ARTIFICIAL INTELLIGENCE BASED DYNAMIC MISSION PLANNING WITH PROBABILISTIC ROADMAPS AND VORONOI DIAGRAMS USING PREDICTIVE LAUNCH ACCEPTABILITY REGION APPROACH

A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES
OF
MIDDLE EAST TECHNICAL UNIVERSITY

BY

MUSTAFA RAŞİT ÖZDEMİR

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR
THE DEGREE OF MASTER OF SCIENCE
IN
COMPUTER ENGINEERING

SEPTEMBER 2021

Approval of the thesis:

ARTIFICIAL INTELLIGENCE BASED DYNAMIC MISSION PLANNING WITH PROBABILISTIC ROADMAPS AND VORONOI DIAGRAMS USING PREDICTIVE LAUNCH ACCEPTABILITY REGION APPROACH

submitted by **MUSTAFA RAŞİT ÖZDEMİR** in partial fulfillment of the requirements for the degree of **Master of Science in Computer Engineering Department, Middle East Technical University** by,

Prof. Dr. Halil Kalıpçılar
Dean, Graduate School of **Natural and Applied Sciences** _____

Prof. Dr. Halit Oğuztüzün
Head of Department, **Computer Engineering** _____

Assoc. Prof. Dr. Seyda Ertekin
Supervisor, **Computer Engineering, METU** _____

Examining Committee Members:

Prof. Dr. Faruk Polat
Computer Engineering, METU _____

Assoc. Prof. Dr. Seyda Ertekin
Computer Engineering, METU _____

Prof. Dr. Mehmet Reşit Tolun
Computer Engineering, Konya Food and Agriculture University _____

Date: 02.09.2021

I hereby declare that all information in this document has been obtained and presented in accordance with academic rules and ethical conduct. I also declare that, as required by these rules and conduct, I have fully cited and referenced all material and results that are not original to this work.

Name, Surname: Mustafa Raşit Özdemir

Signature :

ABSTRACT

ARTIFICIAL INTELLIGENCE BASED DYNAMIC MISSION PLANNING WITH PROBABILISTIC ROADMAPS AND VORONOI DIAGRAMS USING PREDICTIVE LAUNCH ACCEPTABILITY REGION APPROACH

Özdemir, Mustafa Raşit
M.S., Department of Computer Engineering
Supervisor: Assoc. Prof. Dr. Seyda Ertekin

September 2021, 95 pages

In this thesis, dynamic air-to-surface mission planning strategies based on probabilistic roadmaps and Voronoi diagrams using predictive launch acceptability region approach are proposed for opportunity targets in order to strengthen decision support capabilities of aircraft.

Air-to-surface missions are planned in ground support systems and loaded to aircraft before the mission begins. This means that all the waypoints which should be followed during an air-to-surface mission are planned according to various threats and geographical formations. However, opportunity targets sometimes endanger aircraft safety because pilots may be obliged to deviate from planned waypoints in order to destroy the target which is unexpectedly appeared.

First of all, threats on battlefields are modeled by ellipsoids, and geographical formations are simulated by geoTIFFs. Then, predictive launch acceptability region queries are modeled, and a strategy is developed to designate a release state. In the first proposed method, a probabilistic roadmap algorithm with Dubins path distance is developed to form a connected graph that will connect the start and the goal states

in collision-free space. In the second proposed method, Voronoi diagram is generated according to threats in order to generate a roadmap. The shortest path between the start and the goal state in generated roadmaps is derived by Dijkstra's shortest path algorithm for both of the proposed methods. For Voronoi diagram-based method, a typical algorithm is developed in order to optimize the output of Dijkstra's shortest path algorithm. The optimized path is enhanced according to geographical formations by extracting the maximum envelope of elevation profile of the path using Hilbert transform.

Finally, proposed methods are analyzed in terms of convergence rate, mean trajectory length, elapsed time and compared with each other. For the probabilistic roadmap-based method, minimum trajectory length is observed as $6882.8\ m$, convergence rate is observed between 94% and 100% with number of samples is greater than 6000 and the maximum permitted length of Dubins path between two samples is greater than $450\ m$, and minimum execution time is observed as $62\ s$. Mean trajectory length, average execution time, and convergence rate are observed as $7192.6\ m$, $0.60\ s$, and 100%, respectively for Voronoi Diagram-based method. Results show that dynamic mission planning can be accomplished for opportunity targets using a predicted release state with a sub-optimal trajectory, admissible elapsed time, and full convergence rate.

Keywords: artificial intelligence, dynamic mission planning, launch acceptability region, Voronoi diagram

ÖZ

KESTİRİMCİ FIRLATMAYA UYGUNLUK BÖLGESİ KULLANARAK OLASILIKSAL YOL HARİTALARI VE VORONOİ DİYAGRAMLARI İLE YAPAY ZEKA TABANLI HAVA-YER GÖREV PLANLAMA

Özdemir, Mustafa Raşit
Yüksek Lisans, Bilgisayar Mühendisliği Bölümü
Tez Yöneticisi: Doç. Dr. Seyda Ertekin

Eylül 2021 , 95 sayfa

Bu tezde, uçakların karar destek sistemlerini güçlendirmek için fırsat hedeflerine yönelik Kestirimci Fırlatmaya Uygunluk Bölgesi kullanılarak Olasılıksal Yol Haritaları ve Voronoi diyagramları ile tasarlanan hava-yer dinamik görev planlama stratejileri önerilmiştir.

Hava-yer görevleri, görev başlamadan önce yer destek sistemlerinde planlanır ve uçağa yüklenir. Dolayısıyla, hava-yer görevi boyunca takip edilmesi gereken yaklaşma noktaları çeşitli tehditlere ve coğrafi yerşekillerine göre planlanır. Fakat, fırsat hedefleri bazen uçağın güvenliğini tehdit eder, çünkü aniden ortaya çıkan hedefler için pilotlar daha önceden planlanmış olan yaklaşma noktalarından sapmak zorunda kalmaktadır.

Öncelikle, savaş alanındaki tehditler elipsoidler şeklinde ve coğrafi yerşekilleri geoTIFF'ler kullanılarak modellenmiştir. Daha sonra, Kestirimci Fırlatmaya Uygunluk Bölgesi sorguları modellenmiş ve hedef yaklaşma noktası belirlemek için bir strateji

geliştirilmiştir. Sonra, uçağın başlangıç ve hedef durumlarını bağlayan bir çizge oluşturmak amacıyla; önerilen birinci metotta güvenli alanda Dubins mesafesi ile Olasılıksal Yol Haritası kullanılarak yol haritaları, ikinci metotta ise tehditler merkez alınarak Voronoi diyagramları oluşturulmuştur. İki yöntemde de, başlangıç durumundan hedef durumuna olan en kısa güzergah Dijkstra'nın en kısa yol algoritmasıyla bulunmuştur ve Dijkstra'nın en kısa yol algoritmasının çıktısının optimizasyona ihtiyaç duyduğu değerlendirilmiştir. Bu yüzden, Voronoi diyagramı baz alınarak geliştirilen yöntem için özgün bir yörünge optimizasyonu stratejisi geliştirilmiştir. Daha sonra optimize edilen güzergahın bürümü Hilbert dönüşümü kullanılan bir yöntemle bulunmuş ve elde edilen bürüm coğrafi yerşekillerinden kaçınmak amacıyla kullanılmıştır.

Son olarak önerilen metotlar; yakınsaklık oranı, ortalama güzergah uzunluğu ve işletim süresi yönlerinden analiz edilmiş ve birbirleriyle karşılaştırılmıştır. Olasılıksal Yol Haritası tabanlı metotta minimum güzergah uzunluğu $6882.8\ m$ olarak, minimum işletim süresi $62\ s$ olarak, ve örnek sayısı 6000'den ve iki örnek arasında izin verilen en büyük Dubins yörüngesi uzunluğu $450\ m$'den büyük olduğunda yakınsaklık oranı $\%94$ ile $\%100$ arasında gözlemlenmiştir. Voronoi diyagramı tabanlı metot için ise, ortalama güzergah uzunluğu, ortalama işletim süresi ve yakınsaklık oranı sırasıyla $7192.6\ m$, $0.60\ s$ ve $\%100$ olarak gözlemlenmiştir. Sonuçlar, dinamik görev planlamanın fırsat hedefleri için tahminlenen hedef yaklaşma noktası ile en uyguna yakın bir güzergah, kabul edilebilir bir işletim süresi ve tam yakınsaklık oranıyla gerçekleştirilebilirliğini göstermektedir.

Anahtar Kelimeler: yapay zeka, dinamik görev planlama, kestirimci fırlatmaya uygunluk bölgesi, Voronoi diagramı

To my family

ACKNOWLEDGMENTS

I owe special thanks to my advisor Assoc. Prof. Dr. Seyda Ertekin. She always directed me to research and encouraged me to complete this work. I learned practicalness and to be result-oriented from her and she supported our studies trustfully all the time. I always feel comfortable with her valuable and stimulating advisory support. I am so proud to work with her.

I would like to thank Assoc. Prof. Dr. Erol Sahin. We learned a lot from him, especially in terms of engineering and project management. His lab, KOVAN Research Laboratory, was always open for our research group, Collaborator, in undergraduate years.

I also want to thank Prof. Dr. Uluç Saranlı, because my academic life is divided into two parts: before meeting Uluç Hoca and after meeting Uluç Hoca. I was an unsuccessful student before meeting him. Everything started with Mandelbrot set. I will never forget his support. He also advised me to take Introduction to Machine Learning course from Seyda Hoca back in the days and I met her in that course.

I would particularly thank the members of our undergraduate research group, Collaborator: Burak Hocaoğlu, Cansın Canberi, Tahsincan Köse and our team leader Güneş Sucu. We learned to be a team together and improved ourselves a lot. All very successful and I am proud of them.

I would like to offer my special thanks to my research partner and office mate Levent Cevher from Turkish Aerospace because of his kind support and diligence.

Last but surely not the least, I would like to thank my family: Mine, Bahri, Halil and Kerim. I also want to thank Zeynep for her complimentary support and warm love during my graduate years. They always motivate me to finish this work and I could not be successful without their precious support.

TABLE OF CONTENTS

LIST OF TABLES

TABLES

LIST OF FIGURES

FIGURES

LIST OF ABBREVIATIONS

1D	1 Dimensional
2D	2 Dimensional
3D	3 Dimensional
4D	4 Dimensional
AI	Artificial Intelligence
ALIS	Autonomic Logistics Information System
FPGA	Field-Programmable Gate Array
GiB	Gibibyte
GPGPU	General Purpose Graphics Processing Unit
HMD	Helmet-Mounted Display
HUD	Head-Up Display
LAR	Launch Acceptability Region
NED	North - East - Down
NIL	Not in List
PRM	Probabilistic Roadmap
PSAC	Plan for Software Aspects of Certification
RTCA	Radio Technical Commission for Aeronautics
SCMP	Software Configuration Management Plan
SDP	Software Development Plan
SQAP	Software Quality Assurance Plan
SVP	Software Verification Plan
UAV	Unmanned Aerial Vehicle

CHAPTER 1

INTRODUCTION

1.1 Motivation and Problem Definition

All the ground threats, targets, and geographical formations are identified by intelligence agencies or military resources before the mission begins. In ground support systems, like F-35's Autonomic Logistics Information System (ALIS) [1], waypoints of a mission that are suggested to be followed by pilots are planned and loaded to aircraft according to identified threats, targets, and geographical formations. If everything appears normal, pilots should not deviate from planned waypoints because of the dangerousness of theater of the war. However, when the mission is being performed, some unanticipated or moving targets or threats can be noticed by pilots or identified by intelligence agencies. For this reason, dynamic mission planning is a vital necessity in the sense of unanticipated or moving targets or threats.

Decision support systems play a significant role in today's warfare technologies for manned military vehicles. Conventional situational awareness and decision support systems become obsolescence everyday. Development of general-purpose graphics processing unit (GPGPU) and field-programmable gate array (FPGA) technology enables strong parallelism capability, which is demanded by artificial intelligence algorithms [2]. Even military vehicles must be designed and developed considering ethical values. On the other hand, in safety-critical systems like aircraft, integration and maintenance of artificial intelligence algorithms become a trouble. The autonomy of military vehicles is still hazardous in terms of military and civilian causalities. First of all, safety-critical functions could not be realized with artificial intelligence methods with an accuracy and convergence rate lower than $\%100$. Furthermore, algorithms

that implement safety-critical functions should have determinism in terms of behavior and time. Secondly, the integration, and maintenance of artificial intelligence could be expensive in terms of time and money. Change of cost of a safety-critical software is much more significant than change cost of a non-safety critical software because of certification processes like defined in RTCA/DO-178C [3] [4]. For these reasons, an artificial intelligence method must realize non-safety critical functions like generating recommendations and must be compatible with every condition; in other words there will not need maintenance frequently.

Another important topic about air missions is weapon aiming. Essential information about weapon aiming is provided to pilots with special weapon aiming symbology to be visualized on head-up displays (HUDs) or helmet-mounted displays (HMDs) [5]. Guided air-launched weapons generally calculate weapon aiming information as launch acceptability regions (LARs) [6]. Launch acceptability regions indicate the proper region to release the guided weapon. Launch acceptability region (LAR) calculation is done on the weapon side because recertification of aircraft software can be expensive. MIL-STD-1760 [7] standardized the electrical interface between air-launched weapons and aircraft. Since every type of weapon has its specific weapon aiming calculation process, weapon aiming calculations are realized on weapons in order not to change aircraft software. Thus, the electrical interface between weapon and aircraft and aircraft software do not change if a standardized message set is used.

Conventional dynamic mission planners directly aim target's location. However, strategic targets are generally protected by air defense systems. For this reason, directly aiming the target's location is usually dangerous. Instead, aircraft may suggest an in-LAR goal point that is not threatened by air defense systems.

In brief, air-to-surface mission planning is composed of planning the motion of aircraft from the start state to the release point through a safe trajectory, and designation of release point. In this study, fuel consumption of aircraft, weapon configurations and release sequences, and environmental conditions are not taken into consideration when planning a mission.

In our conference paper with L. Cevher and S. Ertekin [27], a method is proposed for artificial intelligence-based dynamic air-to-surface mission planning using pre-

dictive launch acceptability region (LAR) approach. The proposed method is based on probabilistic roadmap algorithm. Therefore, a huge number of nodes is required in a huge battlefield. The huge number of nodes in a probabilistic roadmap algorithm causes long execution times due to the algorithm's complexity, which is $O(V^2)$ where V is the number of vertices in the PRM graph. Since dynamic mission planning is a mission-critical task, execution time is vital for the algorithm's dynamism. Additionally, probabilistic roadmap algorithm does not always converge. Convergence rate has a vital importance because of determinism. Furthermore, the probabilistic roadmap-based method requires High Speed 1760 [7] interface to query a great number of launch acceptability region. In the 'Future Work' section of the conference paper, trajectory optimization and ground collision avoidance are also mentioned. In this thesis, there is a solution for each issue mentioned in the conference paper.

1.2 Proposed Methods and Models

The first proposed method, which is probabilistic roadmap-based, proposes an artificial intelligence-based dynamic mission planning approach based on probabilistic roadmap algorithm with Dubins path distance and predictive launch acceptability region queries. Threats, e.g., ground defense systems, are modeled as ellipsoids because their impact can differ with pitch angle at different rates.

Dubins path distance between two 4D aircraft states $< N, E, D, \psi >$ is calculated by Dubins path distance, because edges and nodes can be very close to threats by considering probabilistic nature of the probabilistic roadmap algorithm. According to probabilistic roadmap algorithm, a huge number of uniformly distributed aircraft states are generated in 4D space $< N, E, D, \psi >$. By combining probabilistic roadmap algorithm and Dubins path distance, all the aircraft motion is taken into consideration to avoid a possible collision.

A specified number of randomly distributed goal states are generated around target coordinates. By considering the minimum and maximum cross-section area of weapon and drag coefficients, the minimum and maximum trail are calculated with numerical iteration of projectile motion. If the distance between the possible goal state and the

target coordinates is not between the minimum and maximum trail, the possible goal state is eliminated. Then, possible goal states that are not eliminated are inserted to set of uniformly distributed aircraft states along the battlefield.

In the whole set of aircraft states, states inside of threat ellipsoids are eliminated, and an adjacency matrix representing probabilistic roadmap graph is generated using Dubins path distance between samples with a threshold. Then, Dijkstra's shortest path algorithm is used to find the closest possible goal state to the starting state. Since probabilistic roadmap algorithm inside extensive battlefields needs a huge number of uniformly distributed samples, the execution time of the algorithm will be too much. Furthermore, the algorithm's convergence rate is less than 100% because of the probabilistic nature of the probabilistic roadmap algorithm.

In the probabilistic roadmap-based method, a massive number of predictive launch acceptability region queries are performed. All of the successful (in-LAR) states which are derived from predictive launch acceptability region queries are inserted to the probabilistic roadmap graph in order to find the closest possible goal state and this causes more time complexity. However, in this thesis, the predictive launch acceptability region querying method is enhanced since too many possible goal states in PRM graph are redundant and inefficient. A greedy approach is used in specifying a goal state for the mission. The enhanced method halts after finding 10 successful (in-LAR) goal states.

Since the execution time is too much and the convergence rate is less than 100%, probabilistic roadmap algorithm is replaced by a customized Voronoi diagram-based path planning algorithm. Seeds are placed in the middle of threat regions, and graph nodes are calculated with an approach similar to [8]. Taking seeds in the middle of threat regions is effective in terms of time complexity, but this approach has some weaknesses: intersecting threats and ragged edges. Solutions are proposed for weaknesses of Voronoi diagram-based path planning algorithm. Furthermore, a novel greedy trajectory optimization method is developed for optimizing the trajectory. For every node of the trajectory, a path is searched for farther nodes with collision checks using line segment - circle intersection, and the most distant possible node is selected as the next node, i.e., an edge is placed between these nodes to form a trajectory. Then, the

edge is partitioned by segments with two times minimum turning radius $(2 \times R_{min})$ length. For every segment of the edge, a path is searched for farther nodes. If a path is found at the end of any segment, the remaining part of the edge is ignored, and an edge is placed between at the end of the segment and the available farthest node.

Finally, the optimized trajectory is modified according to geographical formations. First, elevation along the optimized trajectory is imported as a 1D signal. Hilbert transformation is used to find the upper envelope of the elevation signal. Then, peak points of the envelope signal are extracted. In the end, peak points are interconnected by considering the minimum flight path angle (γ_{max}) of the aircraft.

1.3 Contributions and Novelties

This thesis proposes a comprehensive approach to dynamic mission planning taking certification aspects of weapon system integration to aircraft. The proposed methods consist of autonomous release point designation, 3D path planning to designated release point by considering aircraft's flight capabilities, optimization of planned trajectory, and ground collision avoidance procedure.

Randomly distributed predictive launch acceptability region queries are used for autonomous designation of release point for the first time in the probabilistic roadmap-based method, and the approach is enhanced in this study. Thanks to this approach, aircraft will not directly aim at the target itself because targets are generally protected by air defense systems.

4D $< N, E, D, \psi >$ probabilistic roadmap algorithm with Dubins path distance and 4D $< N, E, D, \psi >$ customized Voronoi diagram based path planning algorithm with fillet flight path segment generation by considering maximum flight path angle and minimum turning radius of aircraft are designed in detail with predictive launch acceptability region queries. Weaknesses of Voronoi diagram based path planning algorithm are analyzed, and efficient solutions are proposed for these weaknesses. In the probabilistic roadmap-based method, the output of path planning algorithm is not optimized. For the Voronoi diagram-based method, a typical trajectory optimization method is developed considering the motional behavior of fixed-wing aircraft.

Finally, geographical formations are taken into consideration when the output trajectory is being formed. Geographical formations of the battlefield are as important as threat regions in terms of safety. A novel approach is developed using Hilbert transformation in order to avoid collisions with geographical formations.

Additionally, execution time and converge rate analyses of the Voronoi diagram-based method are much better than probabilistic roadmap-based method. The execution time of the Voronoi diagram-based method is at least 100 times less than the probabilistic roadmap-based method despite new features i.e., trajectory optimization and ground collision avoidance. The convergence rate of the Voronoi diagram-based method is 100%, which is better than the probabilistic roadmap-based method.

In a nutshell, the contributions of this thesis are as follows:

- From beginning to end, comprehensive analysis of dynamic mission planning process on military aircraft by considering certification aspects
- Usage of predictive launch acceptability region queries for release point designation
- Comprehensive path planning algorithms proposal with weakness analyses and solutions to those weaknesses
- A typical greedy trajectory optimization
- A novel ground collision avoidance method based on Hilbert transformation

1.4 The Outline of the Thesis

The outline of this thesis is as follows:

- In section 2, a literature survey about weapon aiming, mission planning, and related work is presented.
- In section 3, probabilistic roadmap-based method is presented. The proposed method is explained in detail in terms of modeling launch acceptability region,

flight path segment generation, and path planning. The simulation environment is described in detail. This section also includes results and conclusions about the probabilistic roadmap-based method.

- In section 4, the Voronoi diagram-based method is presented. Enhancements on flight path segment generation, enhancement on launch acceptability region queries, path planning with Voronoi diagram-based algorithm, trajectory optimization, and ground collision avoidance methods are explained in this section. This section focuses on results, and comparisons of proposed methods are presented as well.
- In section 5, conclusions are stated.

CHAPTER 2

LITERATURE SURVEY

2.1 Targeting of Air-launched Weapons

Air-launched weapons can be divided into three categories in terms of weapon system integration to aircraft and targeting: dumb weapons (ballistic bombs), smart air-to-surface weapons, and air-to-air missiles. Different types of weapons need different methods of targeting. Especially targeting a smart weapon, either air-to-surface or air-to-air, is very complex since targeting smart weapons has too many parameters.

