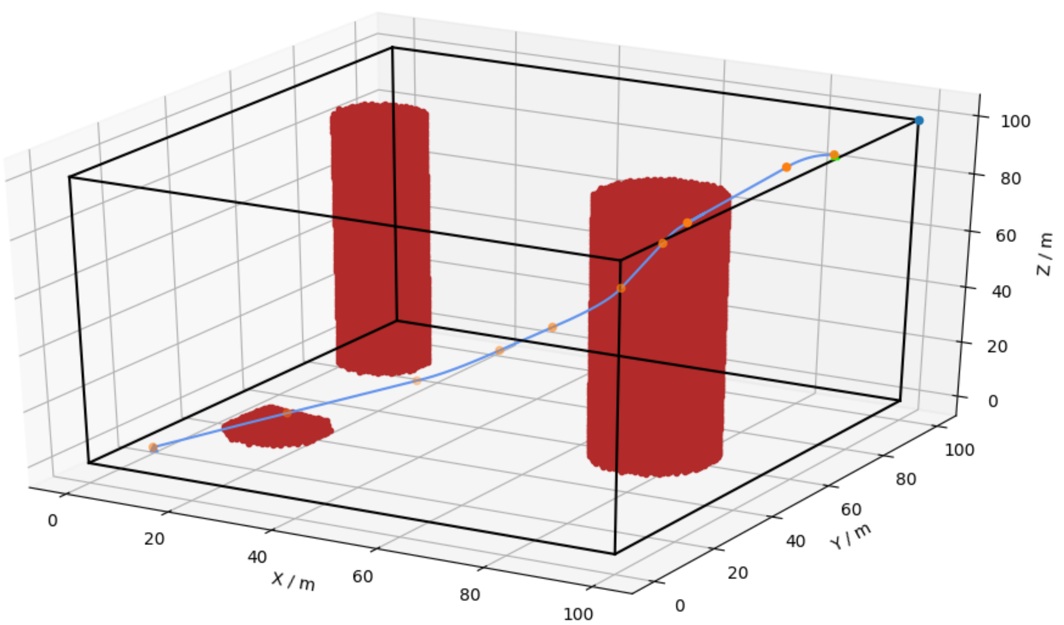

Figure 3.19: Example trajectory for piece-wise B – Spline trajectory.

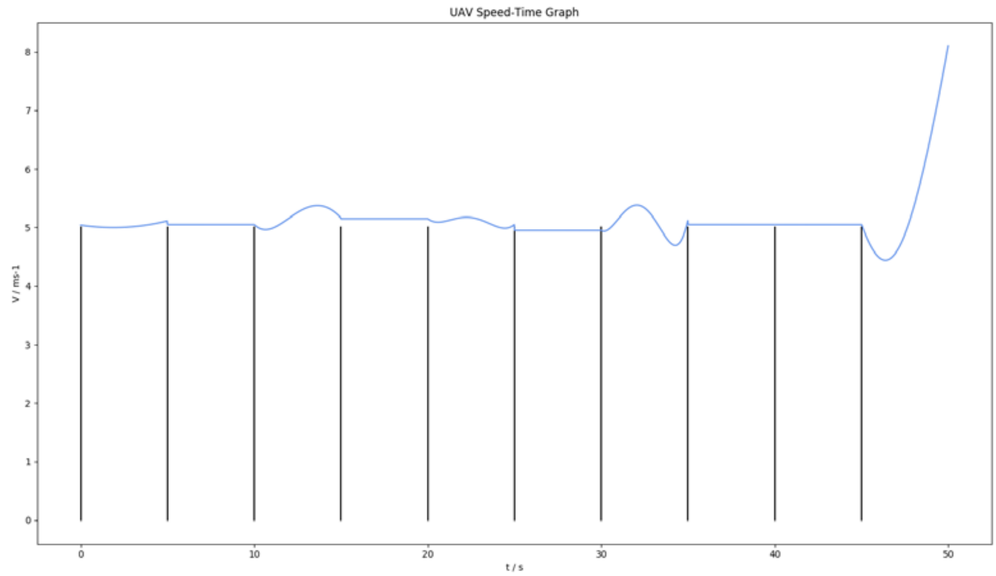

Figure 3.20: Velocity – Time graph for piece-wise B – Spline trajectory.

4. RESULTS AND DISCUSSION

In this section, the results of the algorithm are given and is discussed. Experiments involving five UAVs were performed. It is obvious that the algorithm can be expanded and applied to include more UAVs. In order to encourage the converging of UAV trajectories into a single trajectory, instead of each UAV generating separate PRMs, the same PRM is shared among all of the UAVs. Furthermore, the initial UAV positions are placed close to each other.

The experiment is as follows. Random obstacles are generated and each UAV finds a viable path using its starting position. In most cases, these paths converge to two or less paths. Initial trajectories are generated and checked for collisions with other UAVs. If collisions are present, the time-scaling is applied on the trajectories. The resulting paths are plotted with respect to time and shown to be collision free. A new obstacle is placed randomly such that it lies on UAV 1’s path. The required steps for collision avoidance are followed and the resulting trajectories are once again checked for collisions among each other. Figure 4.1 shows the initial trajectories found for five UAVs in unique colors. The goal is given as a red triangle. In the following trajectory graphs, only the waypoints are shown to reduce clutter and improve the visibility of the graph.

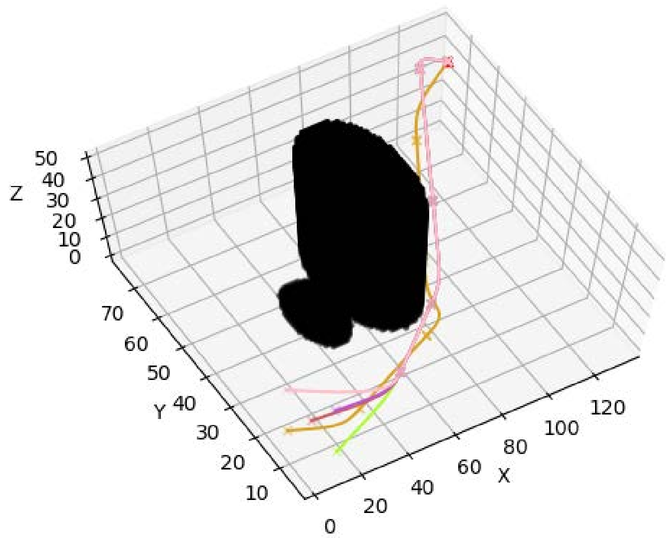

Figure 4.1: Initial trajectories of five UAVs.