Targeting an air-to-surface dumb weapon can be acquired by ballistic calculations. The release point of a dumb weapon has vital importance in terms of a successful shot. Dumb weapons make projectile motion through targets. This motion is affected by many factors like drag coefficients of weapon, air density, weather condition, wind speed and direction, the velocity and position of aircraft at the point of release, and even the earth's rotation around itself.

Smart weapons have aiming capability. Therefore, the point of release of smart weapons does not have that much vital importance. They generate a region that is suitable to release or launch themselves [6].

2.1.1 Continuously Computed Impact Point

Continuously Computed Impact Point is a weapon targeting approach of dumb weapons. In this approach, the impact point of the dumb weapon is continuously computed by using the ballistic coefficients of the weapon, aircraft's navigation data, and environmental conditions [9].

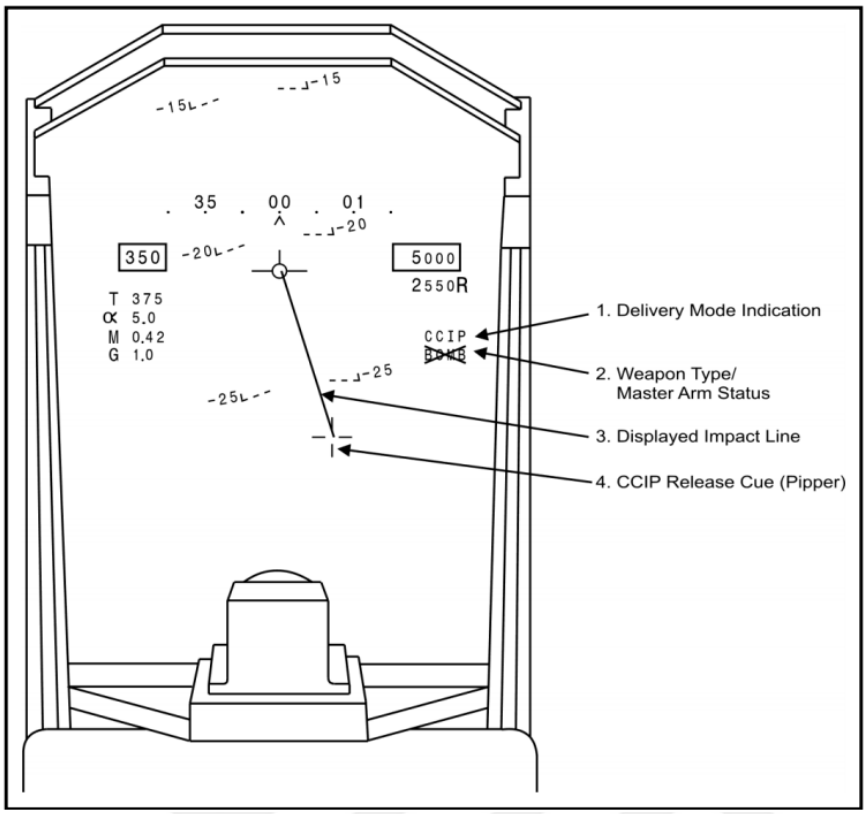

Figure 2.1: Continuously Computed Impact Point symbology taken from [10]

Actually, the impact point has a Gaussian distribution. Therefore, the calculated mean point will be shown to pilots. For this reason, the success rate of Continuously Computed Impact Point weapon targeting approach is not that much. By using powerful warheads with dumb weapons can increase the success rate of Continuously Computed Impact Point.

2.1.2 Continuously Computed Release Point

Continuously Computed Release Point is another weapon targeting approach of dumb weapons. In this approach, the release point of the dumb weapon is continuously computed according to aircraft's current navigation data, ballistic coefficients of the weapon, and environmental conditions [11].

In this approach, pilots try to reach Continuously Computed Release Point. With configuration, the release of the weapon can be automatized. In other words, aircraft

automatically release the weapon shortly after reaching the release point.

Since impact points of dumb weapons have a Gaussian distribution with wider covariances, the success rate of this aiming method is low as well. However, dumb weapons are pretty cheap and always accessible.

2.1.3 Launch Acceptability Region

Air-to-surface smart weapons have guiding capabilities, although they do not have propulsion systems. They manipulate impact point by controlling ailerons. For this reason, the set of release points for releasing a smart air-to-surface weapon form a region, named by launch acceptability region. When pilots release weapon in launch acceptability region, the weapon guarantees a sufficient circular error probable value [13][11]. Figure 2.2 represents an example of launch acceptability region symbology and Figure 2.3 represents logical explanation of it [12].

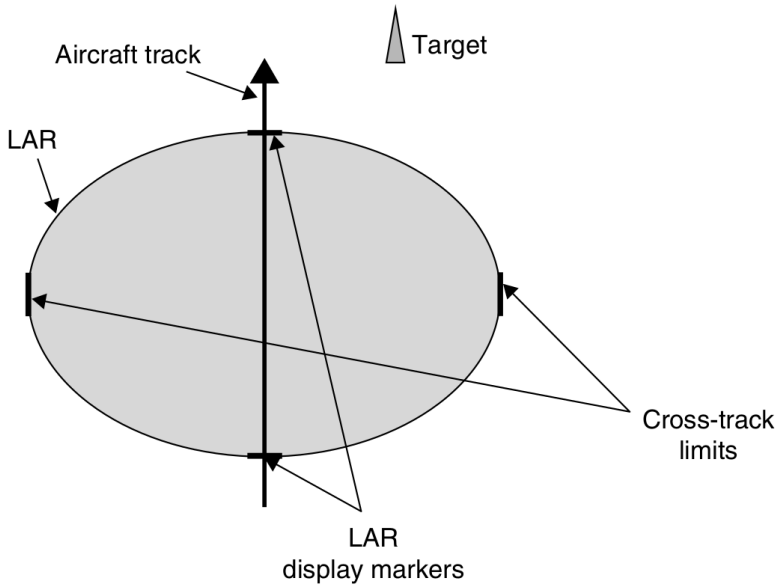

Figure 2.2: Launch Acceptability Region taken from [6]

Using a six degree of freedom model, which involves dynamics of the weapon's motion, the launch acceptability region can be calculated successfully. Launch acceptability region represents the minimum and maximum boundaries of the weapon re-

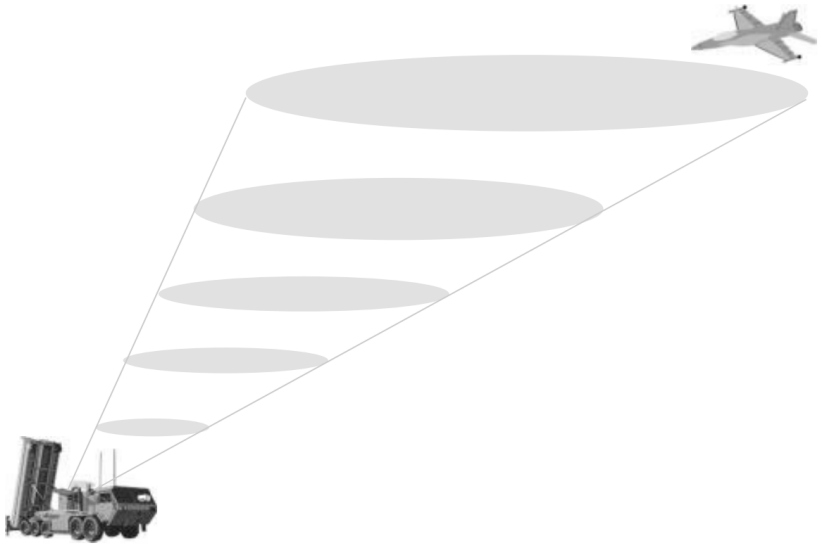

Figure 2.3: Launch Acceptability Region taken from [6]

lease according to current aircraft navigation data, dynamics of the weapon's aiming capability, and environmental conditions.

2.1.3.1 Predictive Launch Acceptability Region

Smart air-to-surface weapons generally calculate launch acceptability region by using navigation and environmental data transmitted by aircraft. This calculation is usually done by weapons because the change impact of aircraft software can cost much. In order to designate a proper release point for weapons, aircraft can transmit dummy navigation and environmental data to weapons for generating predictive launch acceptability region and querying whether the dummy state is sufficient for weapon release or not. This approach can be used for planning a trajectory through a predicted sufficient release point.

2.1.4 MIL-STD-1760E

In the early 1980s, the need for standardization of electrical interfaces of air-launched weapons arisen because there were many applied interfaces spread. The standardization activities of electrical interfaces of air-launched weapons have resulted in MIL-

STD-1760 standard, which covers all possible needs of weapon integrators. MIL-STD-1760 had several revisions in order to satisfy the actual needs of weapons. It consists of a variety of communication mediums and protocols: analog channels for high and low bandwidth video or audio transmission, MIL-STD-1553B [14], Fibre Channel for high-speed data transmission, and discrete channels [15]. Since MIL-STD-1553B has a data transfer limit of less than $1\ Mbps$, it falls behind to meet the needs of some mass data transfer and data communication requirements. In order to provide higher data transfer rates, interfaces for the protocols ANSI INCITS 356-2002 Fibre Channel - Audio Video [16], and ANSI T1 1.3 Project 1648-DT Fibre Channel - Avionics Environment Protocol for MIL-STD-1553B Notice 2 [17] are added to MIL-STD-1760E signal set. MIL-STD-1760E has high-speed data transmission capability with up and down Fibre Channel, which is explained in AS5653 [7]. Since it has $1.0625\ Gbaud$ data transmission rate, it can be said that new generation weapons may require high data transmission rate realization of their functionalities [6].

2.2 Certification of Avionics Software with DO-178C

Software usage in avionics systems and equipment spread in the early 1980s, and this caused many catastrophic accidents and resulted in software assurance's being one of the most safety-critical issues in the avionics domain [18]. In United States of America, Federal Aviation Administration claimed a certification process definition in developing avionics software. After that, Radio Technical Commission for Aeronautics established a committee in order to introduce a certification process for safety-critical avionics software, and RTCA/DO-178B was introduced in 1992 [19]. According to design assurance levels, required processes to develop software are defined in RTCA/DO-178B. Table 2.1 represents definition of design assurance levels and probability conditions of related levels [20].

With the enhances in software technology and software development, RTCA/DO-178B fell behind to meet the requirements of software assurance. Radio Technical Commission for Aeronautics and European Organization for Civil Aviation Equipment established another committee to cover new approaches in software, e.g., model-

based development approach, object-oriented programming, in order to introduce RTCA/DO-178C [3]. The committee was divided into several subgroups according to their specialties: Issues and Rationale Subgroup, Tool Qualification Subgroup, Model-based Development Subgroup, Object-Oriented Technology Subgroup, Format Methods Subgroup, Safety Related Considerations Subgroup, and Document Integration Subgroup [20].

Table 2.1: DO-178B Design Assurance Level Descriptions and Probability Conditions [19]

Level	Severity	Description	Probability
A	Catastrophic	Multiple fatalities	$< 10^{-9}$
B	Hazardous	Inability to operate aircraft	$< 10^{-7}$
C	Major	Highly increase in workload of aircrew	$< 10^{-5}$
D	Minor	Slightly increase in workload of aircrew	$< 10^{-3}$
E	No Effect	No safety effect	1

In addition to RTCA/DO-178C, some supportive documents are prepared to explain regulations and processes to be followed for additional considerations: DO-330 [22] for tool qualification, DO-331 [23] for model-based development, DO-332 [24] for object-oriented technology, and DO-333 [25] for formal methods.

RTCA/DO-178C defined software life cycle starting with the planning process. At the end of the planning process, software plan documents and standards below should be delivered:

- Plan for Software Aspects of Certification (PSAC)
- Software Development Plan (SDP)
- Software Verification Plan (SVP)
- Software Configuration Management Plan (SCMP)
- Software Quality Assurance Plan (SQAP)

PSAC consists of top-level plan of software life cycle and agreements with certification authority about documents. SDP explains plans about software development, e.g., life cycle, methodology, high-level software architecture, design assurance levels, tools to be used for development, transition criteria between phases. SVP defines procedures and guidelines for software verification engineers. SCMP defines configuration management environment and procedures throughout development and verification processes. SQAP defines plans for quality assurance to accomplish objectives of DO-178C. Additionally, software requirements standards, software coding standards, software design standards, and software modeling standards should be prepared to provide rules and guidelines for software engineers. After the planning process, the development process and integral process will be handled. The development process consists of requirement, design, coding, and integration phases, and the integral process consists of verification process, configuration management process, quality assurance process, and certification liaison process [4].

DO-178C defines objectives for each design assurance level. Level A is the most safety-critical level, and it involves all the objectives of less safety-critical levels. For example, Level A requires modified condition/decision coverage analysis and object code traceability analysis in addition to objectives of Level B. In Table 2.2, required number of objectives for corresponding design assurance levels are represented for both DO-178B and DO-178C [4].

Table 2.2: Required # of Objectives for Corresponding Design Assurance Levels Defined in DO-178B and DO-178C [4]

Level	# of Objectives in DO-178C	# of Objectives in DO-178B
A	71	66
B	69	65
C	62	57
D	26	28
E	0	0

DO-178C also defines processes for some special topics like field-loadable software, user-modifiable software, software reuse, and configuration data. Additionally, any

change in certified software requires some analyses as defined in DO-178C. Traceability analysis should be done because it should be analyzed again that whether new changes are suitable for requirements, design elements, and software architecture. Sometimes memory management becomes trouble for embedded systems in terms of memory fragmentation, memory leak etc. Memory margin analyses should be done when a change has happened in software. Timing is crucial in safety-critical systems in terms of determinism. Execution time analyses should be done again when a change has happened in software. Control and data coupling should be analyzed because change may affect control or data flow in a bad way. Input / Output analyses should be performed because interfacing with peripheral devices may cause software failures. Engineers should ensure that operational characteristics like interrupt and exception handling, performance parameters do not have a negative effect on safety. Changes in software can change certification maintenance requirements. For example, the lighting system may be inspected after 200 flight hours after a new software version. Certification maintenance requirements should be analyzed carefully after a software change. Robust partitioning is another thing that has vital importance in safety-critical systems. Software changes should not affect protective mechanisms of partitioning. Partitioning should be analyzed after a change on software [4].

Change impact analysis and required processes can be very expensive for software companies. In this work, proposed methods consider the cost of change in software by considering procedures defined in DO-178C.

2.3 Mission Planning

Mission planning is preparing all the supportive information before the mission begins in order to achieve that mission's objectives in a safe and easiest way. Before the invention of modern military avionics, mission data files were prepared a bunch of printed-out documents. After the development of modern military avionics, mission data files are prepared and delivered as digital files. Missions are still prepared by experience aviation officers manually and loaded to aircraft before the mission begins [26]. Aviation officers should take care of geographical formations, intelligence data, opponent's inventory and capabilities, presumptive precautions are taken

by opponent, and finally, capabilities and characteristics of aircraft [27].

Aviation officers shall take intelligence data related to targets and threats on battlefield into account when preparing mission data files. Missions are planned approximately 24 hours ago before they begin, and mission data must be up-to-date [26]. Briefly, the success of intelligence agencies affects the success of missions positively [27].

Military aircraft have a variety of weapon configurations to carry out different roles in a mission. Weapon configuration is chosen according to scenario planned and contingencies. Furthermore, according to the length of the mission, even fuel tanks can be attached to stores of aircraft. Prognosticatively, counter forces take precautions against potential air missions. That's why pilots remain under stress during operation because of precautions taken by counter forces. Mission plans are helpful to reduce the stress of pilots by proposing waypoints, defining weapon release the sequence for the scenario. In a nutshell, mission plans are prepared for safe and easy handling of missions and lighten pilots' workload [27].

Dynamic mission planning need arises when the pilot is suggested to destroy a target that is not anticipated before and involved in mission data files uploaded to aircraft. In case of appearing of a target of opportunity [28], pilot may deviate from the previously planned trajectory. By considering the uncertainty of battlefield, aircraft and pilots may encounter various threats. Dynamic mission planning aims to guide pilots to a safe and easy trajectory toward the target. Some aircraft may have dynamic mission planning capability. However, it is not efficient without predicting a safe and reachable release point, and this can be dangerous for both pilot's and aircraft's safety [27].

Today's concept for manned aircraft is proposing a safe trajectory to pilots in order to accomplish the mission. Since the aircraft's position is changing dynamically, trajectories should be computed continually according to many considerations like threats, geographical formations, aircraft's dynamic motion. In robotics literature, there are many algorithms for generating a trajectory from the start state to the goal state. For fixed-wing aircraft domain, robot state can be replaced by aircraft state because we only need the number of degrees of freedom. Thrun et al. [29] explain the fundamental algorithms for motion planning.

Motion planners can be divided into number of characterization in robotics domain according to the tasks they did, e.g., navigation, coverage, localization, mapping. Navigation, in robotics domain, is the problem of planning a trajectory from one robot state to another robot state in a safe manner [29].

Motion planning algorithms generate a motion that can be optimal in some way: length, execution time, energy consumption, safety, etc. Generally, there is not an optimal solution for all the aspects. Some planners can generate shorter algorithms but with longer execution times. Another important consideration for motion planners is completeness. Complete motion planners always converge to a solution through the goal state. The completeness of motion planning algorithms is divided into two parts: resolution completeness and probabilistic completeness. Resolution completeness cares about the resolution of the algorithm in terms of a solution exists or not. On the other hand, probabilistic completeness analyzes the probability of success of the algorithm. Some motion planners generate plans according to the known environment, and the generated plan is executed during the whole motion. These kinds of motion planners are so-called offline planners. On the contrary, online planners are continually computing momentary plans while the agent is moving. Therefore, dynamic mission planning requirement of UAVs needs an online and complete motion planning algorithm by considering flight dynamics [29].

In this section, reviews of probabilistic roadmap algorithm, rapidly-exploring random trees algorithm and Voronoi diagrams in robotics literature are explained. Probabilistic roadmap algorithm was used in firstly proposed method in this thesis for the purpose of generating a safe and short trajectory. Rapidly-exploring random trees and probabilistic roadmap algorithm are sampling-based algorithms. However, rapidly-exploring random trees provide better computational efficiency. On the other hand, Voronoi diagrams can be used as a path planner, which is based on retract-like structures. Actually, Voronoi diagrams are widely used in statistical data analysis. However, they are incredibly computational efficient in wide environments with smooth collision objects [29].

Sampling-based algorithms employ randomly generated samples and paths using the edges between samples in order to generate trajectories between the start and the goal

state. Some motion planning algorithms can be very complex in terms of dimension and resolution. Sampling-based algorithms can routinely solve the problems that classical motion planners cannot solve because all the possible paths are subject to be a solution thanks to probabilistic behavior. There are many strategies for generating samples through solution space and connecting the generated samples in order to obtain a proper trajectory [29].

2.3.1 Probabilistic Roadmap

Probabilistic Roadmap algorithm is based on generating an undirected graph $G = (V, E)$ in collision free configuration space Q_{free}. Vertices of graph V are generated by some method e.g. uniform distribution along Q_{free}. Start state and the goal state are attached to V where q_{start} is start state and q_{goal} is goal state. Edges of the graph E is a set of collision free paths between $q^{'}$ and $q^{''}$ [29].

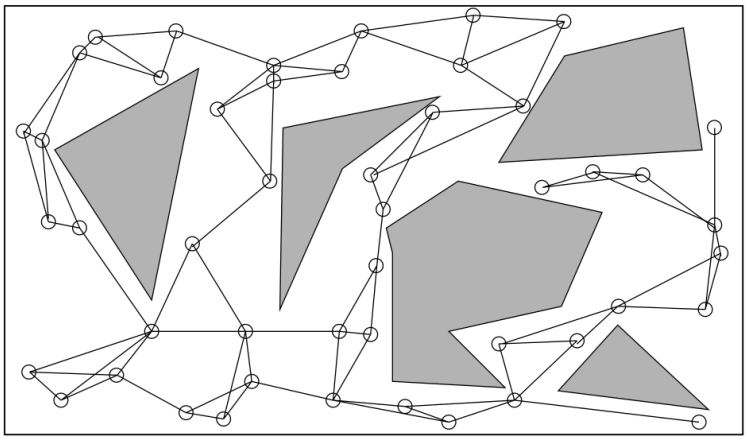

Figure 2.4: Roadmap construction probabilistic roadmap algorithm taken from [29]

Under the definitions below, Algorithm 1 defines roadmap construction step of probabilistic roadmap algorithm.

- Let Δ be a symmetric and deterministic local planner which puts an edge between $q^{'}$ and $q^{''}$ if the edge is collision free, or not.
- Let $dist$ is a distance function with a specific metric which outputs the distance between $q^{'}$ and $q^{''}$.

Algorithm 1 Roadmap Construction Algorithm [29]

procedure genRoadmap (n, k)
  ▷ n: number of nodes to put in the roadmap
  ▷ k: number closest neighbors to examine for each configuration
  $V \leftarrow \varnothing$, $E \leftarrow \varnothing$
  while $|V| < n$ **do**
    repeat
      $q \leftarrow$ a random configuration in Q
    until q is collision-free
    $V \leftarrow V \cup \{q\}$
  end while
  for all $q \in V$ **do**
    $N_q \leftarrow$ the k closest neighbors of q chosen from V according to $dist$
    for all $q' \in N_q$ **do**
      if $(q, q') \notin E$ **and** $\Delta(q, q') \neq NIL$
        $E \leftarrow E \cup \{(q, q')\}$
      end if
    end for
  end for
  return $G = (V, E)$

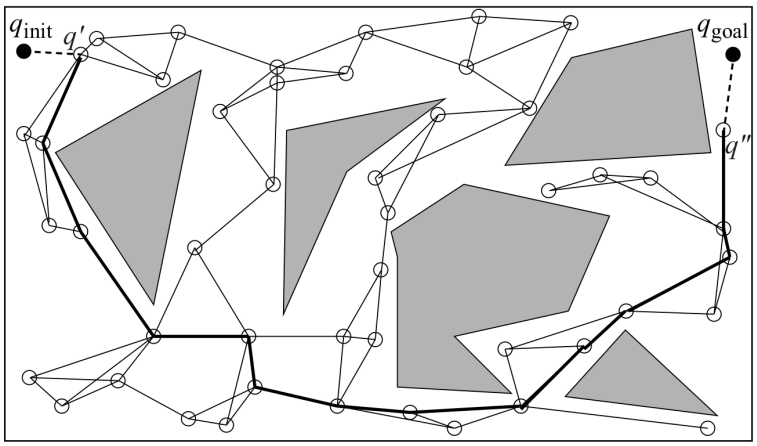

Figure 2.5: Query step of probabilistic roadmap algorithm taken from [29]

After construction the roadmap, shortest path is found between q_{start} and q_{goal} in

query step of probabilistic roadmap algorithm which is represented in Algorithm 2.

Algorithm 2 Solve Query Algorithm [29]

1: **procedure** query $(q_{start}, q_{goal}, k$,G)
2: ▷ q_{start}: the initial state
3: ▷ q_{goal}: the goal state
4: ▷ k: number closest neighbors to examine for each configuration
5: ▷ $G = (V, E)$: the graph computed by Algorithm 1
6: $N_{q_{start}} \leftarrow$ the k closest neighbors of q_{start} from V according to $dist$
7: $N_{q_{goal}} \leftarrow$ the k closest neighbors of q_{goal} from V according to $dist$
8: $V \leftarrow \{q_{start}\} \cup \{q_{goal}\} \cup V$
9: set $q^{'}$ to be the closest neighbor of q_{start} in $N_{q_{start}}$
10: **repeat**
11: **if** $\Delta(q_{start}, q^{'}) \neq NIL$, **then**
12: $E \leftarrow (q_{start}, q^{'}) \cup E$
13: **else**
14: set $q^{'}$ to be the closest neighbor of q_{start} in $N_{q_{start}}$
15: **end if**
16: **until** a connection was successful or the set $N_{q_{start}}$ is empty
17: set $q^{'}$ to be the closest neighbor of q_{goal} in $N_{q_{goal}}$
18: **repeat**
19: **if** $\Delta(q_{goal}, q^{'}) \neq NIL$, **then**
20: $E \leftarrow (q_{goal}, q^{'}) \cup E$
21: **else**
22: set $q^{'}$ to be the closest neighbor of q_{goal} in $N_{q_{goal}}$
23: **end if**
24: **until** a connection was successful or the set $N_{q_{goal}}$ is empty
25: $P \leftarrow$ shortest path(q_{start}, q_{goal}, G)
26: **if** P is not empty **then**
27: **return** P
28: **else**
29: **return** failure
30: **end if**

In some cases, the graph G may not be symmetric if Δ is neither symmetric nor deterministic. When constructing the roadmap, both of the directions should be taken into account. Since shortest path algorithms can take adjacency matrices as input, generating an adjacency matrix that represents the graph can be helpful.

2.3.2 Rapidly-Exploring Random Trees

Rapidly-exploring random trees is another sampling-based motion planning algorithm. Additionally, it is a single-query planning algorithm which is the main difference from probabilistic roadmap algorithm. Therefore, it is generally more efficient than probabilistic roadmap algorithm because single-query based extension can cover Q_{free} rapidly [29].

There can be multiple trees in rapidly-exploring random trees algorithm. This is very useful in terms of parallelism, and each tree can be allocated a core in multi-core environments. In general, one of the trees is rooted at q_{start}, and one of them is rooted at q_{goal}. Each tree of rapidly-exploring random trees algorithm is extended incrementally in each iteration. A state q_{rand} is generated randomly in Q_{free} for each tree in each iteration. The nearest state q_{near} inside tree T is derived and a new state q_{new} is generated between q_{near} and q_{rand} with step size distance to q_{near}. If q_{new} is collision-free, it is added to T as a new vertex with the edge $E' = (q_{near}, q_{new})$ [29].

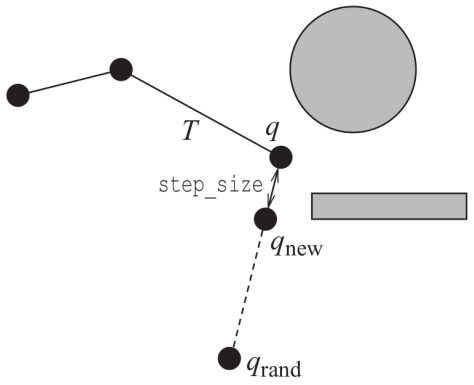

Figure 2.6: Construction step of rapidly-exploring random trees algorithm taken from [29]

Algorithms 3 and 4 represent tree construction of rapidly-exploring random trees algorithm.

Algorithm 3 Build RRT Algorithm [29]

procedure buildRRT (q_0, n)
▷ q_0: the configuration where the tree is rooted
▷ n: the number of attempts to expand the tree
$V \leftarrow \{q_0\}$
$E \leftarrow \varnothing$
for $i = 1$ **to** n **do**
  $q_{rand} \leftarrow$ a randomly chosen free configuration
  $extendRRT(T, q_{rand}, step\_size)$
end for
return T

Algorithm 4 Extend RRT Algorithm [29]

procedure extendRRT $(T, q, step\_size)$
▷ $T = (V, E)$: a rapidly-exploring random tree
▷ q: a state toward which the tree T is grown
▷ $step\_size$: the step size for extending tree through randomly generated vertex
$q_{near} \leftarrow$ closest neighbor of q in T
$q_{new} \leftarrow$ progress q_{near} by $step\_size$ along the straight line in Q between q_{near} and q
if q_{new} is collision-free **then**
  $V \leftarrow V \cup \{q_{new}\}$
  $E \leftarrow E \cup \{(q_{near}, q_{new})\}$
  return q_{new}
end if
return NIL

Step size and sampling method can be varied according to the application of rapidly-exploring random trees algorithm. Selection of nearest collision-free q_{new} to q_{rand} can be considered. Moreover, greedier sampling of q_rand can be an option. In [30], a discussion about all the metrics of rapidly-exploring random trees can be found. In

the merging step, rapidly-exploring random trees are tried to be connected. Each tree is extended through randomly generated state q_{rand}. If they meet together at q_{rand}, the algorithm halts [29].

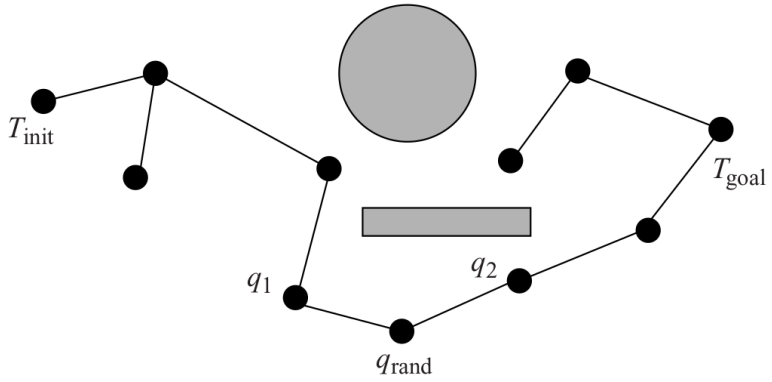

Figure 2.7: Merging step of rapidly-exploring random trees algorithm taken from [29]

2.3.3 Voronoi Diagrams

A set of points which are equidistant to sites [31] form Voronoi diagram. Since all the points in a Voronoi diagram are collision-free, and those points constitute a graph which basically covers Q_{free}, Voronoi diagrams in ^{m} have accessibility, connectivity, and departability [29].

In real-world, obstacles are not just points. Therefore, Voronoi region is defined by

$$\mathcal{F}_i = \{q \in Q_{free} \mid d_i(q) \leq d_h(q),\ \forall h \neq i\} \tag{2.1}$$

for set of points closest to $\mathcal{QO}_i$. By restricting the set for two closest obstacle, $\mathcal{QO}_i$ and $\mathcal{QO}_j$, the subset of points which are equidistant to closest two obstacles can be defined as

$$\mathcal{F}_{ij} = \{q \in S_{ij} \mid d_i(q) \leq d_h(q),\ \forall h\}^2 \tag{2.2}$$

where $d_i(q)$ is the distance between obstacle $\mathcal{QO}_i$ and q. Therefore, whole Voronoi

diagram is represented by

$$\mathcal{F} = \bigcup_i \bigcup_j \mathcal{F}_{ij}. \tag{2.3}$$

Figure 2.8 is an example of construction of a Voronoi diagram. In (a) and (b), the line $\mathcal{S}_{ij}$ is equidistant from obstacles $\mathcal{QO}_i$ and $\mathcal{QO}_j$. In (c), the segment of $\mathcal{S}_{ij}$ ($\mathcal{F}_{ij}$)which is closest to pair of $\mathcal{QO}_i$ and $\mathcal{QO}_j$ is represented. In (d), union of all of the sets ($\mathcal{F}$) represented [29].

A Voronoi diagram-based motion planner firstly constructs Voronoi diagram based on the obstacles in configuration space. The closest collision-free path from Voronoi diagram to the start state q_{start} is found and inserted into the diagram. Similarly, the closest collision-free path from Voronoi diagram to the goal state q_{goal} is found and inserted into the diagram. Therefore, the resultant graph forms a roadmap which includes the start state q_{start} and the goal state q_{goal} [29].

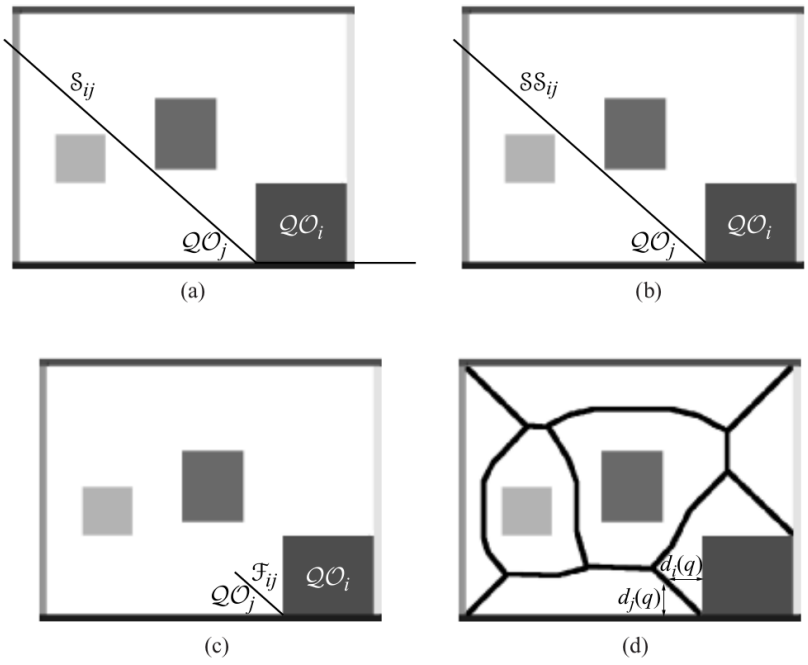

Figure 2.8: Construction of a Voronoi diagram taken from [29]

Voronoi diagrams can be generated by agents which have 2D or 3D ranging sensors or by using geometric features of the environment. If obstacles are of the form of

polygons or circular shapes, equidistant paths can be generated easily between those. Figure 2.9 is a good example of generating Voronoi diagram using geometric features of obstacles. The set of equidistant points of two line segments forms a line segment. Different from this, the set of equidistant points between a point and a line segment forms a parabola [29].

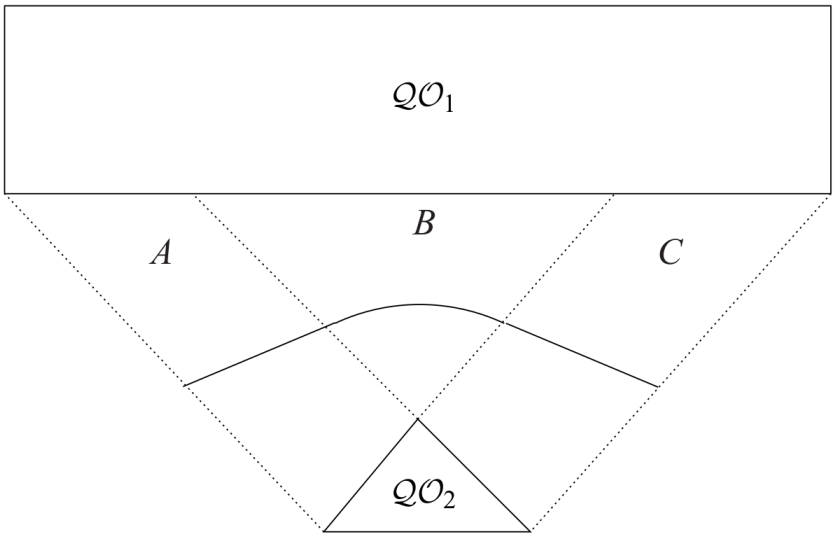

Figure 2.9: Construction of a Voronoi diagram using geometric features taken from [29]

2.4 Related Work

Wilburn et al. [32] proposed a modified Voronoi diagram-based path planning algorithm for unmanned aerial vehicle (UAV) path planning. Traditional path planning with Voronoi diagram-based path planning algorithm is explained, and a new approach is proposed with cylindrical obstacles and threat zones. By using centers of obstacles and threat zones as sites reduces execution time since the number of sites is the input length of Voronoi diagram algorithm. However, there is not any comparison with the traditional path planning algorithm based on Voronoi diagram. There are weaknesses of the proposed algorithm: ragged edges and intersecting obstacles. In other words, edges may be intersecting with cylindrical obstacles in some cases, and the intersection of obstacles can result in an intersecting path between those ob-

stacles. Furthermore, representing obstacles and threats as cylinders is not a realistic approach. The aerodynamic model of YF-22 is used in FlightGear simulation environment in the related study. Vertices of Voronoi diagram should be evicted from obstacles. For this reason, sites are placed on the points of an obstacle that are closest to the nearest obstacle. However, threat zones remain with one site at the center of the zone in this study. This can cause a vulnerability in terms of safety.

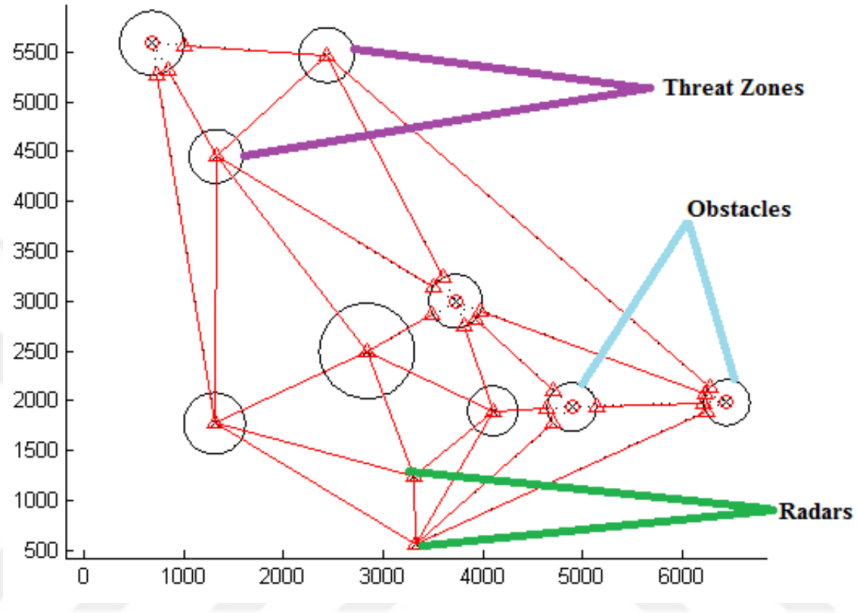

Figure 2.10: Construction of a Voronoi diagram in [32]

Candeloro et al. [33] proposed a 3D Voronoi diagram-based path planning for unmanned underwater vehicles which are working in a sub-sea factory. Dubins path is used for flight path segment generation in this study. Furthermore, the proposed method takes care of moving objects inside configuration space. In the case of moving objects, the Voronoi diagram is reconstructed considering them. Safe navigation rules based on maneuvers of objects are proposed. In other words, the dynamic obstacle avoidance method in the study is a rule-based approach. In the study, a trajectory optimization method that is based on waypoint elimination is proposed as well.

Candeloro et al. [34] proposed a Voronoi diagram-based dynamic path planning system for 3-DOF marine surface vessels. In this study, Fermat Spiral is used for path segment generation. The depth of the sea is taken into consideration as well during

the construction of Voronoi diagram. For other moving marine surface vessels, projected obstacle areas are generated, and the path is re-planned according to projected obstacle areas.

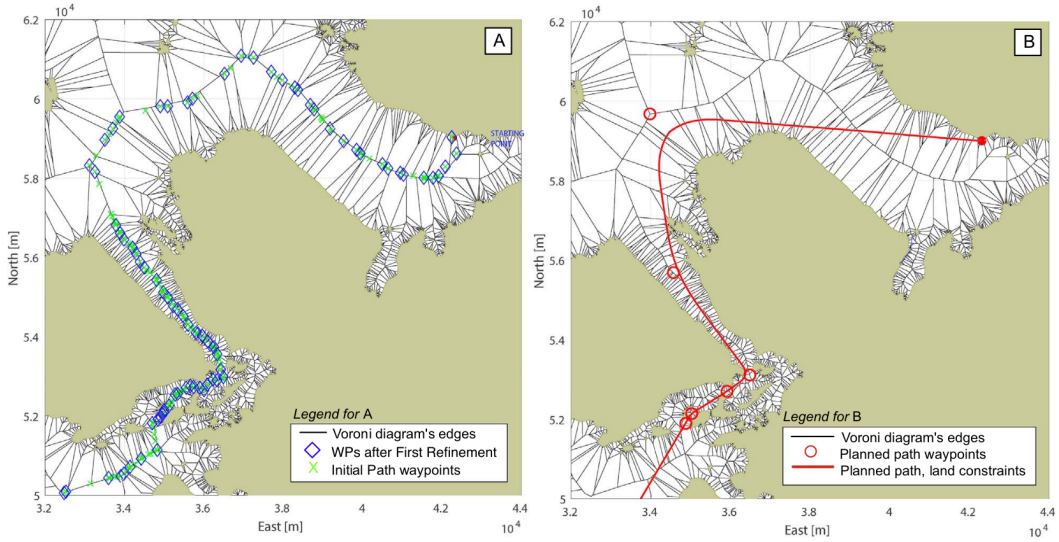

Figure 2.11: Construction of a Voronoi diagram in [34]

Hvamb, K. [35] analyzed Voronoi diagrams, rapidly-exploring random trees and probabilistic roadmap algorithms for marine vehicles in his master's thesis. A comparison is represented for small and large areas in the related thesis. Voronoi diagram-based path planning is decided as very effective in large areas. However, in complex and small areas, sampling-based algorithms are decided as more successful.

Shanmugavel et al. [36] proposed a cooperative path planning strategy for a group of unmanned aerial vehicles. Firstly, flyable paths are generated with Dubins path with clothoid arcs. Secondly, the generated paths are processed in order to avoid intersections by maintaining a minimum separation distance. Finally, all the generated paths are made of equal length in order to reach the goal simultaneously.

Yu et al. [37] proposed a real-time trajectory planning for unmanned aerial vehicles using inverse dynamics optimization method and receding horizontal control, which is efficient in terms of execution time and has good convergence rates. In the related study, an elliptic launch acceptability region modeling approach is proposed as well. However, all the obstacles or threats are represented by spheres, rectangular prisms,

and cylinders. In the presented scenario with stationary obstacles, the average planning time is $1.781s$ with Intel 2.40 GHz quad-core processor, using the Windows XP operating system.

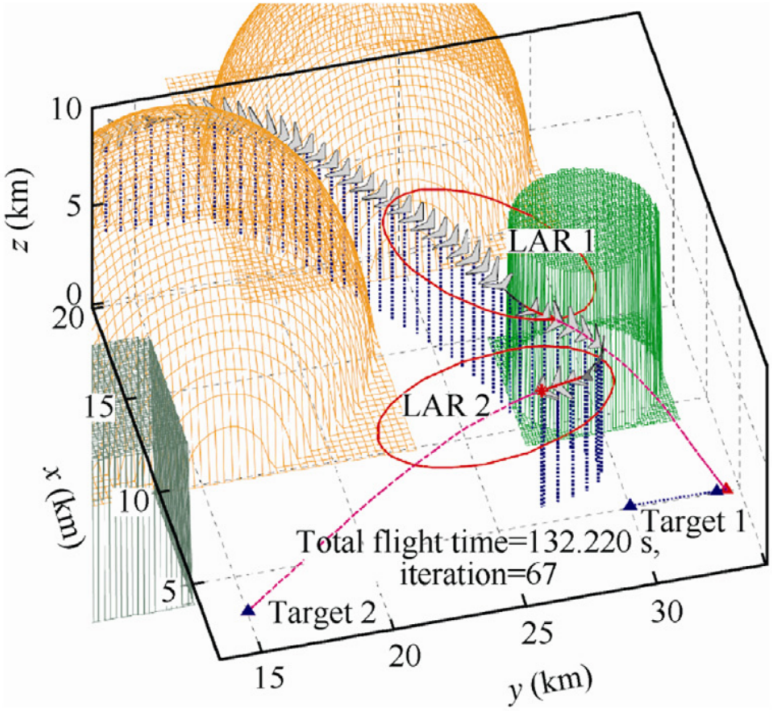

Figure 2.12: Derived air-to-surface attack trajectory in [37]

CHAPTER 3

PROBABILISTIC ROADMAP-BASED METHOD

This chapter summarizes the proposed probabilistic roadmap-based method and which is already published in [27].