As can be seen, with the exception of the pink and purple trajectories (which converge with each other), the other trajectories converge at some point. This is better seen in Figure 4.2 to Figure 4.5. The knot vectors for the UAVs are all the same and are uniform:

$$knot\ vector = \begin{bmatrix} 0 & 0 & 0 & 0 & 0.769 & 1.538 & 2.307 & 3.076 & 3.846 & 4.615 & \ldots \\ & 5.384 & 6.153 & 6.923 & 7.692 & 8.461 & 9.230 & 1 & 1 & 1 & 1 \end{bmatrix}$$

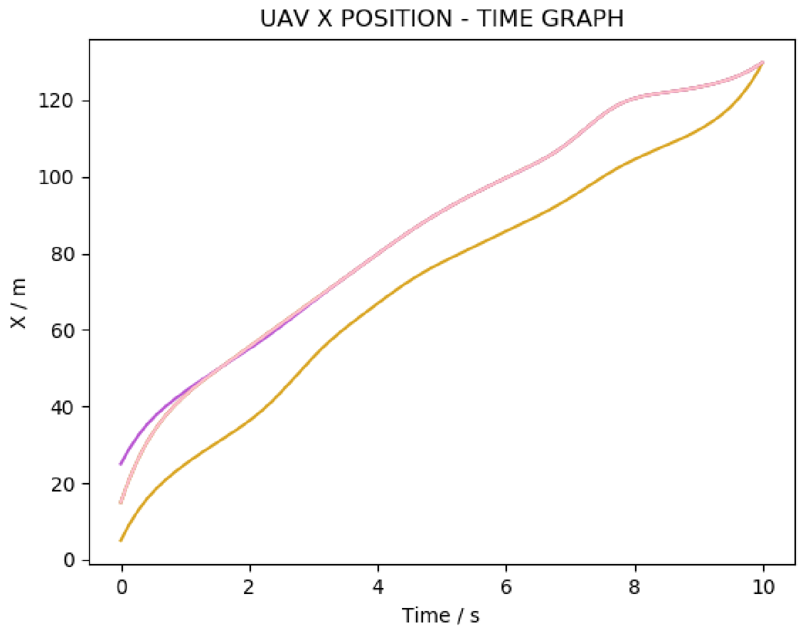

Figure 4.2: The change in x position with time for five UAVs.

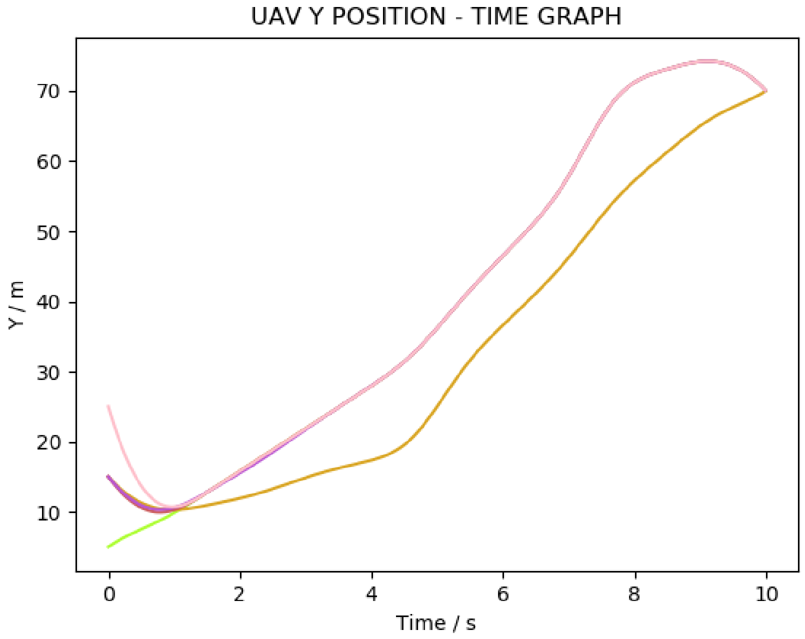

Figure 4.3: The change in y position with time for five UAVs.

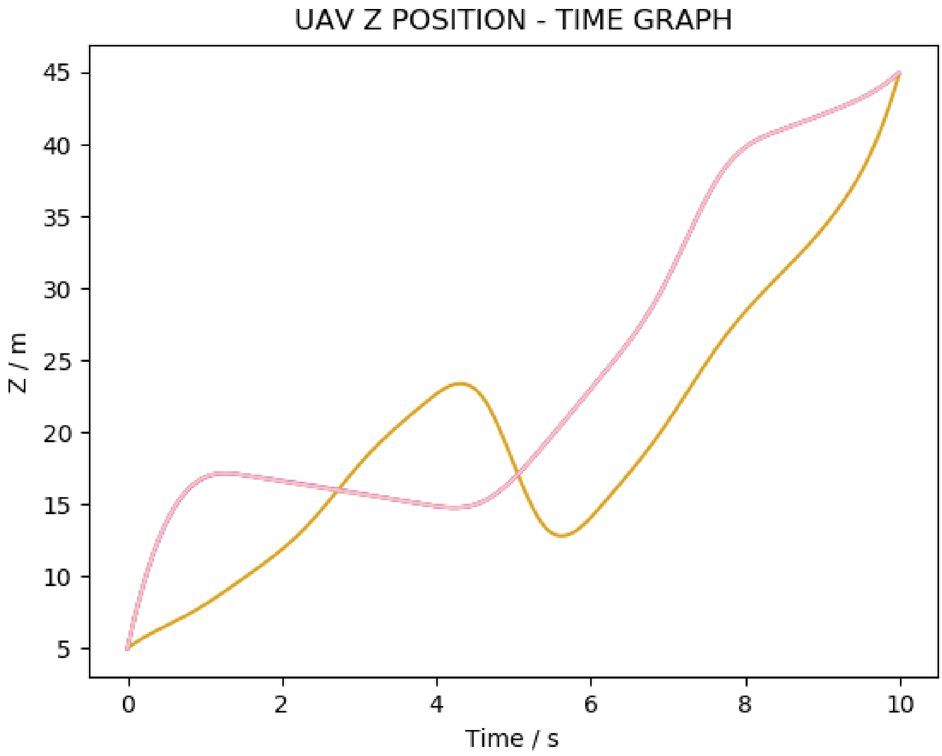

Figure 4.4: The change in z position with time for five UAVs.

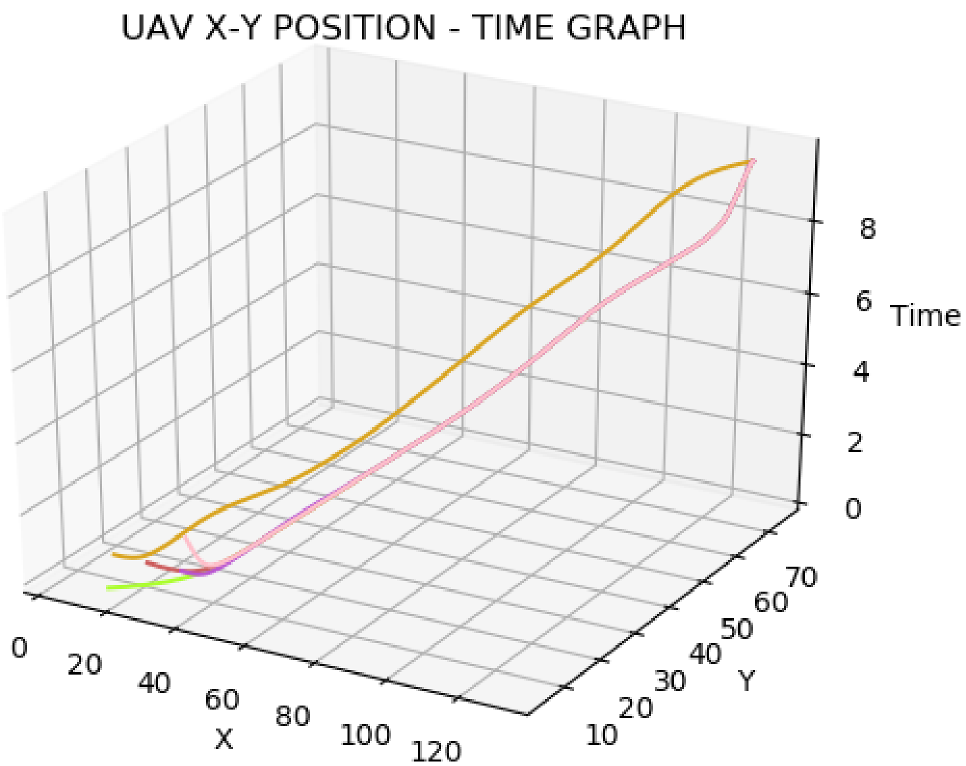

Figure 4.5: The change of x – y position of with time for five UAVs.