3.1 Method

3.1.1 Modeling Launch Acceptability Region

Smart air-to-surface weapons calculate launch acceptability region by using navigation and environmental data transmitted by aircraft. When calculating launch acceptability region, smart weapons generally execute flight models in order to predict boundaries of six degrees of freedom motion in given environmental constraints. Smart air-to-surface weapons generally do not have propulsion systems. Nevertheless, they have guidance capability by using their ailerons. Two forces are affecting weapon's motion after release: drag force and gravitational force. Aileron movements manipulate drag force's magnitude and direction. Air resistance with cross-section area together causes drag force. When modeling launch acceptability region calculation, it is assumed that minimum and maximum cross-section area in the face of air resistance state boundaries of launch acceptability region. For this reason, gravitational and drag forces are calculated according to minimum and maximum cross-section area separately. Drag force (F_d) calculation is done by formula

$$F_d = \frac{1}{2}\rho A v^2 C_d \tag{3.1}$$

where ρ indicates air density, A indicates cross-section area of weapon, v indicates vertical velocity, and C_d indicates drag coefficient. Projectile motion is calculated incrementally because analytical solutions are computationally complex.

D_x and D_y represent air resistance coefficients for both axes and are calculated by

$$D_y = \frac{1}{2}\rho A_y C_{dy} \tag{3.2}$$

and

$$D_x = \frac{1}{2}\rho A_x C_{dx}. \tag{3.3}$$

According to Newton's second law of motion ($F = ma$),

$$ma_y = F_g + F_{dy} \tag{3.4}$$

and,

$$ma_x = F_{dx} \tag{3.5}$$

where a_x and a_y indicate accelerations along both axes, F_g is the resultant gravitational force, F_{dx} and F_{dy} indicates drag forces. By reducing the equations, we have

$$a_y = g + (\frac{D_y}{m})|v_i||v_y| \tag{3.6}$$

and

$$a_x = -(\frac{D_x}{m})|v_i||v_x| \tag{3.7}$$

where g is gravitational acceleration, v_i is resultant velocity, v_x and v_y are projections of velocity on axes.

Applying analytical method makes position and velocity calculation difficult since it's very complex. Current position and velocity are calculated from previous instances with a proper timestamp Δt by

$$v_x = v_x + a_x * \Delta t, \tag{3.8}$$

$$v_y = v_y + a_y * \Delta t, \tag{3.9}$$

$$x_{t+1} = x_t + v_x * \Delta t + \frac{1}{2}a_x * \Delta t^2, \tag{3.10}$$

and

$$y_{t+1} = y_t + v_y * \Delta t + \frac{1}{2} a_y * \Delta t^2. \tag{3.11}$$

v_i is resultant velocity of velocities v_x and v_y, and should be updated in every iteration in terms of them:

$$v_i = \sqrt{(v_x)^2 + (v_y)^2}. \tag{3.12}$$

When weapon hits the ground, the value of y_t equals to terrain elevation and x_t indicates the trail.

Equations from 3.1 to 3.12 belong to [38]. In this calculation, it is assumed that the trail can be maximized by minimizing cross-section area towards trail direction (A_x) and drag coefficient towards trail direction (C_{dx}) and maximizing cross-section area towards the ground direction (A_y) and drag coefficient towards the ground direction. (C_{dy}). On the contrary, it is assumed that the trail can be minimized by maximizing cross-section area towards trail direction (A_x) and drag coefficient towards trail direction (C_{dx}) and minimizing cross-section area towards the ground direction (A_y) and drag coefficient towards the ground direction (C_{dy}).

To conclude, minimum and maximum trails with a given digital elevation map can be calculated with numerical method. If the distance between the release point and target is not between the calculated minimum and maximum trails, that state is counted as unsuccessful. Vice versa, if the distance between the release point and target is between the minimum and maximum trail, that state is counted as successful. The approach used to modeling launch acceptability region is represented in Figure 3.1.

For both of the proposed methods, ROSplane simulation environment is used, which simulates the motion of an unmanned aerial vehicle. There are many air-launched air-to-surface weapons designed to be used on unmanned aerial vehicles. Roketsan's MAM-C is only one of them, and it is designed for low payload capacity air platforms. Its publicly available technical specifications are used in realizing simulations of modeling launch acceptability region. Table 3.1 consists of publicly available information about MAM-C, and some of its specifications are just roughly assumed [39].

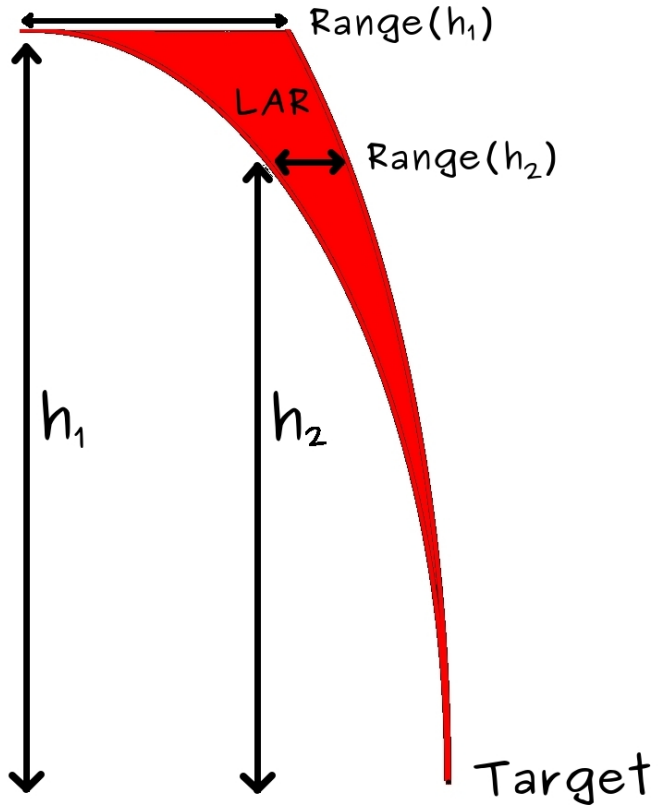

Figure 3.1: LAR calculation with perpendicular heading

Table 3.1: Roketsan MAM-C Specifications

Diameter	2.75” $70mm$
Length	$970mm$
Weight	$6.5kg$
Minimum Drag Coefficient *	0.295
Maximum Drag Coefficient *	0.82
Minimum Cross Section Area (A_{min})$*$	$0,0038m^2$
Maximum Cross Section Area (A_{max})$*$	$0.07m^2$

$*$Assumption

In Figure 3.2, minimum and maximum drag forces are represented. When the cross-section area and the drag coefficient are maximum (left example), the trail will be

minimum. In that case, the cross-section area is roughly calculated by the cylindrical lateral face's cross-section, and the drag coefficient is taken as 0.82 according to [40]. In order to reach the maximum trail, it is assumed that motion is in the direction of the spherical head of the weapon. In this case (right example), the cross-section area is calculated from the circular frontal cross-section of the weapon. Again, the drag coefficient is taken as bullet's, 0.295 [41].

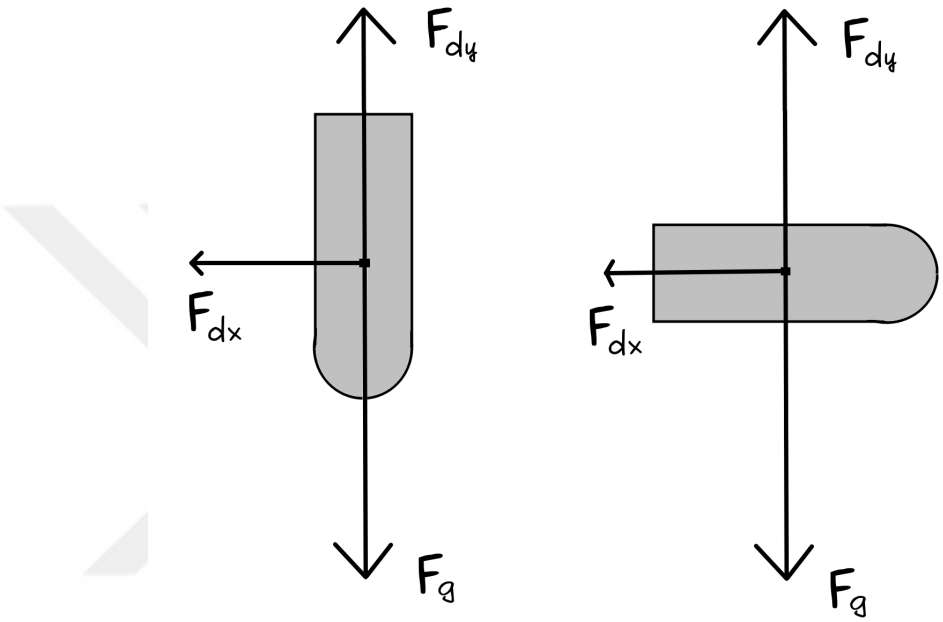

Figure 3.2: Minimum and maximum drag forces

Weapon parameters are listed in Table 3.2 in order to reach minimum and maximum trail.

Table 3.2: Parameters of Numeric LAR Query Method

Parameter	Max Trail	Min Trail
A_x	$0.0038\ m^2$	$0.07\ m^2$
A_y	$0.07\ m^2$	$0.0038\ m^2$
C_{dx}	0.295	0.82
C_{dy}	0.82	0.295

3.1.2 Flight Path Segment Generation with Dubins Path

ROSPlane [42] is a Robot Operating System (ROS) [43] package which provides an autopilot simulation environment based on the dynamics of an unmanned aerial vehicle and Gazebo simulation engine. ROSplane autopilot is composed of path manager, path follower, controller, and state estimator ROS nodes. Path planner node generates waypoints and transmits to path manager node. The output of planner node is replaced by waypoints generated via the proposed method. Path manager node produces flight path segments using waypoints transmitted by path planner. Flight path segments can be produced by two methods: Dubins and fillet path. In the probabilistic roadmap-based method, Dubins path is preferred because waypoints can be very close to threats, and headings of aircraft states should be taken into account. Sensor models and flight dynamics are modeled using Gazebo. Therefore, state estimator estimates aircraft states using Gazebo-simulated sensors and flight dynamics. Then, actuator control outputs are transmitted to Gazebo simulation [44].

In this study, North-East-Down (NED) frame is used for representing free space. Waypoints are represented as aircraft states $< N, E, D, \psi >$ and switching between states is done by Dubins path generation. According to half planes approach, there must be starting state and final state in a flight path segment. In a Dubins path, there are one arc followed by a straight line and a final arc. The arcs can be formed by turning left or right with a specified turning radius R greater than the minimum turning radius of aircraft R_{min}. Let's say starting state is $< p_s, \chi_s >$ where p_s is the starting position and χ_s is starting heading of aircraft. Similarly, end state is $< p_e, \chi_e >$ where p_e is the end position and χ_e is end heading of aircraft. In order to reach from $< p_s, \chi_s >$ to $< p_e, \chi_e >$, aircraft should follow the generated Dubins path: first an arc with a turning direction (left or right) and a turning radius R_1, second, a straight line with a length, and finally another arc with a turning direction (left or right) and a turning radius R_2. R_1 and R_2 should be greater than minimum turning radius of aircraft R_{min}. Furthermore, flight path angle γ of aircraft during flight path segment should be less than maximum flight path angle γ_{max}. Dubins path generation algorithm can generate 4 different possible flight path segments: LSL, LSR, RSL and RSR where L indicates left, R indicates right and S indicates straight [45].

Two circles tangent to aircraft state vector are placed both start $< p_s, \chi_s >$ and end $< p_e, \chi_e >$ states and Dubins paths are calculated for 4 cases separately. In the approach used in this work, R_1 and R_2 are taken as R_{min}, and the shortest path among 4 cases is preferred as flight path segment to be followed between start and end states [21].

θ indicates the angle between centers of selected start and end turning circles. Where l indicates left, r indicates right and s indicates start state and e indicates end state;

- c_{ls} stands for the center of left turning circle of start state
- c_{rs} stands for the center of right turning circle of start state
- c_{le} stands for the center of left turning circle of end state
- c_{re} stands for the center of right turning circle of end state.

w_s, w_l and w_e are switching points. First, aircraft switches straight motion from circular motion in w_s. Second, aircraft switches circular motion from straight motion in w_l. Finally, w_e is last switching point where aircraft switches to another flight path segment generated by Dubins path generation. q_s and q_l are unit vectors where their directions from w_s to w_l, q_e indicates the direction of end state [46]. Dubins algorithm approach used in this work is represented in Figure 3.3 and Algorithm 5.

Lengths of 4 possible flight path segments are calculated by equations 3.13, 3.14, 3.15 and 3.16. Least of all is selected as flight path segment to be followed.

$$L_1 \longleftarrow \|c_{rs} - c_{re}\| + R\langle 2\pi + \langle \theta - \frac{\pi}{2} \rangle - \langle \chi_s - \frac{\pi}{2} \rangle \rangle + R\langle 2\pi + \langle \chi_e - \frac{\pi}{2} \rangle - \langle \theta - \frac{\pi}{2} \rangle \rangle \tag{3.13}$$

$$L_2 \longleftarrow \sqrt[2]{\|c_{le} - c_{rs}\|^2 - 4R^2} + R\langle 2\pi + \langle \theta_2 \rangle - \langle \chi_s - \frac{\pi}{2} \rangle \rangle + R\langle 2\pi + \langle \theta_2 + \pi \rangle - \langle \chi_e + \frac{\pi}{2} \rangle \rangle \tag{3.14}$$

$$L_3 \longleftarrow \sqrt[2]{\|c_{re} - c_{ls}\|^2 - 4R^2} + R\langle 2\pi - \langle \theta + \theta_2 \rangle + \langle \chi_s + \frac{\pi}{2} \rangle \rangle + R\langle 2\pi - \langle \theta + \theta_2 - \pi \rangle + \langle \chi_e - \frac{\pi}{2} \rangle \rangle \tag{3.15}$$

$$L_4 \longleftarrow \|c_{le} - c_{ls}\| + R\langle 2\pi - \langle \theta + \frac{\pi}{2} \rangle + \langle \chi_s + \frac{\pi}{2} \rangle \rangle \; + \\ R\langle 2\pi - \langle \chi_e + \frac{\pi}{2} \rangle + \langle \theta + \frac{\pi}{2} \rangle \rangle \tag{3.16}$$

In Algorithm 5, waypoint transition logic approach defined in [46] is used and all Dubins parameters are derived [47].

Path elevation difference is defined as the altitude difference between the start and end states of a flight path segment when it is assumed that the first derivative of altitude change is always negative or positive in a flight path segment. Dividing altitude difference to the length of the projection of Dubins path on x-y plane provides tangent of flight path angle of related flight path segment γ. If γ is greater than the maximum flight path angle γ_{max} of aircraft, end state cannot be reached. Additionally, R_z represents rotation matrix around z-axis for a given angle on x-y plane.

Algorithm 5 Flight Path Segment Generation using Dubins Approach: DubinsPath($w_s, w_e, R_{min}, \gamma_{max}$)

Data: Path start point $w_s = (p_s, \chi_s)$, where $p_s = (p_{s_x}, p_{s_y}, p_{s_z})$ path end point $w_e = (p_e, \chi_e)$, where $p_e = (p_{e_x}, p_{e_y}, p_{e_z})$, turn radius R_{min}, and maximum flight path angle γ_{max}

1: $c_{rs} \leftarrow p_s + R_{min}\mathbf{R_z}(\pi/2)[\cos\chi_s, \sin\chi_s, 0]^T$
2: $c_{ls} \leftarrow p_s + R_{min}\mathbf{R_z}(-\pi/2)[\cos\chi_s, \sin\chi_s, 0]^T$
3: $c_{re} \leftarrow p_e + R_{min}\mathbf{R_z}(\pi/2)[\cos\chi_e, \sin\chi_e, 0]^T$
4: $c_{le} \leftarrow p_e + R_{min}\mathbf{R_z}(-\pi/2)[\cos\chi_e, \sin\chi_e, 0]^T$
5: $L_1 \leftarrow CalculateRSRDistance\{c_{rs}, c_{re}, R_{min}\}$
6: $L_2 \leftarrow CalculateRSLDistance\{c_{rs}, c_{le}, R_{min}\}$
7: $L_3 \leftarrow CalculateLSRDistance\{c_{ls}, c_{re}, R_{min}\}$
8: $L_4 \leftarrow CalculateLSLDistance\{c_{ls}, c_{le}, R_{min}\}$
9: $[L, i] \leftarrow min\{L1, L2, L3, L4\}$
10: $d_{elev} \leftarrow -(c_{e_z} - c_{s_z})$
11: **if** $|\mathrm{d}_{elev}| \leq \mathrm{L}\tan(\gamma_{max})$
12: $e_1 \leftarrow [1, 0, 0]^T$

if $i == 1$ ▷ RSR

$c_s \leftarrow c_{rs}, \delta_s \leftarrow CW, c_e \leftarrow c_{re}, \delta_e \leftarrow CW$

$\theta \leftarrow \arctan 2(c_{e_y} - c_{s_y}, c_{e_x} - c_{s_x})$

$q_1 \leftarrow \dfrac{c_e - c_s}{\|c_e - c_s\|}$

$w_s \leftarrow c_{rs} + \mathbf{R_z}(-\pi/2)q_1 R_{min}$

$w_l \leftarrow c_{re} + \mathbf{R_z}(-\pi/2)q_1 R_{min}$

else if $i == 2$ ▷ RSL

$c_s \leftarrow c_{rs}, \delta_s \leftarrow CW, c_e \leftarrow c_{le}, \delta_e \leftarrow CCW$

$l \leftarrow \|c_e - c_s\|$

$\theta \leftarrow \arctan 2(c_{e_y} - c_{s_y}, c_{e_x} - c_{s_x})$

$\theta_2 \leftarrow \theta - \pi/2 + \arcsin(2R_{min}/l)$

$q_1 \leftarrow \mathbf{R_z}(\theta_2 + \pi/2)e_1$

$w_s \leftarrow c_{rs} + \mathbf{R_z}(\theta_2)q_1 R_{min}$

$w_l \leftarrow c_{re} + \mathbf{R_z}(\theta_2 + \pi)q_1 R_{min}$

else if $i == 3$ ▷ LSR

$c_s \leftarrow c_{ls}, \delta_s \leftarrow CCW, c_e \leftarrow c_{re}, \delta_e \leftarrow CW$

$l \leftarrow \|c_e - c_s\|$

$\theta \leftarrow \arctan 2(c_{e_y} - c_{s_y}, c_{e_x} - c_{s_x})$

$\theta_2 \leftarrow \arccos(2R_{min}/l)$

$q_1 \leftarrow \mathbf{R_z}(\theta + \theta_2 - \pi/2)e_1$

$w_s \leftarrow c_{ls} + \mathbf{R_z}(\theta + \theta_2)e_1 R_{min}$

$w_l \leftarrow c_{re} + \mathbf{R_z}(\theta + \theta_2 - \pi)e_1 R_{min}$

else if $i == 4$ ▷ LSL

$c_s \leftarrow c_{ls}, \delta_s \leftarrow CCW, c_e \leftarrow c_{le}, \delta_e \leftarrow CCW$

$\theta \leftarrow \arctan 2(c_{e_y} - c_{s_y}, c_{e_x} - c_{s_x})$

$q_1 \leftarrow \dfrac{c_e - c_s}{\|c_e - c_s\|}$

$w_s \leftarrow c_{ls} + \mathbf{R_z}(\pi/2)q_1 R_{min}$

$w_l \leftarrow c_{le} + \mathbf{R_z}(\pi/2)q_1 R_{min}$

end if

$w_e \leftarrow p_e, q_s \leftarrow q_1, q_l \leftarrow q_1, q_e \leftarrow \mathbf{R_z}(\chi_e)e_1$

end if

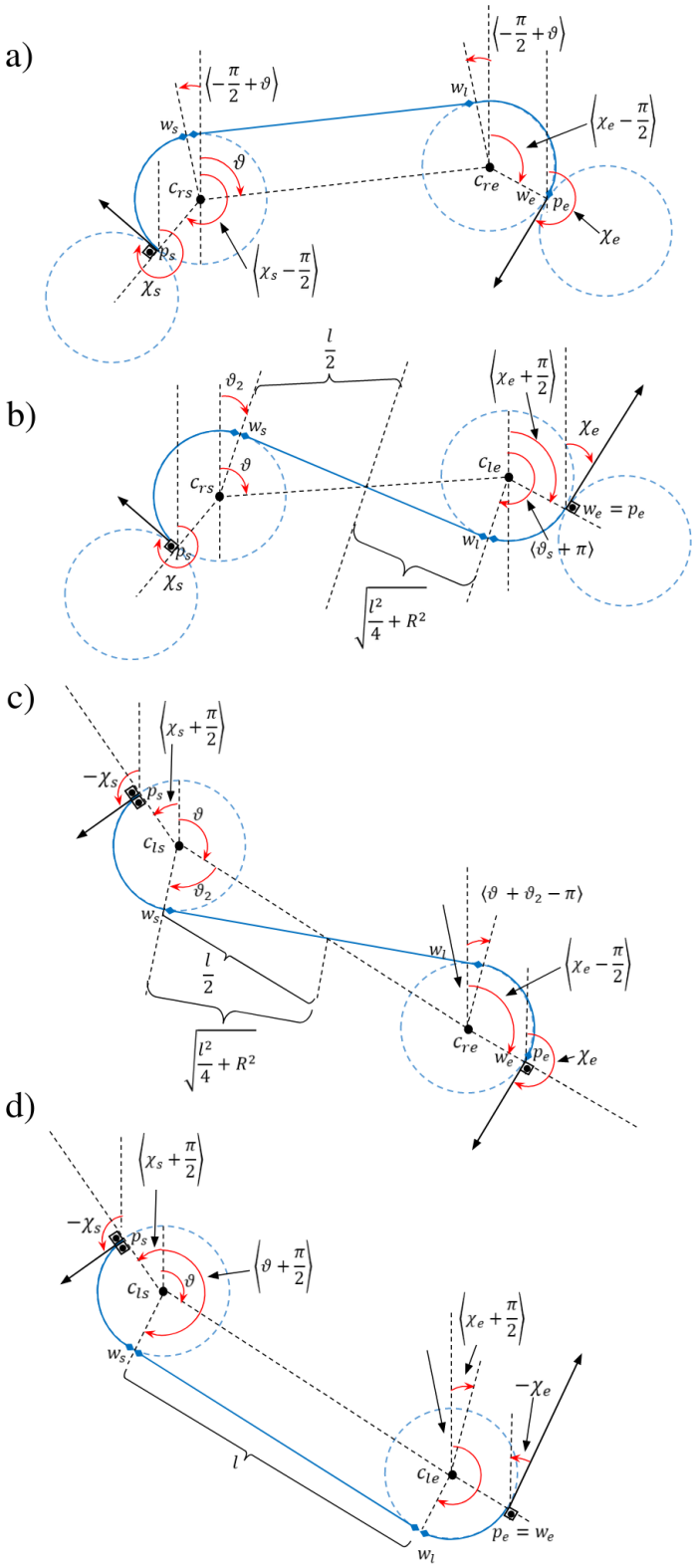

Figure 3.3: Dubins path generation configurations: a) RSR, b) RSL, c) LSR, d) LSL

3.1.2.1 Determining Minimum Turning Radius and Maximum Flight Path Angle

A bunch of experiments is performed for the purpose of finding out the flight parameters of ROSplane platform. A circular motion is commanded with different radius values and constant velocity $12\ m/s$ in order to discover the minimum turning radius value R_{min} of the platform with the given velocity. Furthermore, goal waypoints with various elevation differences are commanded to discover maximum flight path angle γ_{max} which aircraft can reach with the given velocity.