After time scaling, the knot vectors for the five UAVs are given in Table 4-1. The resulting trajectory remains the same but its variation in time can be seen in Figure 4.6 to Figure 4.9. It can be seen that for at least one axis the UAVs do not have the same position at the same time, thereby not colliding. The trajectories can be said to be elongated or expanded in time.

Table 4-1: Knot vectors of b-spline trajectories after time-scaling.

UAV No:	Knot Vector
UAV 1	$\begin{bmatrix} 0 & 0 & 0 & 0 & 0.769 & 1.538 & 2.307 & 3.076 & 3.846 & 4.615 & \dots \\ 5.384 & 6.153 & 6.923 & 7.692 & 8.461 & 9.230 & 10 & 10 & 10 & 10 & \end{bmatrix}$
UAV 2	$\begin{bmatrix} 0 & 0 & 0 & 0 & 0.769 & 1.538 & 2.307 & 3.076 & 3.846 & 4.615 & \dots \\ 5.384 & 6.153 & 6.923 & 7.692 & 8.461 & 9.230 & 10 & 10 & 10 & 10 & \end{bmatrix}$
UAV 3	$\begin{bmatrix} 0 & 0 & 0 & 0 & 1.269 & 2.038 & 2.807 & 3.576 & 4.346 & 5.115 & \dots \\ 5.884 & 6.653 & 7.423 & 8.192 & 8.961 & 9.730 & 10.5 & 10.5 & 10.5 & 10.5 & \end{bmatrix}$
UAV 4	$\begin{bmatrix} 0 & 0 & 0 & 0 & 1.769 & 2.538 & 3.307 & 4.076 & 4.846 & 5.615 & \dots \\ 6.384 & 7.153 & 7.923 & 8.692 & 9.461 & 10.230 & 11 & 11 & 11 & 11 & \end{bmatrix}$
UAV 5	$\begin{bmatrix} 0 & 0 & 0 & 0 & 2.269 & 3.038 & 3.807 & 4.576 & 5.346 & 6.115 & \dots \\ 6.884 & 7.653 & 8.423 & 9.192 & 9.961 & 10.730 & 11.5 & 11.5 & 11.5 & 11.5 & \end{bmatrix}$

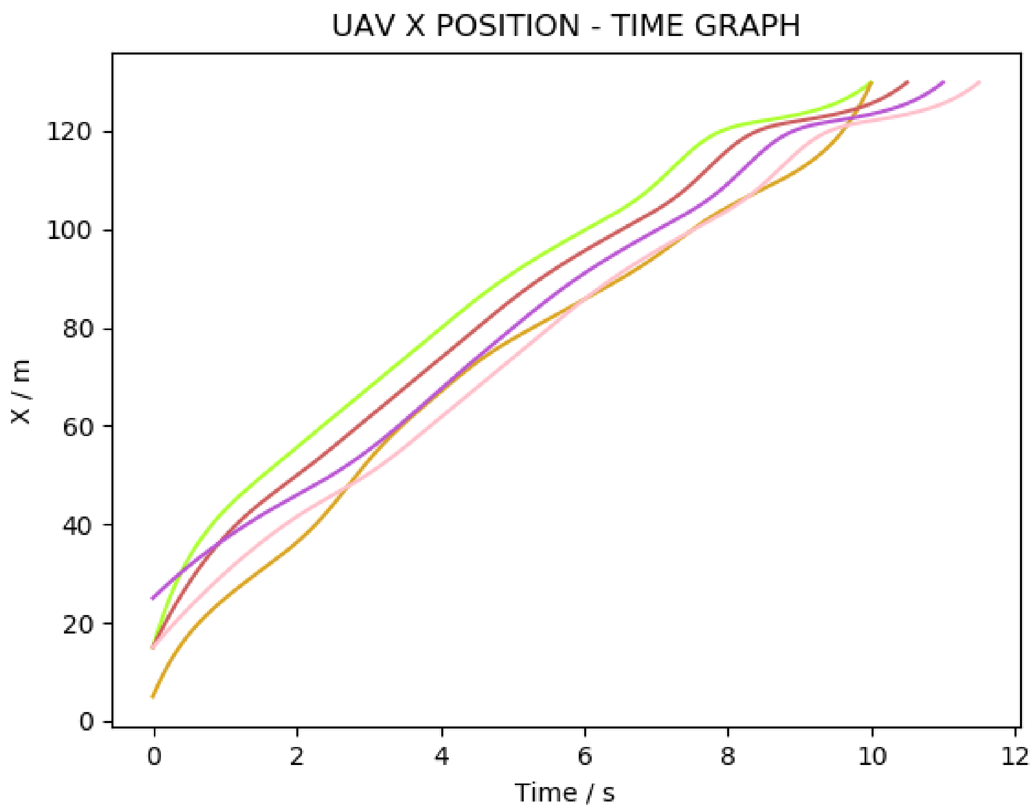

Figure 4.6: The change in x position with time for five UAVs after time-scaling.

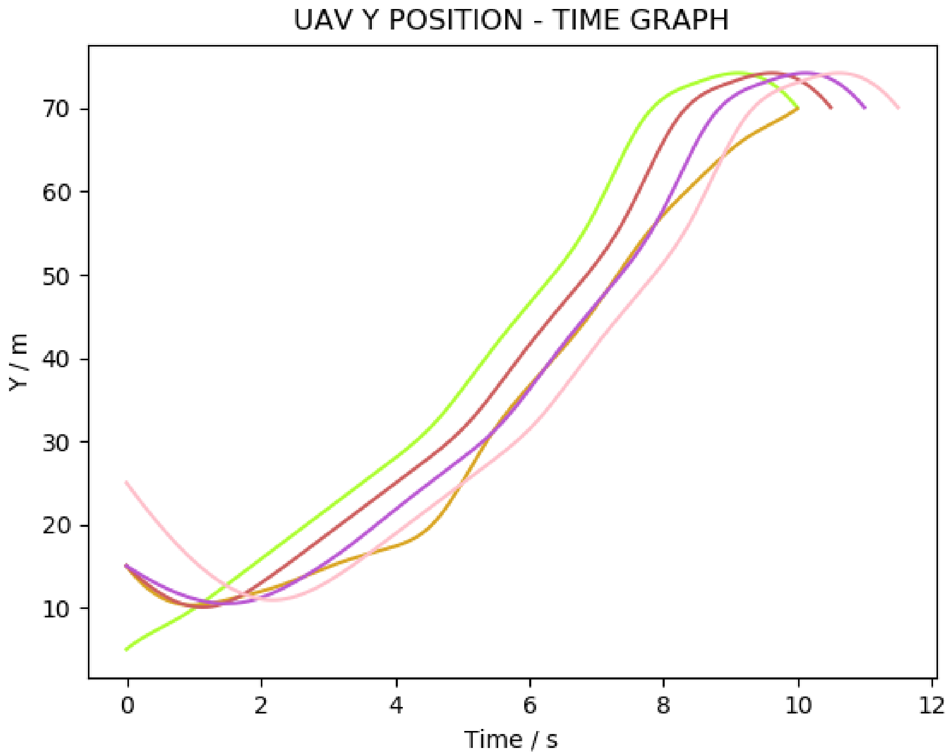

Figure 4.7: The change in y position with time for five UAVs after time-scaling.

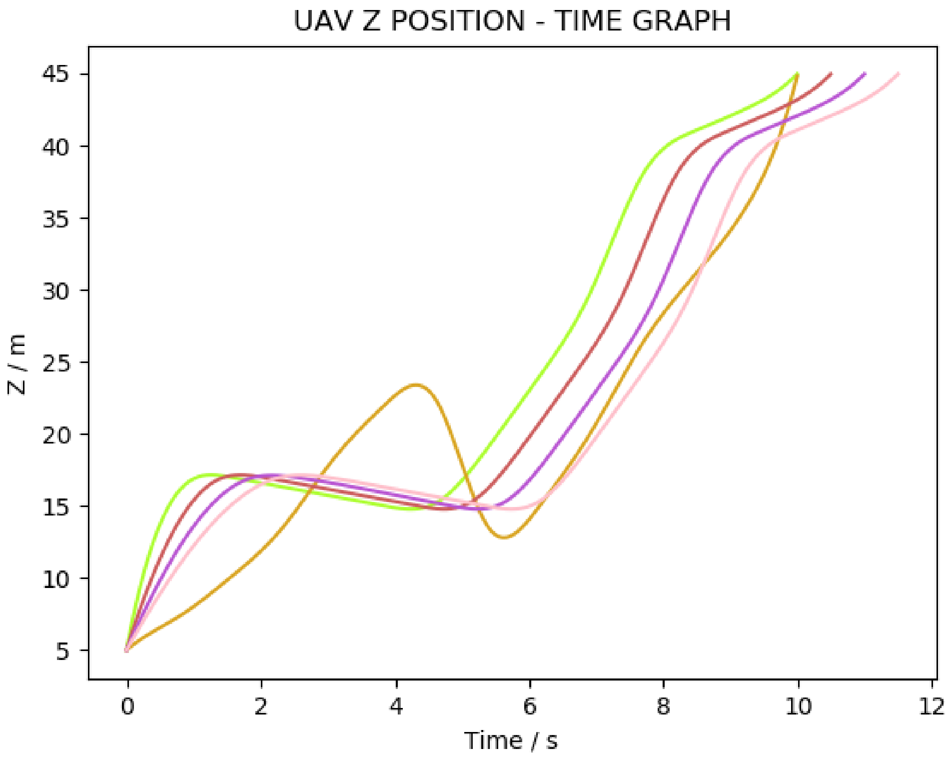

Figure 4.8: The change in z position with time for five UAVs after time-scaling.

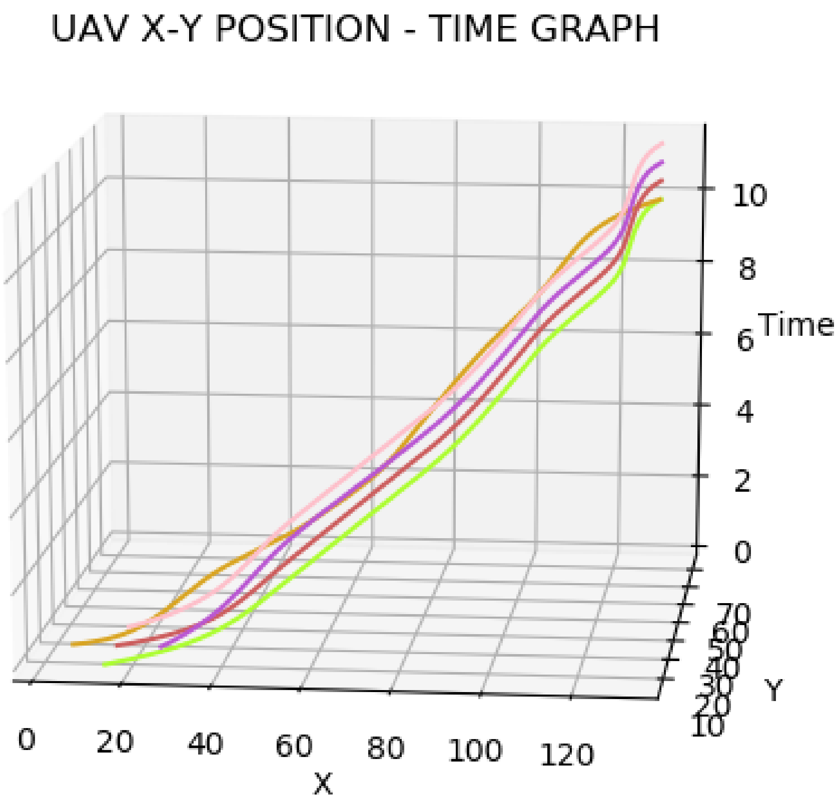

Figure 4.9: The change of x – y position of with time for five UAVs after time-scaling.

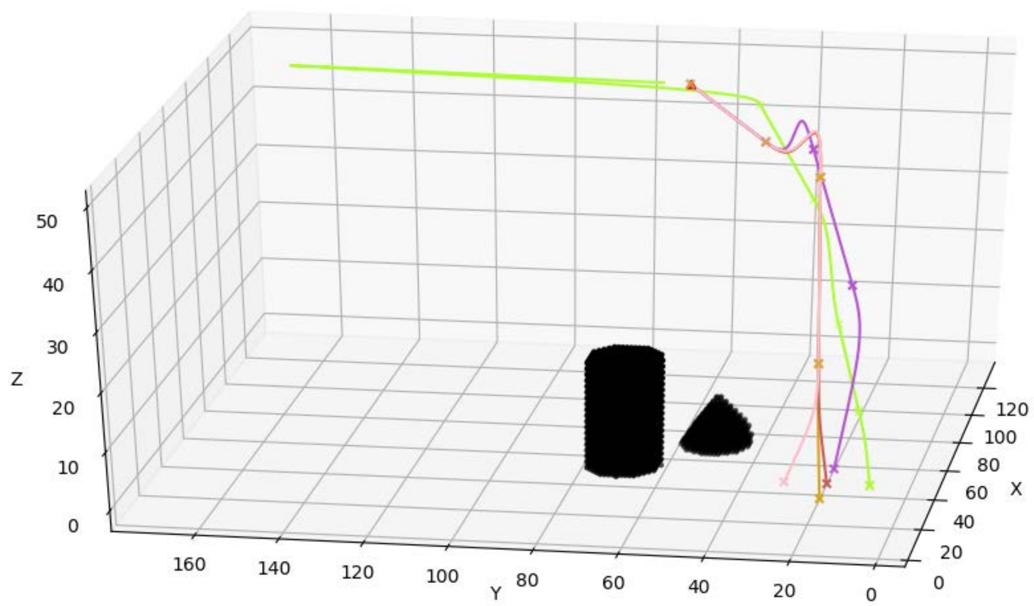

Figure 4.10: Initial trajectories of five UAVs for new static avoidance simulation.