At the end of experiments, minimum turning radius R_{min} is obtained as $30\ m$ with constant linear velocity $12\ m/s$. Additionally, maximum flight path angle γ_{max} is obtained as $0.15\ rad$ with the same constant linear velocity value. In Figure 3.4, some examples of aircraft's motion are illustrated when different turning radii are commanded. On the left, $R = 25\ m$ is commanded; on the right, $R = 30\ m$ is commanded. It can be easily seen that aircraft deviates from its orbital path when $R = 25\ m$.

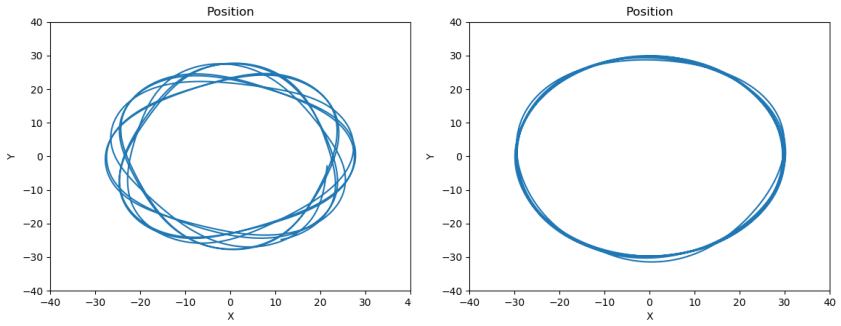

Figure 3.4: Experiment results for $R = 25m$ and $R = 30m$

3.1.3 Path Planning

Probabilistic roadmap path planning algorithm generates a sampling-based roadmap and searches the shortest path with artificial intelligence search algorithms. A custom probabilistic roadmap algorithm [48] with Dubins path distance and Dijkstra's shortest path algorithm is developed in order to reach desired release point found out

launch acceptability region queries.

Air defense systems are represented by ellipsoidal volumes, which can utilize different semi-axis lengths. Querying whether a state is inside of a threat region or not is done by inequality

$$\frac{(x-x_c)^2}{a^2}+\frac{(y-y_c)^2}{b^2}+\frac{(z-z_c)^2}{c^2}\leq 1 \tag{3.17}$$

where $< x_c, y_c, z_c >$ indicates coordinates of center of ellipsoidal volume which represent an air defense system, where a, b and c represent semi axis lengths of ellipsoidal volume.

Algorithm 6 represents the main structure of the customized probabilistic roadmap path planning algorithm developed in this work. gen_wp requires start state of aircraft ($start$), boundaries of the battlefield where uniformly distributed aircraft states are sampled to generate a roadmap ($bounds$), the number of samples to be generated inside of boundaries (num), coordinates of threats and semi-axis lengths of them ($threats$), coordinates of the target in terms of NED ($tgtCoords$), maximum Dubins path distance between samples permitted to place edge when producing roadmap by using those samples ($threshold$), minimum turning radius (R_{min}) and maximum flight path angle that aircraft can reach (γ_{max}). First of all, $genSamples$ generates a number of aircraft state samples with respect to boundaries of the battlefield and eliminates samples that are inside of threat volumes according to inequality 3.17. $genSamples$ is detailed in Algorithm 7. Then, $start$ state is inserted into the generated set of $samples$. Then, $genGoalSamples$ generates a specified number of possible goal states, queries whether they are inside launch acceptability region or not, and forms a set of successful goal states. Then, possible goal states are inserted to $samples$. $genGraph$ function forms a directed roadmap by outputting an adjacency matrix according to Dubins path distances between samples and $threshold$ value. Then, Dijkstra's shortest path algorithm is applied to the directed adjacency matrix in order to obtain distances of each sample to start state and form a predecessor tree. Next, the closest goal to start state is searched, and waypoints of resultant trajectory are fetched from the predecessor tree.

Algorithm 6 Generation of waypoints (PRM)

1: **procedure** gen_wp $(start, bounds, num, threats, tgtCoords, threshold, R_{min}, \gamma_{max})$
2: $samples \leftarrow$ **genSamples**$(bounds, num, threats)$
3: $samples \leftarrow \{start\} \cup samples$
4: $samples \leftarrow samples \cup$ **genGoalSamples**$(tgtCoords)$
5: $graph \leftarrow$ **genGraph**$(samples, threshold, R_{min}, \gamma_{max})$
6: ▷ First index is starting waypoint
7: $dists, predec \leftarrow$ **dijkstra**$(graph, 1)$
8: $goal \leftarrow$ **findClosestGoal**$(dists, goals)$
9: ▷ Extracting trajectory from predecessor tree
10: $waypoints \leftarrow$ **fetchWaypoints**$(1, goal, predec)$
11: **return** $waypoints$ ▷ Desired trajectory

$genGoalSamples$ function generates normally distributed aircraft state samples inside a cylindrical volume that centers target coordinates. According to the approach presented in section 3.1.1, launch acceptability regions are queried for generated samples. Then, samples that are not in launch acceptability region and inside of threat volumes are eliminated. A set is formed by remaining samples and inserted into $samples$ set. Since High Speed 1760 provides high-speed data transmission between aircraft and weapons, a huge number of possible goal states can be generated with this approach. This approach is computationally expensive since it inserts a great number of possible goal states to the roadmap, but it is effective in terms of finding the closest release point.

Algorithm 7 explains the generation of samples method used in this work. In $genSamples$ function, uniformly distributed samples are generated inside boundaries of the battlefield in terms of dimensions of aircraft state vector $< N, E, D, \psi >$. Then, each generated sample is queried in terms of safety by using inequality 4.6. The remaining set of samples are returned.

Generation of roadmap strategy used in the probabilistic roadmap-based method is represented in Algorithm 8 which implements $genGraph$ function. The resultant graph should be directed because of the nature of Dubins path generation. Then,

Dubins path distances between all samples are calculated. If Dubins path distance is less than two times the minimum turning radius ($2 \times R_{min}$ and greater than $threshold$, the distance between two samples is taken as ∞. Furthermore, if flight path angle γ of flight path segment between two samples is greater than maximum flight path angle γ_{max}, again the distance between two samples is taken as ∞. A safety check is performed for each flight path segment generated by Dubins path generation in order to avoid threat regions. Flight path segments colliding with ellipsoidal volumes are eliminated as well. Otherwise, the adjacency matrix is formed by Dubins path distances between samples.

Algorithm 7 Generation of samples

1: **procedure** genSamples $(bounds, num, threats)$
2: $X \leftarrow \{\}$, $Y \leftarrow \{\}$, $Z \leftarrow \{\}$, $samples \leftarrow \{\}$
3: $X \leftarrow X \cup$ **rand**$(bounds[1], bounds[2], num)$
4: $Y \leftarrow Y \cup$ **rand**$(bounds[3], bounds[4], num)$
5: $Z \leftarrow Z \cup$ **rand**$(bounds[5], bounds[6], num)$
6: $\Psi \leftarrow \Psi \cup$ **rand**$(-\pi, +\pi, num)$
7: $temp \leftarrow X \cup Y \cup Z \cup \Psi$
8: $i \leftarrow 1$
9: **while** $i \leq num$ **do**
10: **if** $temp[i]$ **not** inside any threat
11: $samples \leftarrow samples \cup temp[i]$
12: **end if**
13: $i \leftarrow i + 1$
14: **end while**
15: **return** $samples$

Dijkstra's shortest path algorithm inputs index of start state and roadmap as an adjacency matrix, and outputs shortest trajectory distances of each sample consisted of $samples$ set and predecessor tree, which indicates the index of the previous waypoint in shortest trajectory. Then, the closest possible goal state is searched in $findClosestGoal$ function. Starting from the index of the closest goal state, the shortest trajectory to the goal state is formed by traversing the predecessor tree in $fetchWaypoints$ function.

Algorithm 8 Generation of graph

```
1:  procedure genGraph(samples, threshold, R_min, γ_max)
2:    num ← length(samples)
3:    graph ← Inf(num, num)
4:    for i=1:num
5:      for j=1:num
6:        w_s ← samples(i), w_e ← samples(j)
7:        {path, len} = dubinsPath(w_s, w_e, R_min, γ_max)
8:        if len > threshold or len < 2 * R_min
9:          continue
10:       end if
11:       if isPathSafe(path)
12:         graph[i][j] ← len
13:       end if
14:     end for
15:   end for
16:   return graph                    ▷ Represented as adjacency matrix
```

Figure 3.5 represents one of the results of the customized probabilistic roadmap algorithm. Edges between samples are represented by black line segments, green path represents output trajectory of customized probabilistic roadmap algorithm, possible goal states generated by modeling launch acceptability region queries form a red cone, and ellipsoidal volumes represent surface threats. On the battlefield, there are disorderly placed nine surface threats represented by ellipsoidal volumes. This scenario is simulated with ROSplane autopilot simulator. Aircraft starts its motion in state <$0m$, $0m$, $150m$, $0rad$> and target is placed to <$4800m$, $4800m$, $150m$>. Algorithm designates the aircraft state <$4700.0m$, $4758.4m$, $291.7m$, $0.394rad$> as closest goal state. In order to form the roadmap, 6000 samples are used with a threshold 250 m. The output trajectory has 41 waypoints, where all of them represent an aircraft state.

In this study, the customized probabilistic roadmap path planning algorithm is analyzed in terms of convergence rate and observed trajectory lengths. The main inputs which affect results directly are the number of samples num and the maximum

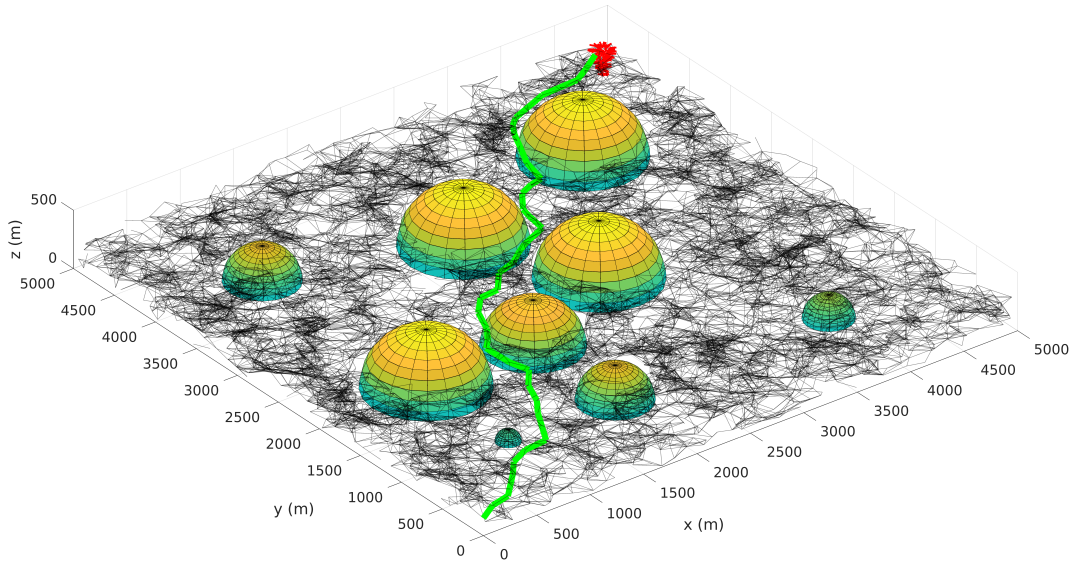

Figure 3.5: One of the results of PRM based path planning algorithm

permitted Dubins path distance between samples $threshold$, and they are used as independent variables in experiments.

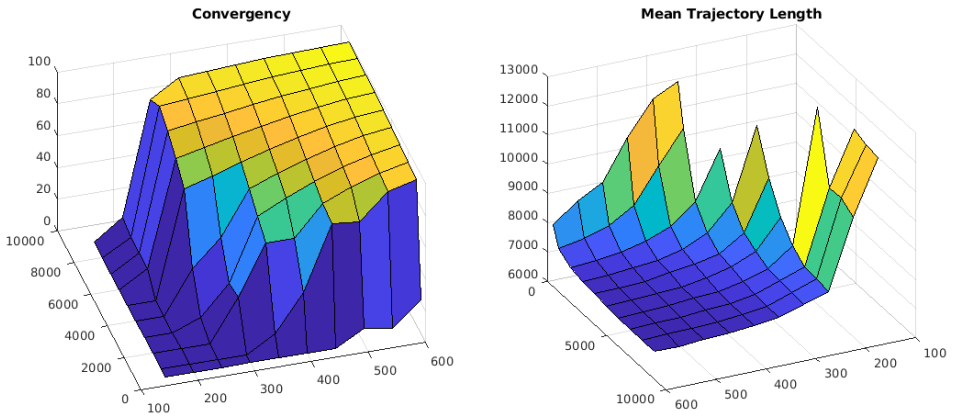

Figure 3.6: Convergence rate and mean trajectory length analyses

Many experiments were done in order to analyze the proposed method in terms of convergence rate and mean trajectory length. Customized probabilistic roadmap path planning algorithm is executed 150 times for each combination of num and $threshold$. The most important observation about convergence rate is that it is greater than 94% and less than 100% when selecting $threshold > 450$ and $num > 6000$. It can be easily seen that when the convergence rate increases, mean trajectory length

decreases. In other words, there is an inverse proportion between them. For instance, in cases which have low convergence rate, mean trajectory length increases up to 12430 m for <$threshold = 200m$, $num = 7000$>. Shortest observed trajectory is 6882.8 m in case of <$threshold = 550m$, $num = 8000$>. The range of tried $threshold$ values are between 150 m to 600 m, with intervals of 50 m. Similarly, the range of tried num values are between 1000 to 8000, with intervals of 1000 m.

Figure 3.6 represents the results of convergence rate and mean trajectory length analysis as surfaces. In both, horizontal axes represent num and $threshold$ values which are used as independent variables in these analyses. The vertical axis represents the convergence rate in convergence rate graph and the mean trajectory length on the right.

3.2 ROSplane Simulation

ROSplane [42] provides an autopilot simulation using a fixed-wing unmanned aerial vehicle. ROSplane is developed on Robot Operating System (ROS) [43] which provides a modular node-based publisher/subscriber environment. ROSplane is developed on ROSflight [49] which procures a bridge between Robot Operating System and ROSflight firmware. Both ROSplane and ROSflight are developed for research purposes, and many research projects are conducted based on them [44]. ROSplane can be applied on real hardware or simulated on Gazebo [50] simulation engine.

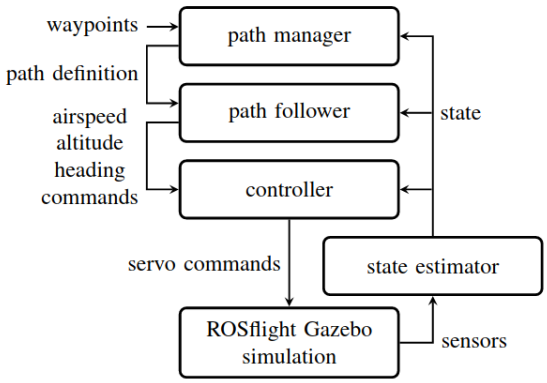

Figure 3.7: Component diagram of ROSplane with ROSflight Gazebo Simulation, taken from [44]

ROSplane consists of a few Robot Operating System nodes. $path\ planner$ node can be implemented to publish the desired trajectory as waypoints with desired flight path segment generation method: fillet or Dubins. $path\ manager$ node constitutes flight path segments based on the desired method. $path\ follower$ put aircraft in desired flight path segment using vector fields. $controller$ node consists of classical PID control loops inside. Finally, $state\ estimator$ node gets inputs from simulated or real sensors and estimates aircraft's navigation information and publishes it to other nodes. Figure 3.7 represents the feedback loop structure of ROSplane autopilot simulation environment used in this study.

3.2.1 Simulation Results

$dynamic\ mission\ planner$ node is implemented, which subscribes target's position and threat volumes, and publishes waypoints of generated trajectory to be followed. $path\ planner$ node is replaced by newly implemented $dynamic\ mission\ planner$ node. Published waypoints are subscribed by $path\ manager$ node in order to generate flight path segments. Figure 3.8 consists of the generated path to be followed, planned waypoints, and aircraft's motion in simulation. It is the same trajectory represented in Figure 3.5.

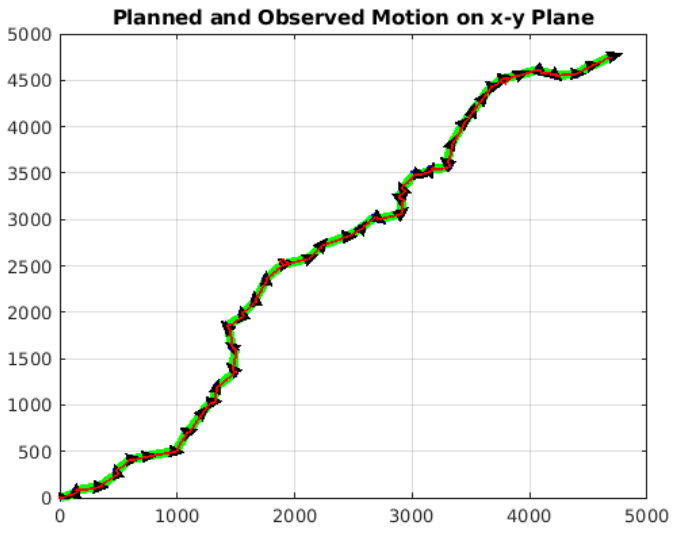

Figure 3.8: Planned and observed motion on x-y plane

In Figure 3.8, waypoints are represented as black arrows, red path represents aircraft's motion, and the green path is the planned trajectory.

By definition, aircraft samples are generated randomly on the battlefield according to probabilistic roadmap algorithm. Dijkstra's shortest path algorithm tries to find the shortest possible path on the generated roadmap. Typically, the shortest path requires optimization, especially in terms of ψ and altitude D. It is easily seen that generated path is not optimal when Figure 3.9 is analyzed adequately where red path represents aircraft motion simulation, black arrows represent the projection of waypoint states on x-y plane, blue paths represent left or right turning arcs, and green paths represent straight motions of generated Dubins flight paths. Minimum turning radius changes according to the linear velocity of aircraft. From one waypoint to another, aircraft may not preserve the magnitude of its linear velocity. Because of that, deviation from planned paths is more significant for turning arcs.

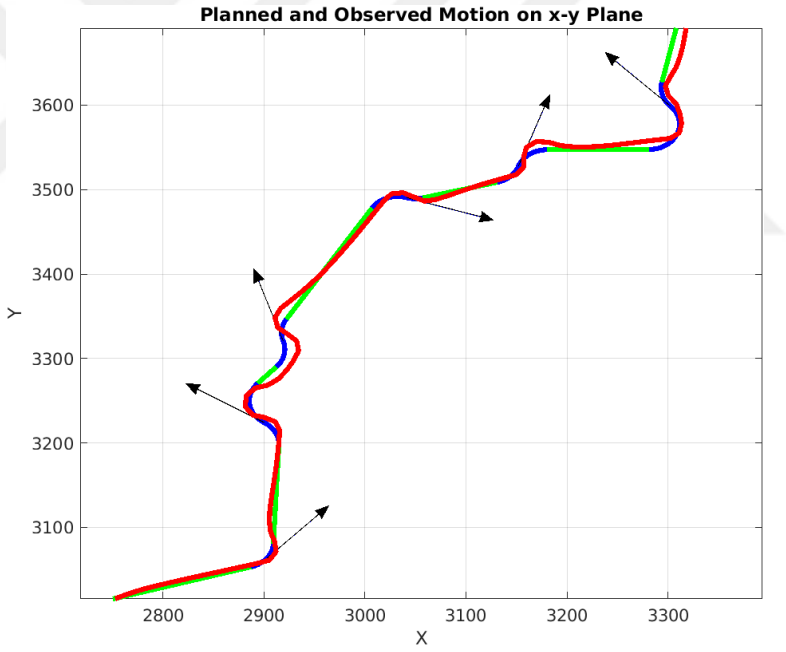

Figure 3.9: A section of planned and observed motion on x-y plane

In order to analyze the capability to follow generated trajectories, 25 flight simulations are carried out for different outputs of the proposed method. Each generated trajectory has different number of waypoints since the roadmap is generated randomly on each trial, and there is not any dependence between waypoint number and deviation

from that waypoint because trajectories differ in each try. The maximum observed deviation from a waypoint is observed as $11.1\ m$, and the mean of deviations in all cases is $2.2\ m$. Note that deviations are calculated with Euclidean distance. Figure 3.10 represents the deviation from waypoints in each trial, and each color represents a flight simulation.