In another simulation, new static obstacle avoidance is tested. The obstacles and initial trajectories of the UAVs are given in Figure 4.10. Other than the UAV with the green trajectory, which has a unique path, all other UAVs converge at the second and fourth waypoint. The corresponding time-scaled x – y – z positions in trajectories are plotted against time in Figure 4.11 to Figure 4.14.

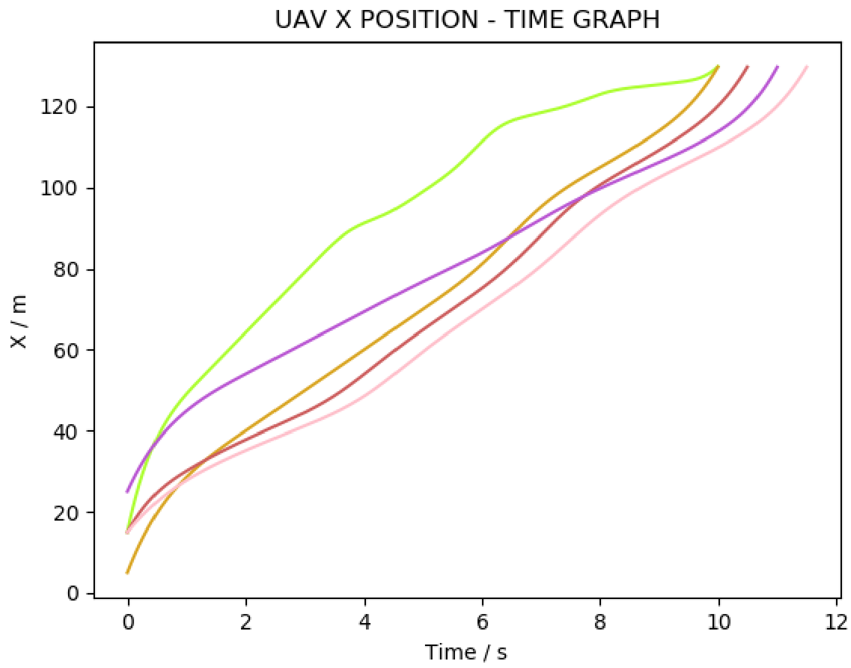

Figure 4.11: The change in x position with time for five UAVs after time-scaling.

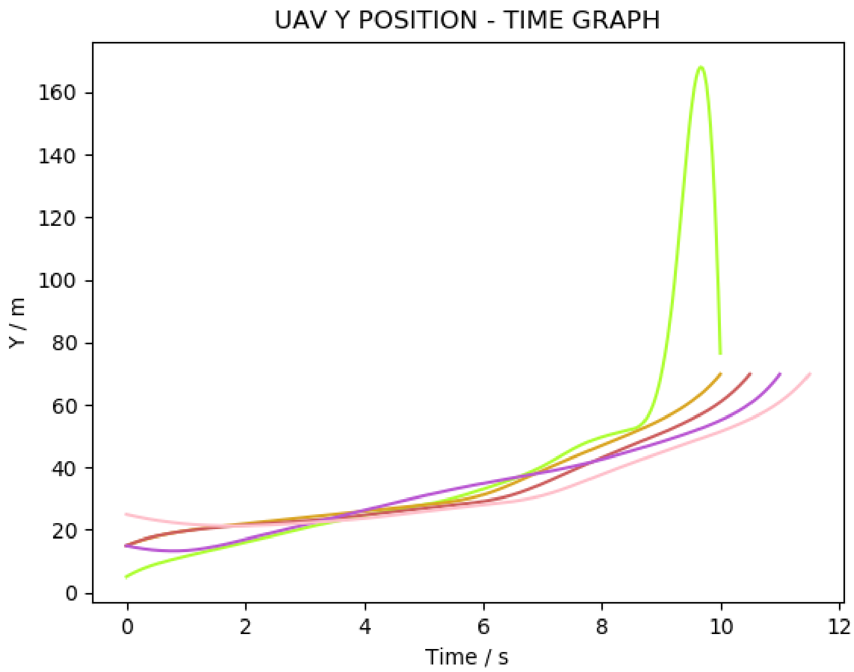

Figure 4.12: The change in y position with time for five UAVs after time-scaling.

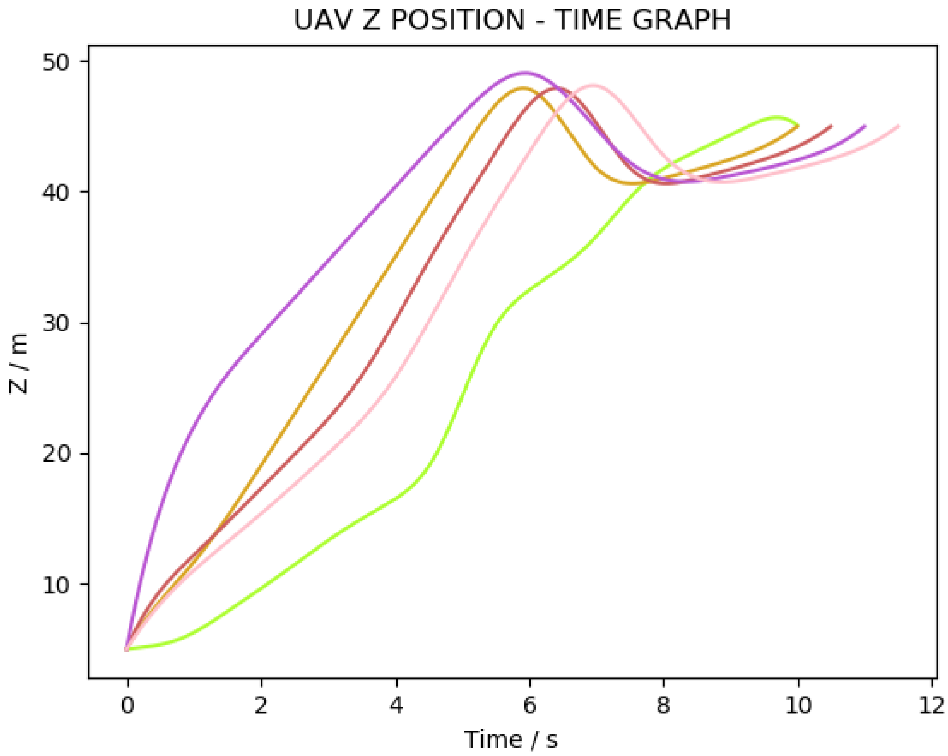

Figure 4.13: The change in z position with time for five UAVs after time-scaling.