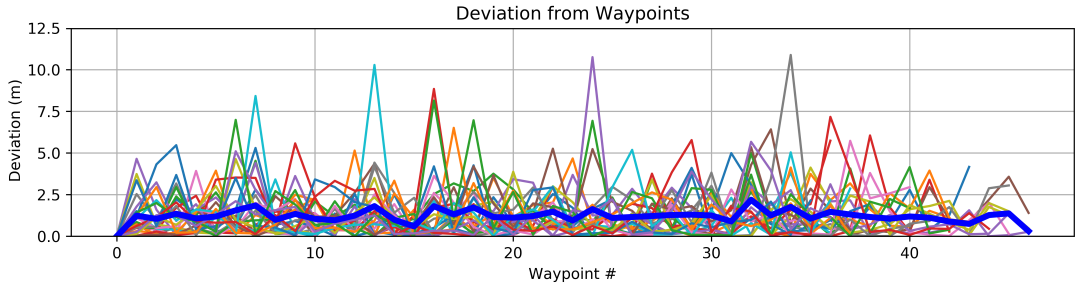

Figure 3.10: Deviation from waypoints in 25 flight simulations

3.3 Conclusions

The proposed method provides dynamic mission planning capability for targets of opportunity. The proposed method is analyzed in terms of convergence rate, mean trajectory length, and deviation from waypoints in flight simulations. However, the method is not analyzed in terms of execution time which is very important for dynamism and determinism of algorithm. To conclude, the lessons taken from this work are listed below:

- Predictive launch acceptability region provides a closer and safer estimation of release point
- A great number of launch acceptability region queries can be done with High Speed 1760
- Querying launch acceptability region is cheaper than adopting it to aircraft software in terms of certification aspects of avionics software
- With using 8000 samples and threshold as $550\ m$, trajectory length is observed as smallest, $6882.8\ m$ after 100 tries for each combination of num and $threshold$

- When the number of samples and maximum Dubins path length between aircraft samples are greater than 6000 and $450\ m$, the convergence rate is greater than 94%
- The maximum deviation observed is $11.1\ m$, and the mean of deviations is $2.2\ m$ in 25 flight simulations with different outputs of the proposed algorithm.
- The algorithm needs to be analyzed in terms of execution time
- Trajectory optimization is needed

CHAPTER 4

VORONOI DIAGRAM-BASED METHOD

4.1 Flight Path Segment Generation with Fillet Algorithm

In the probabilistic roadmap-based method, a directed graph is used for probabilistic roadmap graph because distances between aircraft states can be too short, and the agent is a fixed-wing aircraft. In probabilistic roadmap graph, there should be thousands of aircraft states $< N, E, D, \psi >$ along the battlefield. Therefore, distances between aircraft states were calculated with Dubins path distance, and Dubins path algorithm is used for flight path segment generation. Dubins path algorithm defines the motion of aircraft precisely by considering flight path angle and every waypoint is reached precisely in all the axes of the state, including heading with using Dubins path algorithm. In brief, calculation of the distance between waypoints that are very close in Cartesian space by considering flight dynamics of fixed-wing aircraft is a necessity when probabilistic roadmap algorithm is used.

By nature, distances between boundary points of Voronoi diagrams and are long in large areas with sparse obstacles or threats. Moreover, Voronoi diagram provides great clearance fields from obstacles in large areas with sparse obstacles or threats. Due to these reasons, using Dubins path distance in roadmap generated in Voronoi diagram-based path planning algorithms and Dubins path algorithm as flight path segment generation method is unnecessary.

In the Voronoi diagram-based method, fillet flight path segment generation algorithm [46] can be used with Voronoi diagram-based path planning since smoothed paths do not pose a problem because of these reasons.

However, the distance between two aircraft states cannot be calculated with fillet path algorithm since generating a graph in order to find a trajectory requires distance between only vertices, and distance between two aircraft states depends on third aircraft state. Since, distance calculated by fillet path algorithm is approximately the same for sparse states, Euclidean distance

$$d(p,q) = \sqrt{(p_N - q_N)^2 + (p_E - q_E)^2 + (p_D - q_D)^2} \tag{4.1}$$

can be used where N represents north, E represents east and D represents down in NED frame.

Path smoothing strategy of fillet path algorithm in 2D space is demonstrated in Figure 4.1. Assume that, the angle between vectors $< w_{i-1},\ w_i >$ and $< w_i,\ w_{i+1} >$ is β. A fillet is inserted between two vectors with radius R which should be greater than or equal to minimum turning radius R_{min} of aircraft. Then, the center of the fillet circle is $R\ /\ sin(\beta\ /\ 2)$ distant from waypoint w_i and the distance between waypoint w_i and tangent points is $R\ /\ tan(\beta\ /\ 2)$. Therefore, the actual path distance between w_{i-1} and w_{i+1} is calculated by

$$|W_{2D}| = \left\|w_i - w_{i-1}\right\| + \left\|w_{i+1} - w_i\right\| - 2 \times R \times tan(\frac{\beta}{2}) + R \times \beta. \tag{4.2}$$

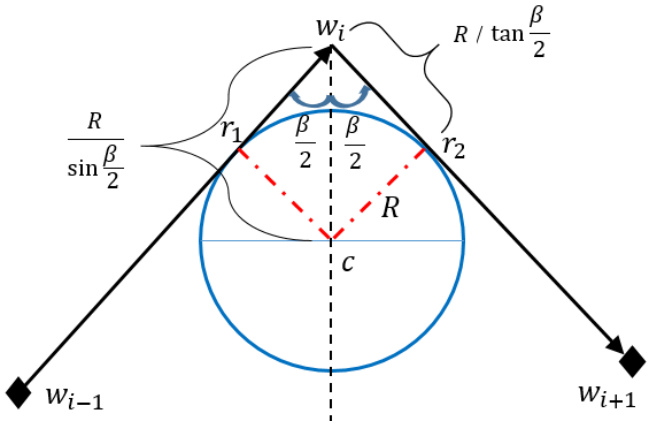

Figure 4.1: Fillet path representation

In 3D space with 3D motion, elevation change should be taken into account as well.

Therefore, the path distance between waypoints in 3D space can be calculated by

$$|W_{3D}| = \sqrt{|W_{2D}|^2 + (|W_{2D}| \times tan(\gamma))^2} \tag{4.3}$$

where γ is flight path angle.

4.2 Enhancement on Launch Acceptability Region Queries

In probabilistic roadmap-based method, many aircraft states are generated uniformly distributed around the target, and launch acceptability region queries are done in order to find out whether the generated states are in launch acceptability region or not. States which are inside of launch acceptability region are inserted to generated aircraft states for probabilistic roadmap algorithm in order to form a roadmap. Generation of aircraft states for launch acceptability region queries in the probabilistic roadmap-based method is represented in Algorithm 9.

Algorithm 9 Generation of Goal States in PRM-based Method

1: **procedure** generateGoalStates ($tgtCoords$)
2: ▷ $tgtCoords$: coordinates of the target
3: ▷ $Sampling\ \ Phase$
4: $xT \leftarrow tgtCoords[0]$
5: $yT \leftarrow tgtCoords[1]$
6: $zT \leftarrow tgtCoords[2]$
7: $R \leftarrow$ **rand**$(0,\ rangeMax,\ numOfGoal)$
8: $\Theta \leftarrow$ **rand**$(0,\ 2\pi,\ numOfGoal)$
9: $Z \leftarrow$ **rand**$(zT,\ zT + zMax,\ numOfGoal)$
10: $goals \leftarrow insertHeadings(R,\ \Theta,\ Z)$
11: ▷ $Query\ \ Phase$
12: $goals \leftarrow eliminateGoals(goals,\ xT,\ yT,\ zT)$
13: **return** $goals$

The sets of R, Θ, and Z are generated with a number of $numOfGoal$ uniformly distributed samples. R indicates the set of uniformly generated radii, Θ indicates the

set of direction angles, and Z is the set of uniformly distributed elevation differences. In addition, $rangeMax$ indicates the maximum range between the generated sample and target on the x-y plane, and $zMax$ indicates the maximum elevation difference between the generated sample and target. Generated goals checked in terms of collision with threat regions and launch acceptability.

Suppose that 1000 goal states are generated and inserted to $samples$ array of probabilistic roadmap algorithm. This operation increases execution time proportionately by the complexity of the probabilistic roadmap algorithm.

In this method, the strategy of generation of goal states is modified in order to shorten execution time by using a greedy approach. Because of the fillet flight path segment generation algorithm, a dummy goal state was inserted into the roadmap in order to reach the correct heading before releasing the weapon. An example scenario is demonstrated in Figure 4.2. It is assumed that the green path on the figure is the derived trajectory, and the red dot is the goal state. Since the safe separation of weapons can be affected by the orientation of aircraft, still turning when releasing the weapon is not feasible enough. Therefore, we need a dummy state between the goal state and the trajectory. Dummy goal state can be calculated by

$$d\_goal =< N_g - cos(\psi_g) \times dist,\ E_g - sin(\psi_g) \times dist,\ D_g,\ \psi_g - \pi/2 > \quad (4.4)$$

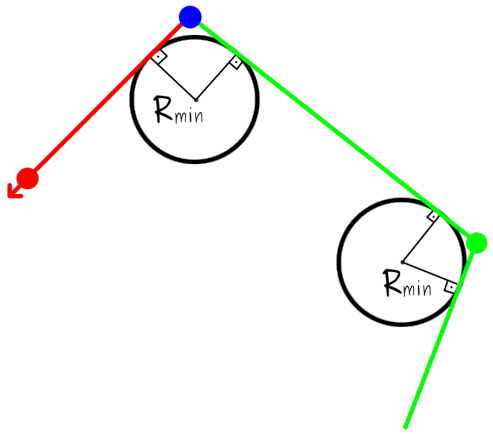

Figure 4.2: Dummy goal state representation

where goal state is $g =< N_g,\ E_g,\ D_g,\ \psi_g >$. If the enhanced goal state generation algorithm finds 10 states which are inside of launch acceptability state, it halts. The enhanced algorithm is represented in Algorithm 10. It is easily seen that the algorithm takes target coordinates as inputs. The algorithm executes the sampling and query phase until finding ten successful goal states. If an aircraft state is successful for weapon release, it is launch acceptable and collision-free with threats and obstacles. At the end of the process, the algorithm outputs ten possible goals and their calculated dummy goals.

Algorithm 10 Generation of Goal States in the Voronoi Diagram-based Method

procedure generateGoal $(tgtCoords)$
▷ $tgtCoords$: coordinates of the target
$goals \leftarrow \varnothing$
$d\_goals \leftarrow \varnothing$
$xT \leftarrow tgtCoords[0]$
$yT \leftarrow tgtCoords[1]$
$zT \leftarrow tgtCoords[2]$
while $\#\ of\ goals\ <\ 10$
▷ $Sampling\ \ Phase$
$r \leftarrow$ **rand**$(0,\ rangeMax,\ 1)$
$\theta \leftarrow$ **rand**$(0,\ 2\pi,\ 1)$
$z \leftarrow$ **rand**$(zT,\ zT + zMax,\ 1)$
$goal \leftarrow insertHeading(r,\ \theta,\ z)$
$d\_goal \leftarrow calculateDummyGoal(goal)$
▷ $Query\ \ Phase$
if $isInsideLAR(goal, tgtCoords)$ **and** $isCollisionFree(goal,\ d\_goal)$
$goals \leftarrow goals \cup goal$
$d\_goals \leftarrow d\_goals \cup d\_goal$
end if
end while
return $goals$, d_goals

4.3 Path Planning with Voronoi Diagram Algorithm

In Chapter 2, probabilistic roadmap, rapidly-exploring random trees, and Voronoi diagram-based path planning algorithm are compared. The comparison states that sampling-based path planning algorithms are successful in converging in very complex environments. On the contrary, the time performance of sampling-based algorithms is very poor since the number of nodes and edges of constructed roadmaps are great. However, Voronoi diagram-based path planning algorithm is sufficient in simple solution spaces with a full convergence rate and high timing performances.

In air-to-surface missions, dangerous fields can be represented as cylindrical, ellipsoidal, or rectangular parallelepiped volumes by pre-loaded mission files or selection of pilot(s). Furthermore, there can be limited air defense systems on a battlefield, and there will be significant distances between them. The battlefield scenario studied in the probabilistic roadmap-based method can be an example of air-to-surface battlefield conditions. Therefore, air-to-surface battlefields can be counted as non-complex solution spaces. In this thesis, a Voronoi diagram-based path planning approach is proposed for air-to-surface mission planning with predictive launch acceptability region queries in addition to the probabilistic roadmap-based method.

The probabilistic roadmap-based method results in approximately at least $62\ s$. Moreover, the execution time of the probabilistic roadmap-based method strictly depends on the number of samples and threshold between vertices of the constructed roadmap, and the convergence rate is wide of full convergence. This means that the probabilistic roadmap-based method is not deterministic. For example, after five attempts of dynamic mission planning, there is a substantial possibility for failure to find a proper trajectory to destroy the target of opportunity. Another issue is that five attempts can take minutes long, and this can be a dramatic failure story. Consequently, the probabilistic roadmap-based method had to be improved in terms of convergence rate and execution time performance. Since Voronoi diagram-based path planning algorithms have good convergence characteristics and are very efficient in simple solution spaces, Voronoi diagrams shape thesis studies.

The Voronoi diagram-based path planning algorithm takes 3D geographical forma-

tions in addition to 3D dangerous volumes. Taking a 2D horizontal cut of the battlefield makes the algorithm simpler, which is important for execution time performance and convergence. Therefore, it can be easily said that the Voronoi diagram-based path planning algorithm has 3D nature.

By assuming horizontal semi axis lengths as same, according to [51], the radius on a specific height of an ellipsoidal

$$\frac{(x-x_c)^2}{r^2}+\frac{(y-y_c)^2}{r^2}+\frac{(z-z_c)^2}{k^2}=1 \tag{4.5}$$

can be calculated by

$$r_h=\frac{r\sqrt{k^2-h^2}}{k} \tag{4.6}$$

where x_c, y_c and z_c indicate centers of threats, r and c indicate semi axis lengths, and h is the elevation. In Figure 4.3, cross sections of threats of the same battlefield in the probabilistic roadmap-based method in elevation of $150\ m$ are demonstrated.

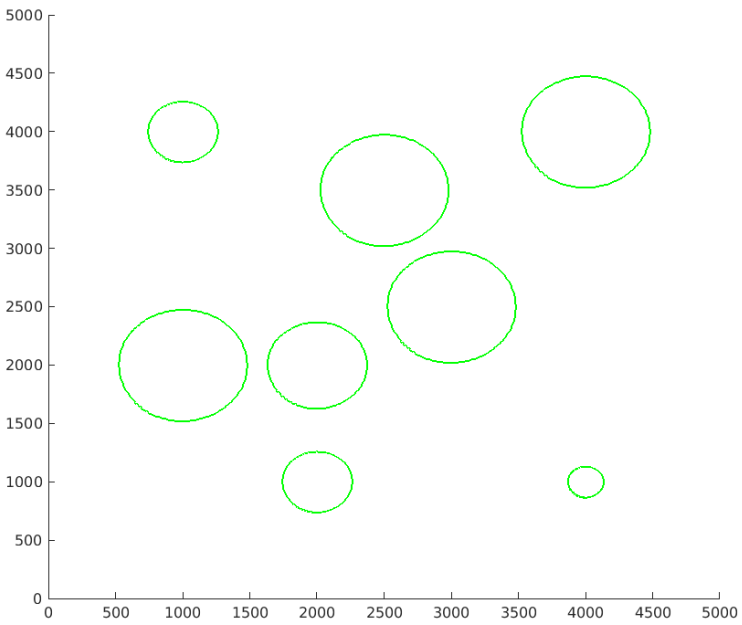

Figure 4.3: Threat cross sections on elevation of $150\ m$

After fetching the cross-section of threats at a specific elevation, Delaunay triangulation algorithm is applied by taking the center of cross-sections as sites. Remember that, Voronoi diagram is the combination of equidistant points from the closest two obstacles. Therefore, Delaunay triangulation is helpful for grouping closest 2's combinations of threats.

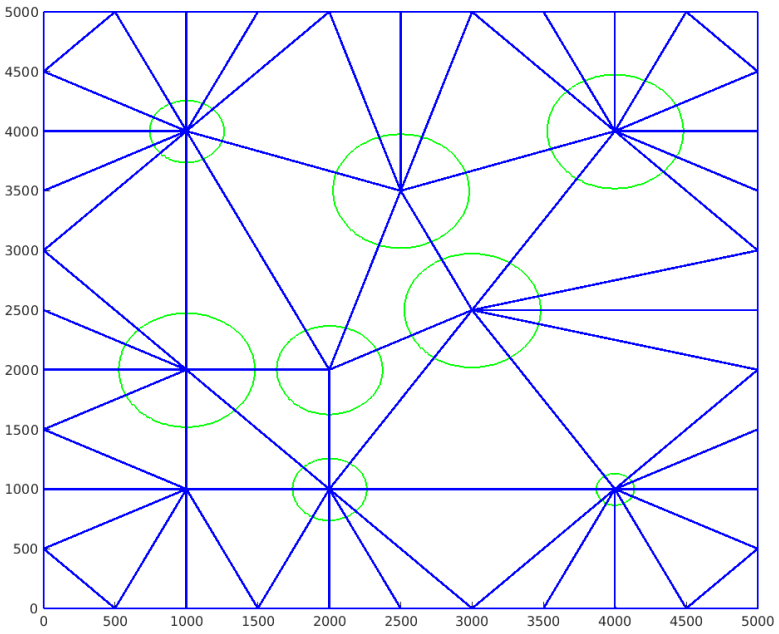

Figure 4.4: Delaunay triangulation

Delaunay triangulation can be done with divide and conquer algorithms easily $O(n\,logn)$ in worst-case [52]. Figure 4.4 is shown Delaunay triangulation application on sites that are the centers of cross-sections of threat regions. In order to limit the bounds of the roadmap, sites are placed on the bounds of the battlefield with specified height and range between them. If threats are the same size, circumcenters of Delaunay triangles can be boundary points of Voronoi diagram. However, every threat has its own dimensions in different semi-axes. For this reason, intersection points between Delaunay triangles and threat cross sections are calculated. In Figure 4.5, intersection points of Delaunay triangles and threat cross sections are represented with red dots. This is important because the edges of Voronoi diagram should have a safe clearance from threats.

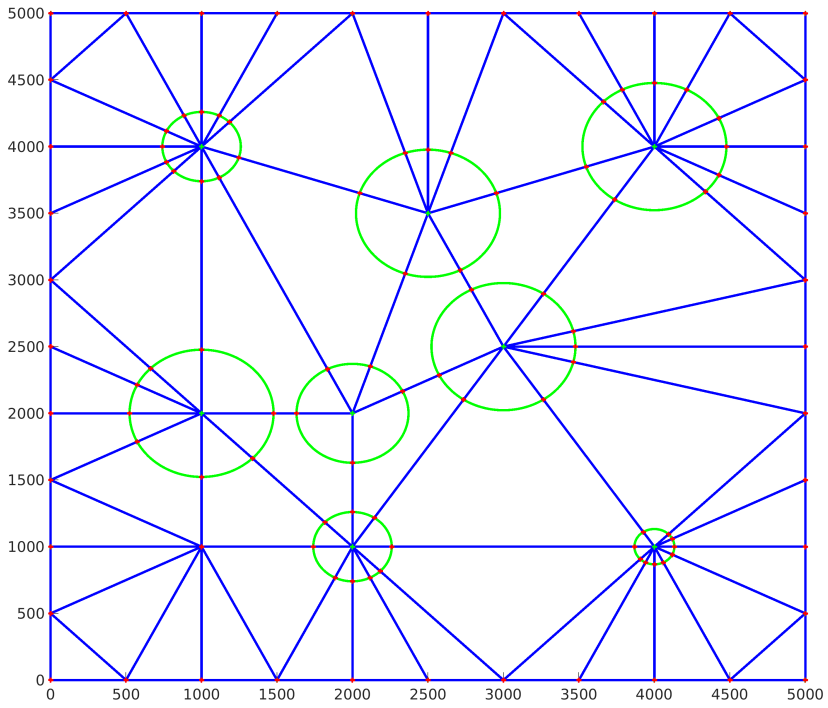

Figure 4.5: Intersections of threat cross sections and Delaunay triangles

After specifying intersection points between Delaunay triangles and threat cross-sections, central points of intersections of each threat cross-section are attained in order to find a point that is distant from each threat cross-section. Then, a triangle constituted with these midpoints. Finally, circumcenter calculation is done for the resultant triangle, similarly [32].

Figure 4.6 demonstrates the method of Voronoi boundaries extraction. Green circles are threat cross-sections; it is easily seen that blue triangle represents Delaunay triangle, red dots indicate intersection points between threat cross-sections and Delaunay triangle, blue dots represent midpoints of intersection points of each threat cross-section, the purple triangle represents the inner triangle which is constituted by midpoints, and finally red dot inside the inner triangle is extracted Voronoi boundary. By applying this method, Voronoi boundary has sufficient clearance from threat cross-sections. Thanks to this, extracting only boundaries is adequate for path planning.

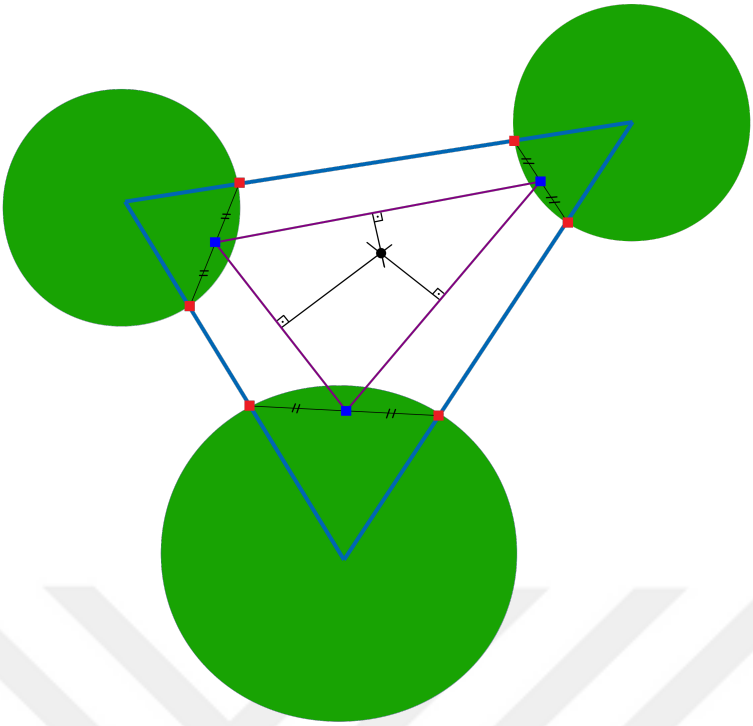

Figure 4.6: Extraction of Voronoi diagram boundary points

An adjacency list based on edge-sharing of triangles is generated. By use of adjacency list and Euclidean distance between Voronoi boundary points, the roadmap is generated as an adjacency matrix of Voronoi boundary points. Euclidean distance is used between Voronoi boundary points because fillet algorithm is used for flight path segment generation, and headings of possible aircraft states are not specified yet.

In order to complete the roadmap, the nearest Voronoi boundary points are found for the initial aircraft state and the goal state. Edges are placed in adjacency matrix between nearest Voronoi boundary points of initial aircraft state and goal aircraft state. The resulting Voronoi diagram roadmap can be seen easily in Figure 4.7 with red line segments in addition to Figure 4.6. The generated roadmap does not collide with any threat cross-section, and there is not any threshold between Voronoi boundary points because Voronoi diagram provides a sufficient clearance from threat cross-sections. One more issue is Voronoi boundary points which are too close to each other. This problem will be significantly resolved by the trajectory optimization method proposed in this thesis.

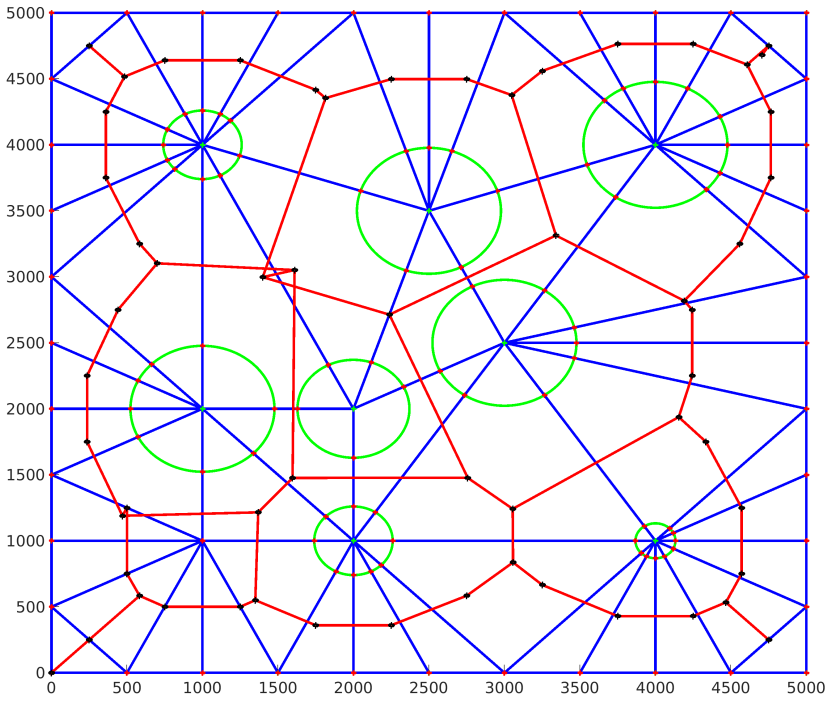

Figure 4.7: Roadmap generation with Voronoi diagram

After generating a roadmap with Voronoi diagram, the shortest path should be found for trajectory suggestion to pilot. Dijkstra's shortest path algorithm is used to find the shortest trajectory. Since the number of edges and vertices of the new roadmap is almost reduced approximately 100 times and the complexity of Dijkstra's shortest path algorithm is $O(|E| + |V|log|V|)$ when Fibonacci heap is used where E is the number of edges and V is the number of vertices in the roadmap, the new approach brings an obvious run-time efficiency. Furthermore, roadmap generation in probabilistic roadmap algorithm takes $O(V^2)$. Nevertheless, Voronoi diagram is created in $O(V\ logV)$ by using the divide and conquer Delaunay triangulation algorithm in this thesis.

In Figure 4.8, the derived shortest path is represented with the bold blue path. It is easily seen that the derived trajectory is not an optimal case in free configuration space. However, Voronoi diagram helps us to find a collision-free path that can be optimized to be an optimal trajectory.

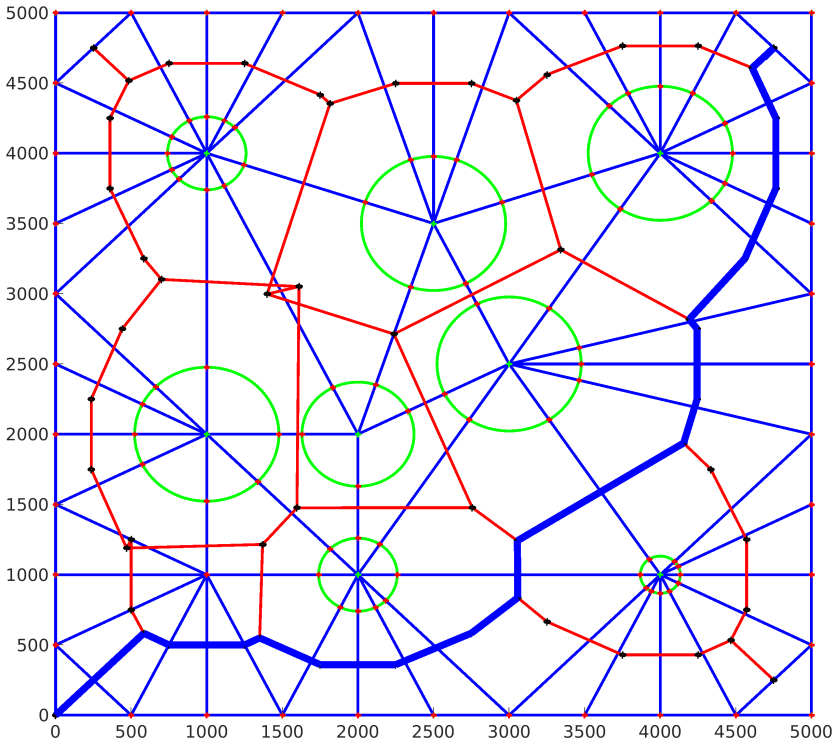

Figure 4.8: Voronoi diagram with shortest path

Generation of roadmap with Voronoi diagram-based path planning algorithm is explained in Algorithm 11.

Algorithm 11 Generation of Roadmap with Voronoi Diagram

procedure voronoi $(threats,\ bounds)$

▷ $threats$: coordinates and semi-axis lengths of threats

▷ $bounds$: bounds of the battlefield

$X \leftarrow threats(:,\ 1)$

$Y \leftarrow threats(:,\ 2)$

$R \leftarrow threats(:,\ 3)$

$X,\ Y,\ R \leftarrow insertBoundarySites(X,\ Y,\ R,\ bounds)$

$triangulation \leftarrow delaunay(X,\ Y)$

9: $adjacencyList,\ vertices \leftarrow findCircumcenters(triangulation,\ X,\ Y,\ R)$

10: **return** $triangulation,\ adjacencyList,\ vertices$

First of all, x and y coordinates and radii of threats are separated into sets. Then, sites are generated on the boundaries of the battlefield and inserted into the constituted sets. Delaunay triangulation is done using sets of x and y coordinates of sites. Finally, boundary points of Voronoi diagram (circumcenters) are calculated according to the radii of threats by the method represented in 4.6. According to adjacencies of triangles, an adjacency list for Voronoi boundary points is formed. Finally, triangulation array ($triangulation$), adjacency list ($adjacencyList$), and Voronoi boundary points ($vertices$) are returned at the end of the algorithm.

4.3.1 Weaknesses and Solutions

Many scenarios were taken into consideration, and weaknesses of the proposed Voronoi diagram-based algorithm are analyzed in this thesis. Constituting a roadmap with only Voronoi boundary points is very efficient in the case of the time complexity of the algorithm. However, ignoring other equidistant points which are between Voronoi boundaries can result in a collision with threat cross sections by considering the ellipsoidal shape of threats.

According to analyses, possible weaknesses are divided into two types: intersecting threats and ragged edges. Intersecting threats issue can be resolved by detecting an edge's collision with two closes threats and ignoring the collided edge.

4.3.1.1 Intersecting Threats

Air defense systems can cover common volumes in a variety of battlefields. This causes a colliding edge in the roadmap, and it is a vital problem in terms of safety. In Figure 4.9, an example of intersecting threats is demonstrated. Although the threats are colliding, Delaunay triangulation is done, and circumcenters are calculated according to the resultant triangulation.

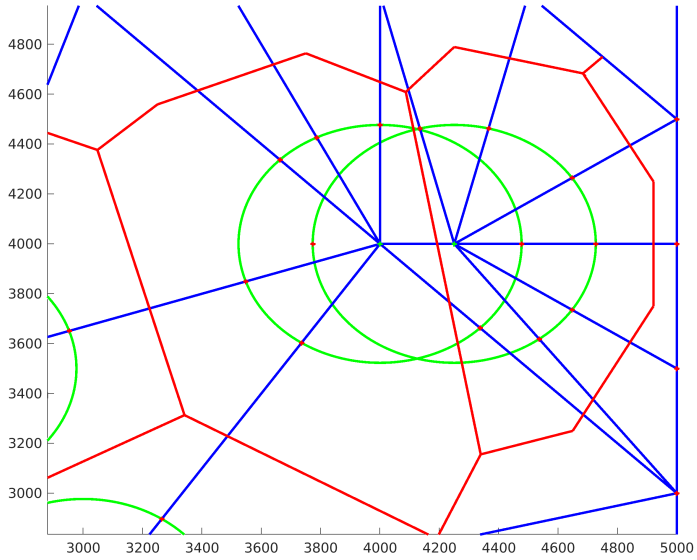

Figure 4.9: Weakness: Intersecting threats

All the edges are checked in terms of collision with two closes threat cross-sections. If so, the related edge is removed from the roadmap. In Figure 4.10, the result of enhancement is presented.

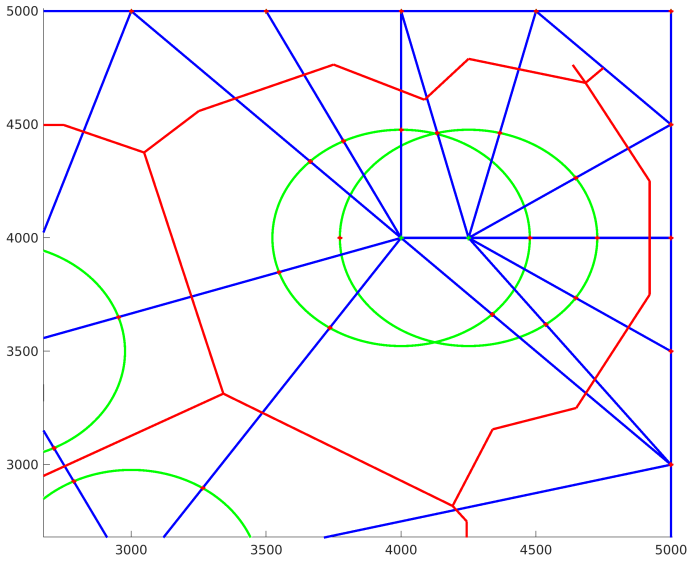

Figure 4.10: Solution: Intersecting threats

4.3.1.2 Ragged Edges

Some edges in the roadmap can collide with threat cross-sections. This is also a vital problem in terms of safety, and it is difficult to resolve this issue. In Figure 4.11, an example of ragged edge is demonstrated. The algorithm outputs the orange edge, which collides with a threat cross-section.

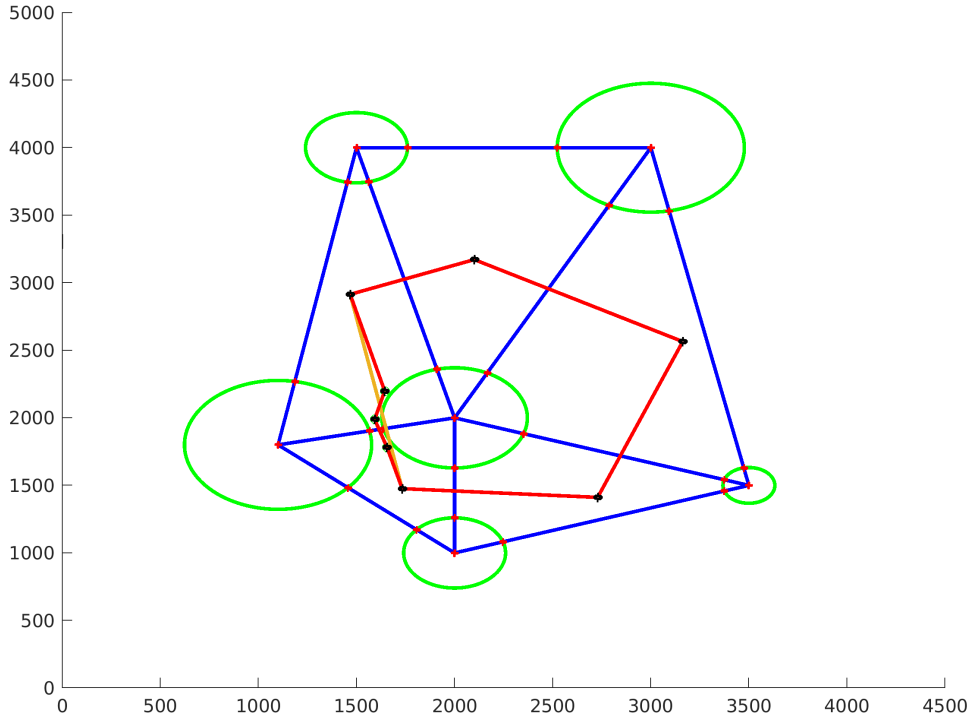

Figure 4.11: Weakness: Ragged edge problem

First of all, ideal distance from two threat cross sections are calculated with

$$k = \frac{\left\|c_1,\ c_2\right\| - r_1 - r_2}{2} \tag{4.7}$$

where c_1 and c_2 are centers of two closest threats, r_1 and r_2 are radii of threat cross sections. Therefore, the turning radius around colliding threat cross section will be

$$r_k = r_1 + k \tag{4.8}$$

assuming that O_1 is the colliding threat. Angular period around colliding threat cross section to place additional vertices to the roadmap on the purpose of getting rid of collision should satisfy the inequality

$$\alpha < 2\ cos^{-1}\left(\frac{r_1}{r_k}\right). \tag{4.9}$$

Assume that Voronoi boundary points are p_1 and p_2, the edge between p_1 and p_2 is removed and additional vertices between p_1 and p_2 and around colliding threat cross section are placed with less than a period $2\ cos^{-1}\left(\frac{r_1}{r_k}\right)$ in order to generate collision-free edges. The resultant vertices are connected between respectively and adjacency matrix should be updated according to this enhancement.

It can be easily assumed that the orange edge connects p_1 and p_2, and the colliding threat cross section is O_1 in Figure 4.12. Three new vertices calculated around O_1 which are not collide with O_2 as well with a frequency α from $< c_1,\ p_1 >$ to $< c_1,\ p_2 >$. The resulting path between O_1 and O_2 is represented by red line segments, which are collision-free.

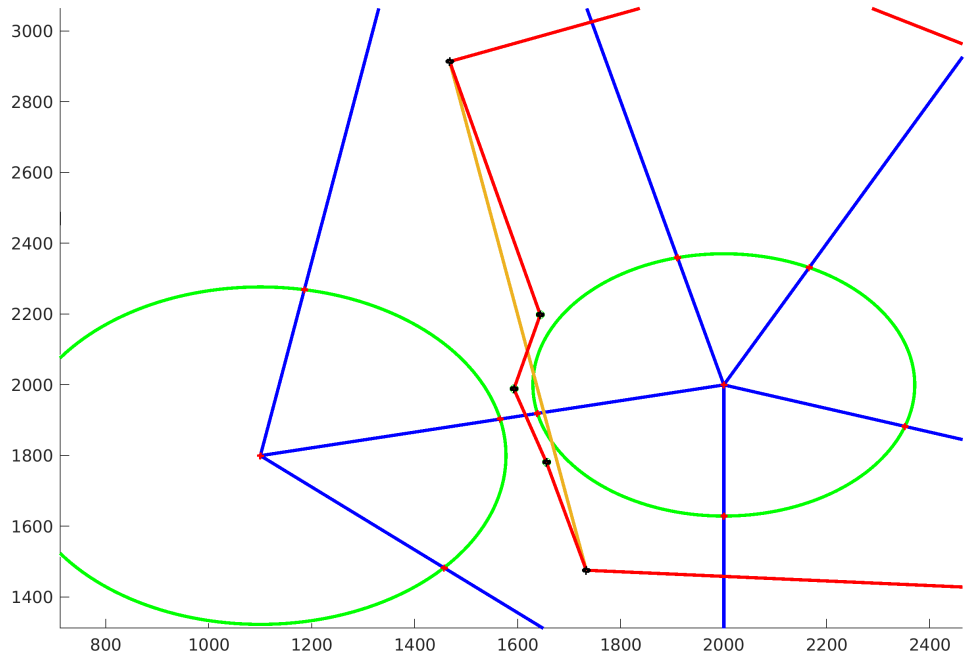

Figure 4.12: Solution for ragged edge

4.3.2 Trajectory Optimization

The resultant trajectory is $8225\ m$ long by using end-to-end Fillet path distance. This result is slightly bad when comparing with the results of the proposed probabilistic roadmap algorithm. Moreover, the resultant paths require optimization for the probabilistic roadmap-based method. On the other hand, the resultant trajectory of Voronoi diagram-based path planning algorithm needs path optimization as well. Sampling-based algorithms are generally more successful in terms of mean trajectory length. However, it is possible to reach the same success by applying trajectory optimization on trajectories generated by Voronoi diagram-based path planning algorithm.

In this thesis, a novel greedy trajectory optimization method is proposed, and analyses are done on the results. The optimization starts from the beginning node and has two steps for each node on the trajectory: forward propagation and edge division. In forward propagation, the furthermost next waypoint is searched for each waypoint on the trajectory, which can be considered as the next state of source waypoint; in other words, a collision-free edge can be placed between the source waypoint and furthermost next waypoint. If a further waypoint is found which can be considered as the next state, the trajectory will be pruned. In backward propagation, the found edge is divided into a number of segments. Segments are formed by placing some intermediate waypoints on edge. An intermediate waypoint that can have a collision-free edge with the next-next waypoint of source waypoint on the divided edge is searched. If found, the next waypoint of the source waypoint is removed, and the found intermediate waypoint is placed on the trajectory instead of that. The trajectory optimization algorithm is represented in Algorithm 12.

Algorithm 12 Trajectory Optimization

1: **procedure** optimize $(threats,\ trajectory,\ index)$
2: ▷ $threats$: coordinates and semi-axis lengths of threats
3: ▷ $trajectory$: trajectory array which includes aircraft states
4: ▷ $index$: recursion depth
5: **if** $index >= length(trajectory) - 1$
6: **return** trajectory
7: **end if**

8: $trajectory \leftarrow forward\_propagation(threats,\ trajectory,\ index)$

9: $trajectory \leftarrow edge\_division(threats,\ trajectory,\ index)$

10: **return** $optimize(threats,\ trajectory,\ index + 1)$

The algorithm starts its execution from the start state by taking it as the source state and iterates over the aircraft states of the trajectory, which is the output of Voronoi diagram-based path planning algorithm. $forward\_propagation$ function searches a possible furthest waypoint that can be concatenated with the source waypoint in a collision-free manner. If found, the trajectory is pruned. Then, the edge between the source state and the next state is divided into segments to prune the edge. The trajectory is modified again if a sufficient intermediate waypoint is found. Finally, the algorithm is called again for the next state, with index incrementation.

Forward propagation and edge division steps of trajectory optimization are represented in Figure 4.13 and 4.14.