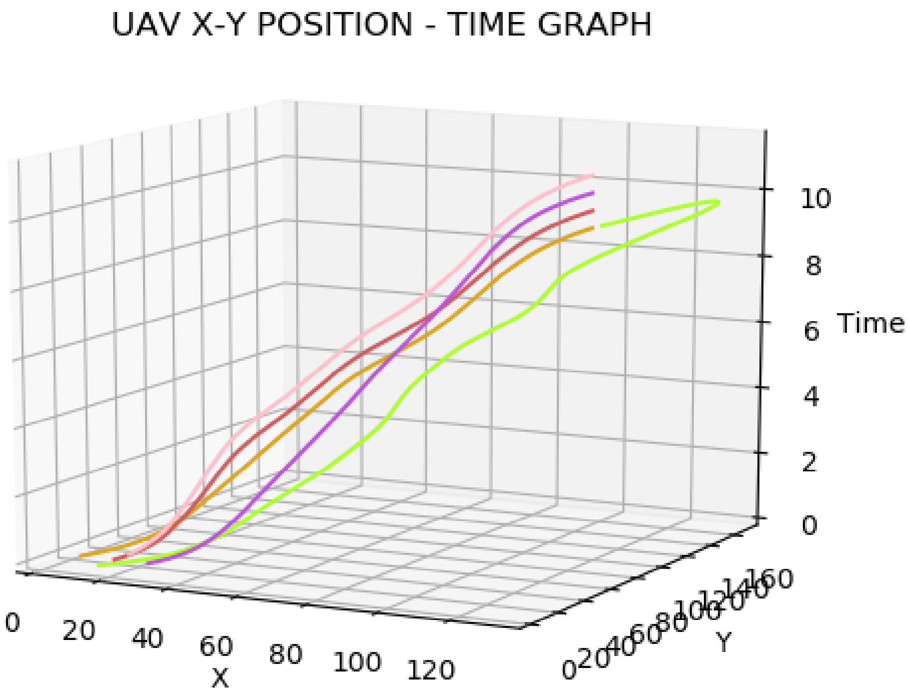

Figure 4.14: The change of x – y position of with time for five UAVs after time-scaling.

Figure 4.15 shows the new trajectories generated after the new static obstacle is detected. The new trajectories are plotted along with the previous ones in order to notice the difference in paths. The updated trajectories are plotted in a darker shade of the initial trajectories. It can be seen that only the UAV with the green trajectory has had to update its path. The time-scale plots are given in Figure 4.16 to Figure 4.19. Here, the plots are in the original color for each UAV.

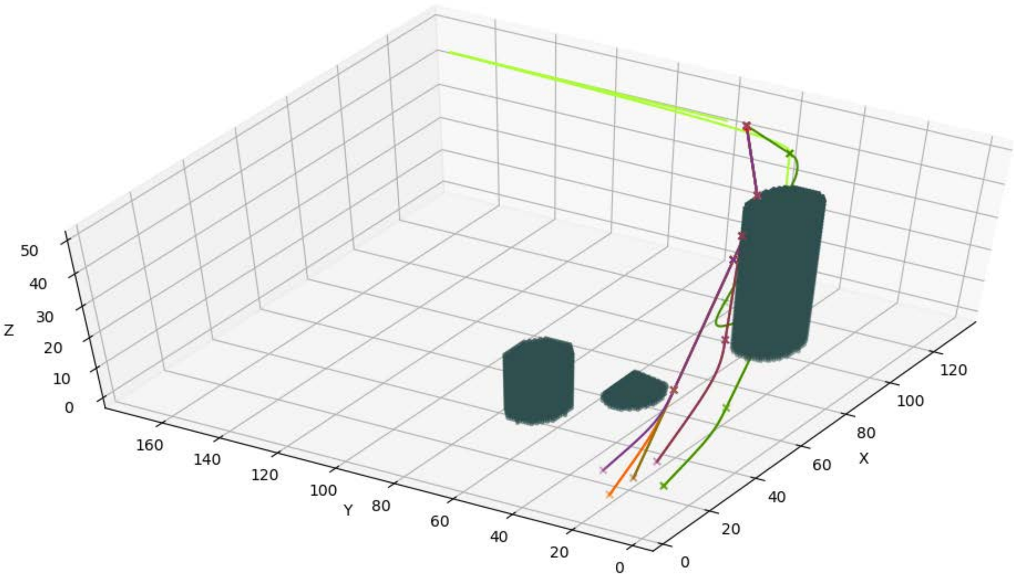

Figure 4.15: New trajectories for five UAVs after the detection of the new static obstacle.

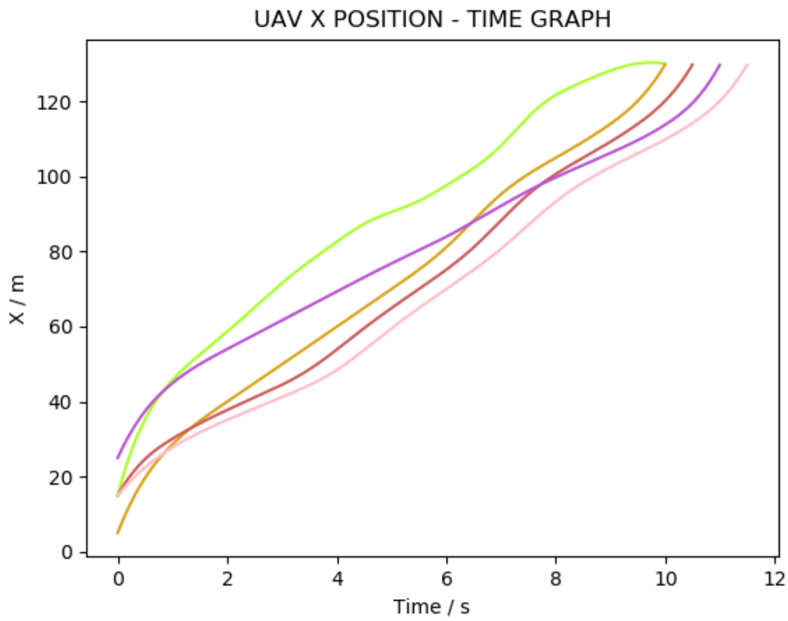

Figure 4.16: The change in x position with time for five UAVs after time-scaling for the updated trajectory.

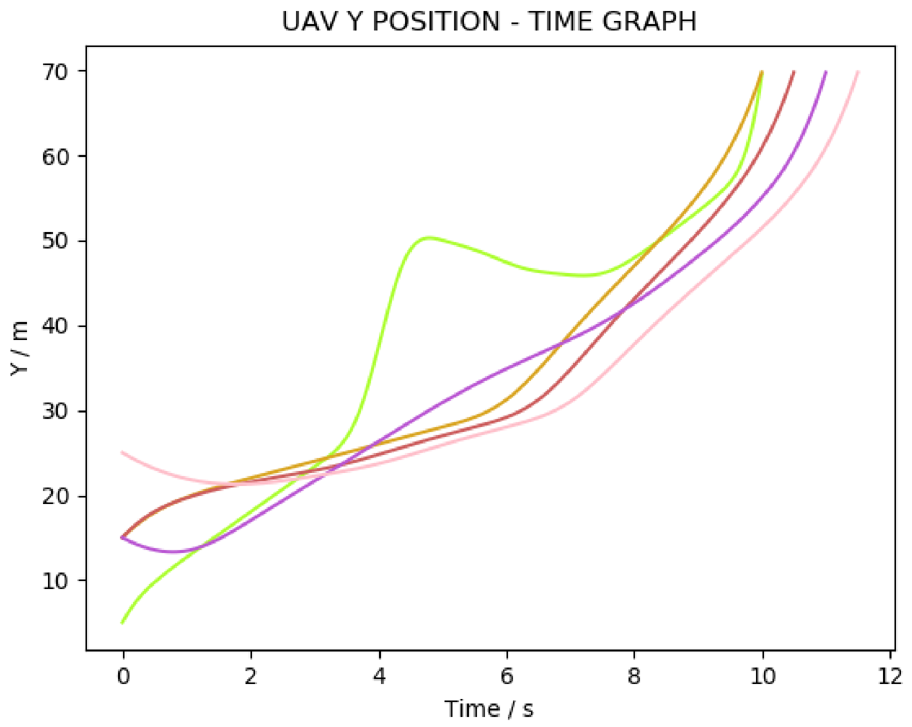

Figure 4.17: The change in y position with time for five UAVs after time-scaling for the updated trajectory.

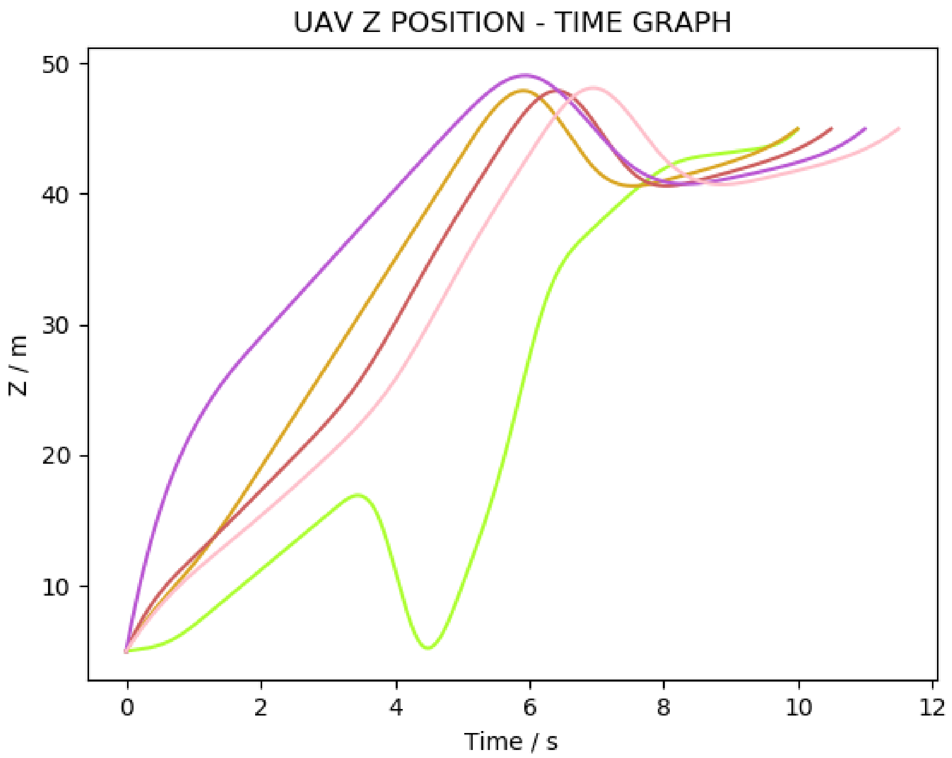

Figure 4.18: The change in z position with time for five UAVs after time-scaling for the updated trajectory.

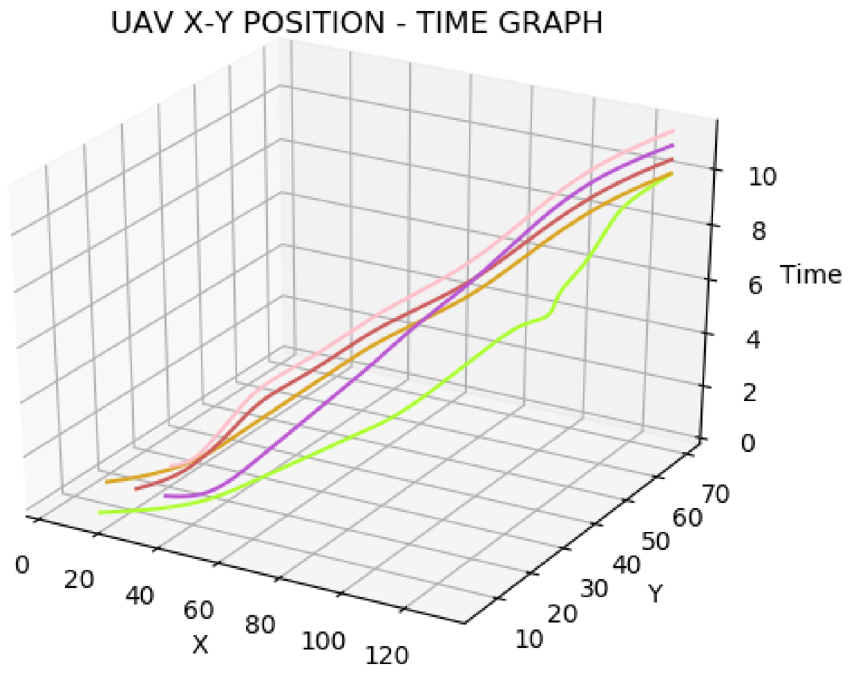

Figure 4.19: The change in x – y position with time for five UAVs after time-scaling for the updated trajectory.

5. CONCLUSION AND FUTURE WORKS

In this thesis, an algorithm for trajectory synchronization for swarm UAVs was developed and tested. The Probabilistic Roadmap planner was used alongside Dijkstra's shortest path algorithm to find a probabilistically optimal path initially in an environment with known, static obstacles, and again in the presence of new obstacles. The paths of the UAVs were guaranteed to converge by using the same roadmap for each UAV. The continuity and scalability properties of B-Spline curves were utilized in generating continuous time-scaled trajectories for multiple UAVs. Through simulations, it was shown that through optimal placement of pseudo-control points and shifting of knots in the knot vector, the trajectories of the UAVs do not collide.

Future works include using the differential flatness property to further optimize the generated trajectories by enforcing dynamic constraints on the trajectory derivatives.

REFERENCES

Aisha Walcott, N. I. (2001, 10 31). *MIT - Index of Courses.* Retrieved 05 02, 2019, from http://www.ai.mit.edu/courses/16.412J/lectures/

Anežka Chovancová, T. F. (2014). Mathematical Modelling and Parameter Identification of Quadrotor (a survey). *Procedia Engineering , 96*, 172-181.

Aurenhammer, F. (1991). Voronoi diagrams—a survey of a fundamental geometric data structure. *ACM Computing Surveys (CSUR), 23*(3), 345-405.

Beni, G. (1988). The Concept of Cellular Robotic System. *Proceedings IEEE International Symposium on Intelligent Control 1988.* Arlington, VA.

Bentley, J. L. (1987). Multidimensional binary search trees used for associative searching. *Communications of the ACM, 18*(9), 509-517.

Daniel Mellinger, V. K. (2011). Minimum Snap Trajectory Generation and Control for Quadrotors. *IEEE International Conference on Robotics and Automation.* Shangai.

David Hsu, J.-C. L. (2006). On the Probabilistic Foundations of Probabilistic Roadmap Planning. *International Journal of Robotics Research, 25*(7), 627-643.

Debnath S.K., O. R. (2018). A Review on Energy Efficient Path Planning Algorithms for Unmanned Air Vehicles. In *Computational Science and Technology. Lecture Notes in Electrical Engineering* (pp. 523-532). Singapore: Springer.

Dijkstra, E. W. (1959). A note on two problems in connexion with graphs. *Numerische Mathematik, 1*(1), 269-271.

Erol Şahin, W. M. (2004). Swarm Robotics: From Sources of Inspiration to Domains of Application. *International Workshop on Swarm Robotics.* Santa Monica, CA.

F. Pedregosa, G. V. (2011). Scikit-Learn: Machine Learning in Python. *Journal of Machine Learning Research, 12*, 2825-2830.

Fei Yan, Y.-S. L.-Z. (2013). Path Planning in Complex 3D Environments Using a Probabilistic Roadmap Method. *International Journal of Automation and Computing, 10*(6), 525-533.

Food and Agriculture Organization of the United Nations. (2018). *E-Agriculture in Action: Drones for Agriculture.* Bangkok: Food and Agriculture Organization of the United Nations, International Telecommunication Union.

Gerardo Beni, J. W. (1989). Swarm Intelligence in Cellular Robotic Systems. *Proceedings on NATO Advanced Workshop on Robots and Biological Systems.* Tuscany, Italy.

Hrabar, S. (2008). 3D Path Planning and Stereo-Based Obstacle Avoidance for Rotorcraft UAVs. *IEEE/RSJ International Conference on Intelligent Robots and Systems.* Nice.

Jacob E. Goodman, J. O. (2004). Chapter 39: Nearest Neighbours in high-dimensional spaces. In *Handbook of Discrete and Computational Geometry (2nd ed.).* CRC Press.

Jayesh N. Amin, J. D. (2006). A Fast and Efficient Approach to Path Planning for Unmanned Vehicles. *Proc. AIAA Guidance, Navigation and Control Conference.* Keystone, CO.

Jose Luis Sanchez-Lopez, J. P. (2017). A Robust Real-Time Path Planner for the Collision-Free Navigation of Multirotor Aerial Robots in Dynamic Environments. *2017 International Conference on Unmanned Aircraft Systems.* Miami, FL.

Judy E. Scott, C. H. (2017). Drone Delivery Models for Healthcare. *Proceedings of the 50th Hawaii International Conference on System Sciences.* Honolulu.

K. Fujimura, H. S. (1989). A hierarchical strategy for path planning among moving obstacles (mobile robot). *IEEE Transactions on Robotics and Automation, 5*(1), 61-69.

Karpowics, J. (2017). How are Emergency Response & Search and Rescue Professionals Using Drones in 2017? *Commercial UAV Expo.* Brussels-Las Vegas.

Kevin Judd, T. M. (2001). Spline based path planning for unmanned air vehicles. *AIAA Guidance, Navigation, and Control Conference and Exhibit, Guidance, Navigation, and Control and Co-located Conferences.* Montreal.

L. E. Kavraki, P. S. (1996). Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE Transactions on Robotics and Automation, 12*(4), 566-580.

Les Piegl, W. T. (1996). *The NURBS Book.* Springer.

Liang Yang, J. Q. (2016). Survey of Robot 3D Path Planning Algorithms. *Journal of Control Science and Engineering, 2016*, 22.

Liwei Huang, H. Q. (2016). A novel coordinated path planning method using k-degree smoothing for multi-UAVs. *Applied Soft Computing, 48*, 182-192.

Mariusz Wzorek, P. D. (2006). Reconfigurable Path Planning for an Autonomous Unmanned Aerial Vehicle. *2006 International Conference on Hybrid Information Technology.* Cheju Island, South Korea.

Mehmet Korkmaz, A. D. (2018). Comparison of Optimal Path Planning Algorithms. *14th International Conference on Advanced Trends in*

Radioelecrtronics, Telecommunications and Computer Engineering (TCSET). Lviv-Slavske.

Muhammad Umer Farooq, Z. Z. (2017). Quadrotor UAVs Flying Formation Reconfiguration with Collision Avoidance Using Probabilistic Roadmap Algorithm. *2017 International Conference on Computer Systems, Electronics and Control.* Dalian, China.

N. M. Amato, Y. W. (1996). A randomized roadmap method for path and manipulation planning. *Proceedings of IEEE International Conference on Robotics and Automation.* Minneapolis.

Omer Cetin, G. Y. (2016). Real-time Autonomous UAV Formation Flight with Collision and Obstacle Avoidance in Unknown Environment. *Journal of Intelligent & Robotic Systems, 84*(1-4), 415-433.

Overwatch. (2017). *How the Use of Drones is Revolutionising the Emergency Services - Overwatch - Secure, Integrated Drone Operations*. (Komodo) Retrieved April 24, 2019, from https://www.overwatchdroneapp.com/articles/how-the-use-of-drones-is-revolutionising-the-emergency-services/

P. B. Sujit, R. B. (2007). Cooperative Path Planning for Multiple UAVs Exploring an Unknown Region. *Proceedings of the 2007 American Control Conference.* New york City.

P. Vadakkepat, K. C.-L. (2000). Evolutionary artificial potential fields and their application in real time robot path planning. *Proceedings of the 2000 Congress on Evolutionary Computation.* La Jolla, CA.

Pehlivanoglu, Y. V. (2012). A new vibrational genetic algorithm enhanced with a Voronoi diagram for path planning of autonomous UAV. *Aerospace Science and Technology, 16*(1), 47-55.

Per Olof Pettersson, P. D. (2005). Probabilistic Roadmap Based Path Planning for an Autonomous Unmanned Helicopter. *Journal of Intelligent & Fuzzy Systems: Applications in Engineering and Technology, 17*(4), 395-405.

Qian Xue, P. C. (2014). Offline Path Planning and Onnline Replanning of UAVs in Complex Terrain. *Proceedings of 2014 IEEE Chinese Guidance, Navigation and Control Conference.* Yantai, China.

S. S. Ge, Y. C. (2002). Dynamic Motion Planning for Mobile Robots Using Potential Field Method. *Autonomous Robtos, 13*(3), 207-222.

Sai Chen, C. L. (2017). A Distributed Coverage Algorithm for Multi-UAV with average Voronoi Partition. *17th International Conference on Control, Automation, and Systems.* Jeju.

Seda, M. (2007). Roadmap Methods vs. Cell Decomposition in Robot Motion Planning. *Proceedings of the 6th WSEAS International Conference on Signal Processing, Robotics and Automation.* Corfu Island.

Tyner, B. (2006). *Benjamin Tyner - Purdue Statistics*. Retrieved 05 02, 2019, from http://www.stat.purdue.edu/~btyner/packages.html

Voronoi, G. (1908). Nouvelles applications des paramètres continus à la théorie de formes quadratiques. *Journal für die reine und angewandte Mathematik, 133,* 97-178.

Warren, C. W. (1989). Global Path Planning Using Artificial Potential Fields. *Proceedings of the 1989 International Conference on Robotics and Automation.* Scottsdale, AZ.

Yaohong Qu, Y. Z. (2015). A UAV Solution of Regional Surveillance Based on Pheromones and Artificial Potential Field Theory. *International Conference on Unmanned Aircraft Systems.* Denver.

Yap, P. (2002). Grid-Based Path-Finding. *Conference of the Canadian Society for Computational Studies of Intelligence.* Alberta.

YongBo Chen, J. Y. (2015). Path Planning for Multi-UAV Formation. *Journal of Intelligent & Robotic Systems, 77*(1), 229-246.

CURRICULUM VITAE

Name Surname : Halise Turkmen

Place and Date of Birth : Istanbul, Turkey – 14.10.1992

E-Mail : halise.turkmen@gmail.com

EDUCATION :

- **B.Sc.** : 2016, Marmara University, Faculty of Technology, Mechatronic Engineering
- **M.Sc.** : 2019, Istanbul Technical University, Graduate School of Science, Engineering and Technology, Mechatronic Engineering

PROFESSIONAL EXPERIENCE AND REWARDS:

- 2015-2016 Istanbul Technical University at the Mechatronic Education and Research Center.
- 2016-2018 Istanbul Technical University at the Aerospace Research Center