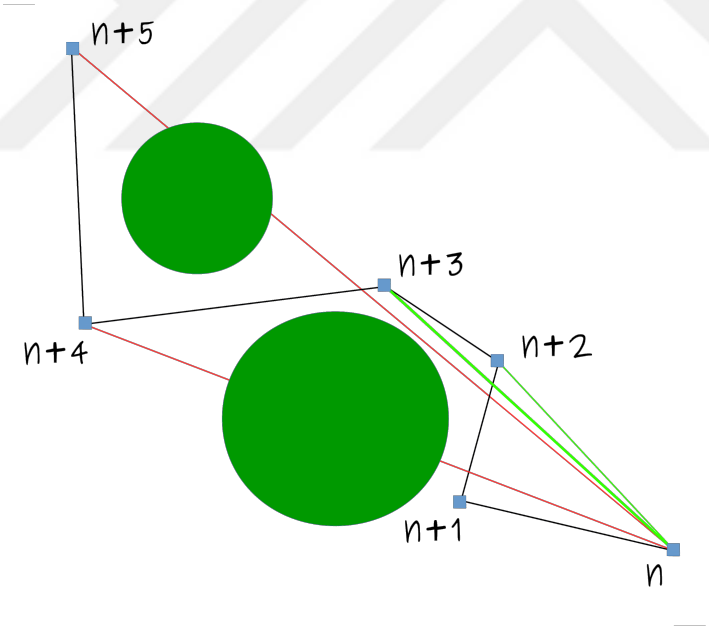

Figure 4.13: Trajectory optimization algorithm: Forward propagation

In 4.13, source waypoint is n and assume that black path is input trajectory. All the further waypoints are checked by placing an edge between them and the source waypoint for collision. In this case, $n+3$ is the furthest and can be collision-free next

state. Therefore, $n+1$ and $n+2$ are pruned from the input trajectory. In other words, green edge segments are collision-free, and red line segments are colliding edges.

The same forward propagation step is applied in 4.14. $n+3$ is chosen as next state and an edge placed between n and $n+3$. Then, the placed edge is divided by intermediate aircraft states on edge. It is easily seen that the eighth intermediate aircraft state is the nearest state around suitable states to form an edge with $n+4$. Then, $n+3$ is placed with found intermediate aircraft state.

An example of optimized output trajectory is demonstrated with the green path in Figure 4.15. In the end, the length of the optimized trajectory is $7169\ m$. This means that the proposed trajectory optimization method provides 12.8% saving from trajectory length in this scenario.

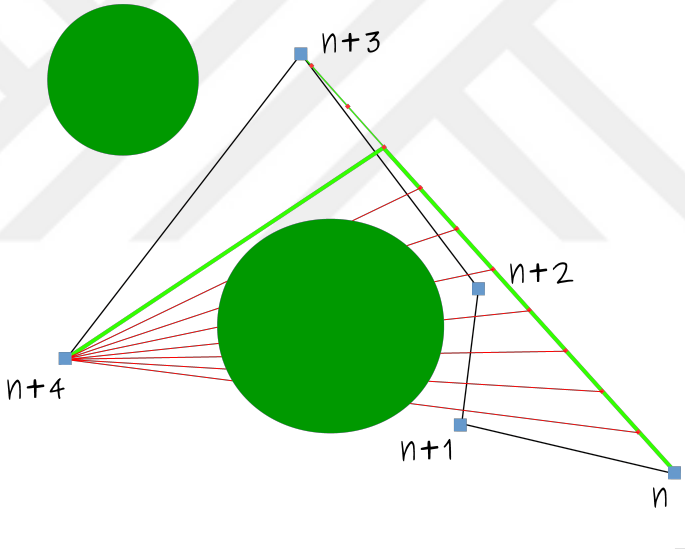

Figure 4.14: Trajectory optimization algorithm: Forward propagation and edge division

Suppose that there is not any threats on the battlefield. In that case, Delaunay triangulation is done with the sites placed on the bounds of the battlefield and a very simple Voronoi diagram is acquired which is represented by red line in Figure 4.16. The output of Dijkstra's shortest path algorithm and the optimized trajectory are represented with magenta and green lines respectively.

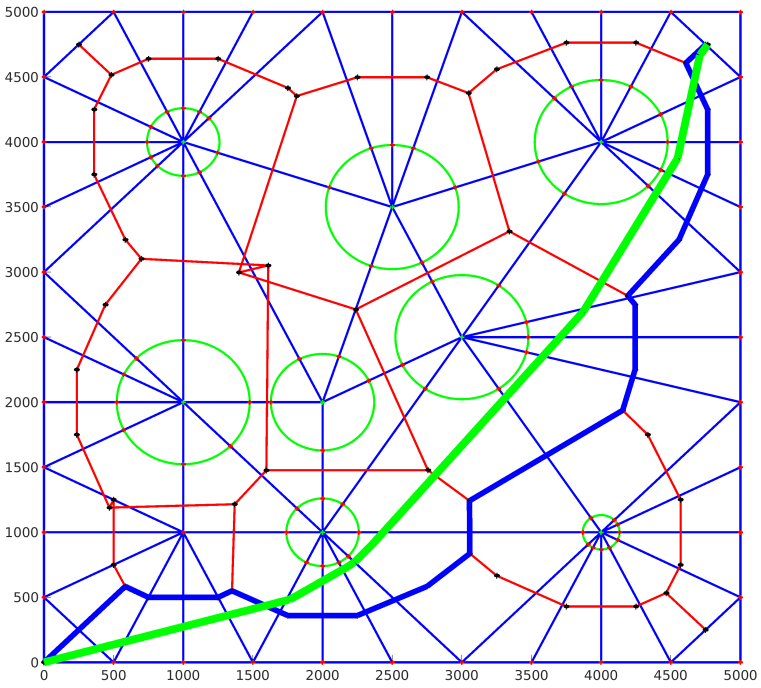

Figure 4.15: Optimized trajectory

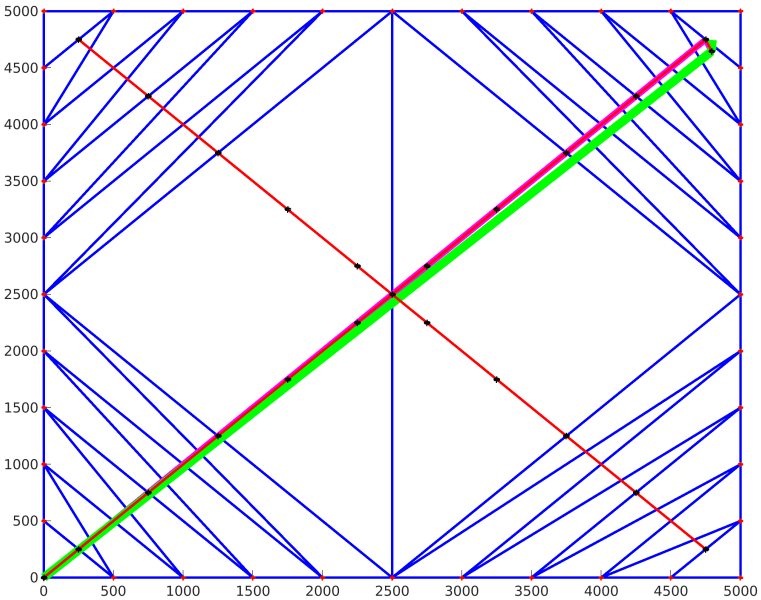

Figure 4.16: Path planning in a battlefield without threats

In Figure 4.17, the output of Dijkstra's shortest path (magenta) and the optimized trajectory (green) intersect at dummy goal state. The algorithm directly specifies the dummy goal state as the next state after the start state. At the end of the optimized trajectory (green), aircraft release the weapon.

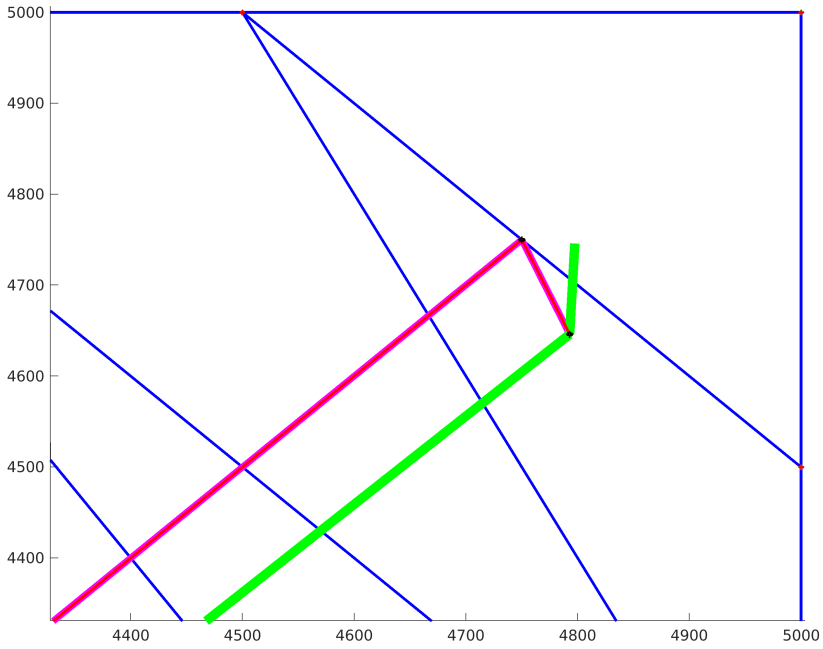

Figure 4.17: Dummy goal state in a battlefield without threats

4.3.3 Terrain Collision Avoidance

In addition to surface threats, geographical formations should be taken into account as well when planning a path because aircraft shall avoid not only threats but also geographical formations. In well-known robotics applications, obstacle avoidance has a significant place. However, avoiding threats has vital importance as much as obstacle avoidance has in the military avionics domain. Geographical formations do not change over time unless volcanic eruptions, lands slides, etc. happen. On the contrary, surface threats can be movable, and new threats may be identified dynamically. For this reason, trajectory planning is based on surface threats, and planned trajectories are processed according to geographical formations. In other words, avoiding

threats has a higher priority than avoiding obstacles when planning a trajectory because threats have dynamic nature.

In order to simulate geographical formations, a digital elevation map of pre-eruption mountain Saint Helens is used [53]. In Figure 4.18, its normalization to $0 - 255$ version is demonstrated. Normally, the peak was $2951\ m$, but elevations are scaled with $1/10$. In addition to it, the map has $30\ m$ resolution, and it is scaled with $1/30$ in simulations. Since the ROSplane simulation has $12\ m/s$ speed and the average speed of a military jet aircraft is approximately $240\ m/s$, scaling geographical formations by $1/30$ in width and $1/10$ in height is logical.

Figure 4.18: Saint Helens Mountain digital elevation map, normalized to greyscale

After that, the scaled elevation map is placed to coordinates $< x = 2200\ m,\ y = 1900\ m >$, $< x = 3300\ m,\ y = 3450\ m >$ and the remaining parts are filled by terrain with $80\ m$ height. Consequently, there are two mounts on the elevation map, and the mounts are placed on coordinates that are in the aircraft's trajectory. In Figure 4.19, a digital elevation map of the battlefield with two simulated mountains is represented in greyscale format. Additionally, geographical formations and surface threats are demonstrated in Figure 4.20. The blue path in Figure 4.20 represents the output of Voronoi diagram-based path planning algorithm, and the red path represents the

optimized version of the output of Voronoi diagram-based path planning algorithm. It is as clear as a crystal that optimized path collides with geographical formations. In this thesis, a generic terrain collision avoidance method is developed for optimized trajectory.

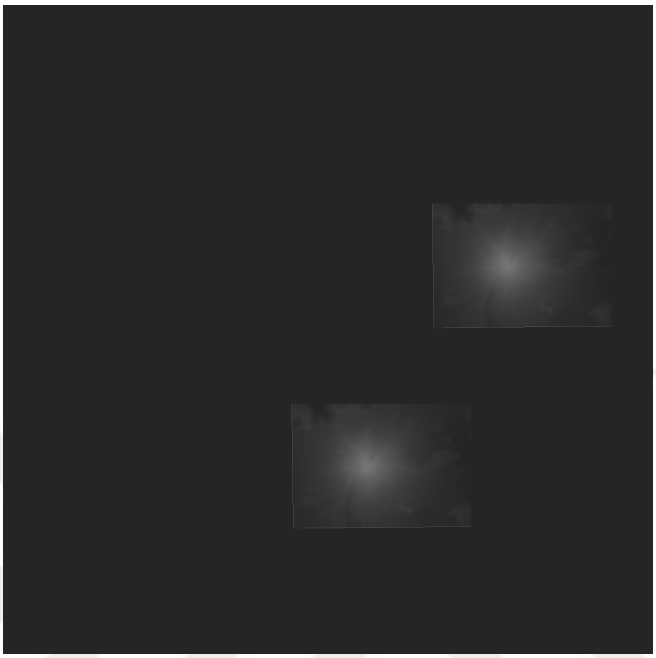

Figure 4.19: Elevation map of battlefield with two simulated mountains

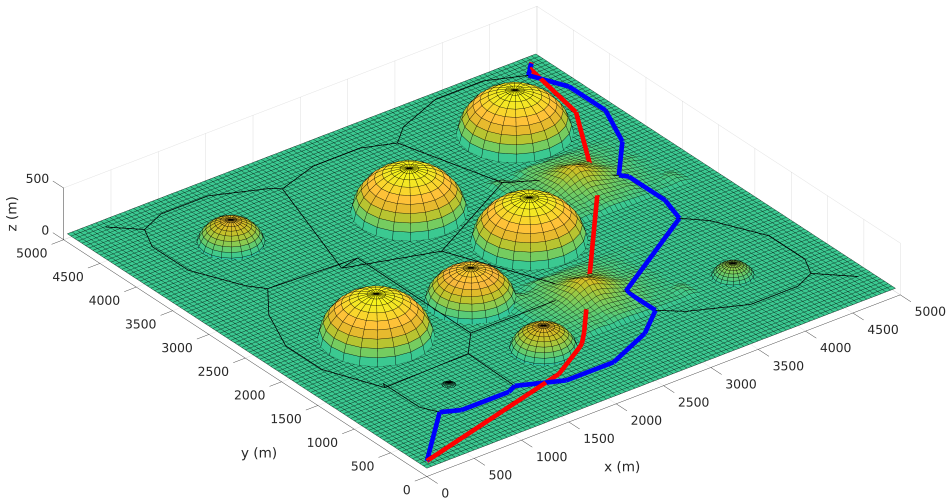

Figure 4.20: Battlefield with geographical formations and surface threats

In order to get rid of collision with geographical formations, the minimum height of the optimized trajectory, which is sufficient for collision avoidance, is converted to a 1D signal. Then, a safety margin with $20\ m$ is placed on elevations different than standard terrain higher than $150\ m$ height. In addition, dummy step elevations with $151\ m$ are placed at the beginning and at the end of the 1D signal in order to take advantage of Hilbert transform. In Figure 4.21, the blue line represents the minimum height of the optimized trajectory, and the orange line represents the signal with the application of safety margin.

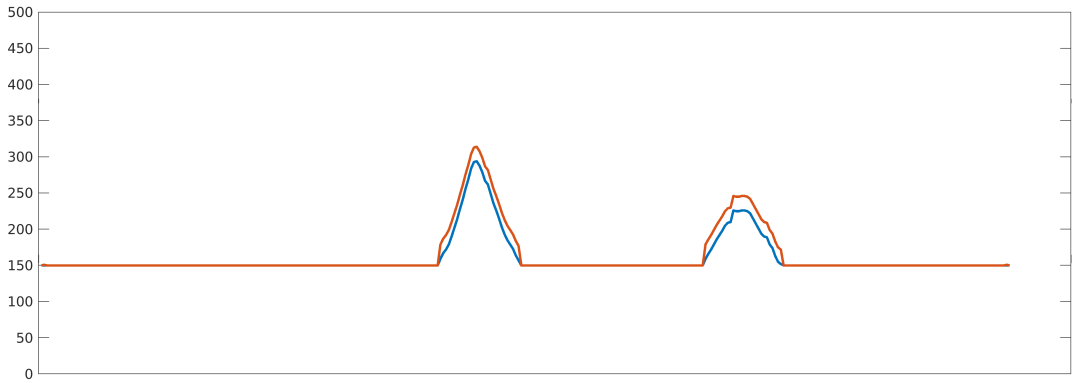

Figure 4.21: 1D signal of elevations on optimized trajectory with and without safety margin of $20\ m$

Hilbert transformation is used to derive upper envelope signal of 1D elevation signal since Hilbert transform outputs the minimum phase response and discrete time analytical signal can be found by

$$X = X_r + i \times X_i \tag{4.10}$$

where X_i is Hilbert transform of X_r [54]. After all, the magnitude of discrete-time analytical signal forms the upper envelope signal of the input signal, which represents the 1D elevation sequence. The upper envelope signal is useful because the trajectory should envelop at least the minimum height signal. Its peaks can be used for new milestones for the elevation change of moving aircraft.

The upper envelope signal is represented by the yellow line in Figure 4.22. It can be easily seen that it covers all the elevation changes, and it envelopes the elevation

signal with the safety margin.

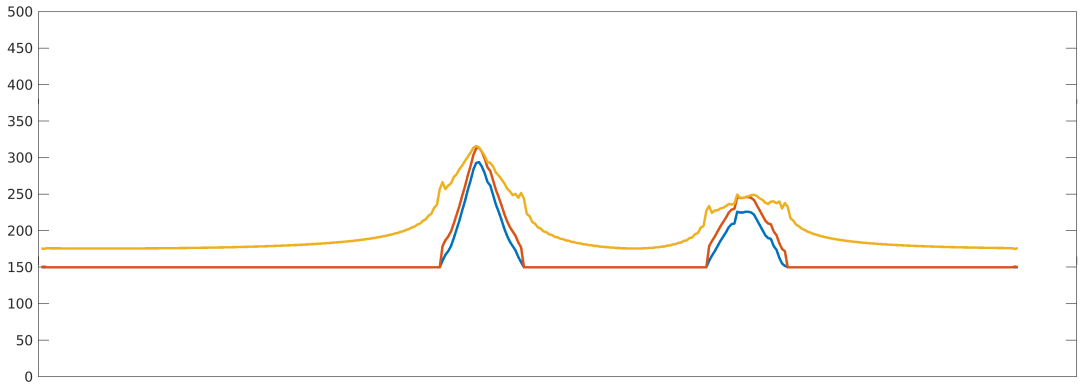

Figure 4.22: Yellow line: Upper envelope signal derived using Hilbert transform

Then, peaks of the upper envelope are extracted by detecting local maxima points. Local maxima points are detected in discrete time by finding indices that are greater than their neighbors. After finding peaks of the upper envelope signal, they are concatenated in order to form new milestones of elevation change in the trajectory. The purple line represents in Figure 4.23 the concatenation of peaks of upper envelope signal of minimum elevations with a safety margin of $20\ m$.

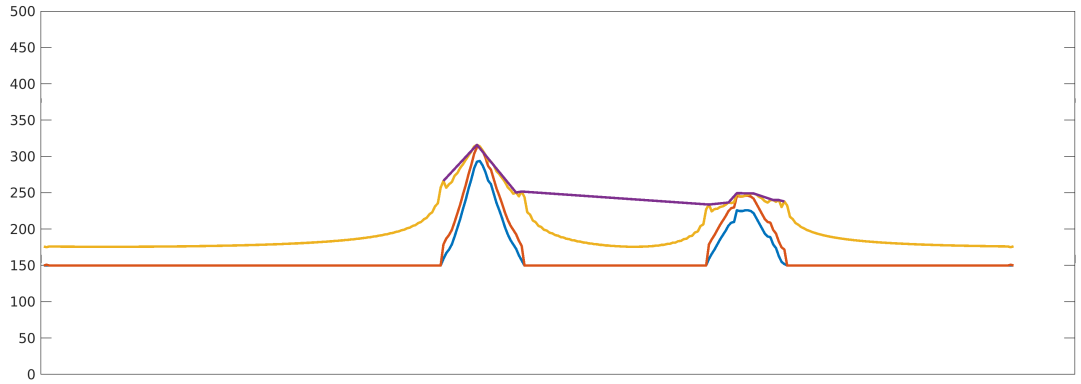

Figure 4.23: Purple line: Concatenation of peaks of upper envelope signal

After extracting peaks of the upper envelope signal, they are inserted into the trajectory with respect to their projections on the minimum elevation signal. Any waypoint which is between any inserted waypoints is raised to the slope between inserted waypoints. The green line in Figure 4.24 represents elevations of the resultant trajectory,

which consists of inserted and raised waypoints.

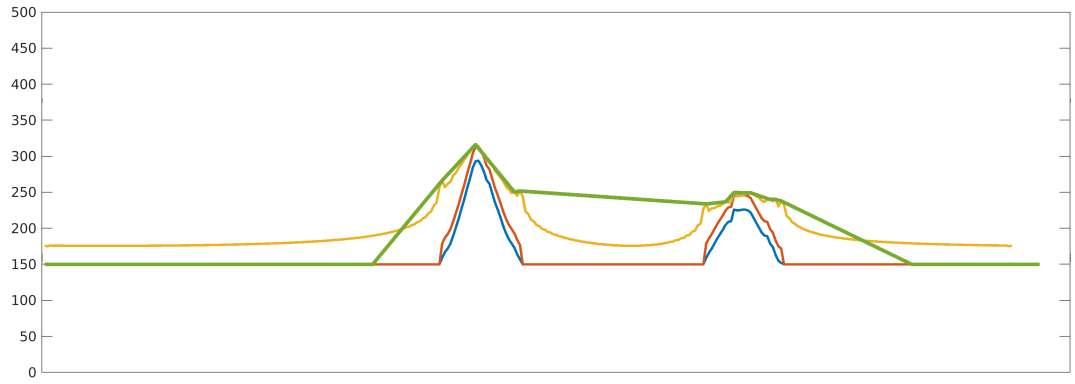

Figure 4.24: Elevation signal of resultant trajectory which consists of inserted and raised waypoints

Finally, although the resultant trajectory consists of all the peak points of the upper envelope signal and intermediate waypoints are raised to the slope, which corresponds to neighbor peaks, the resultant trajectory cannot be followed by aircraft because of the maximum flight path angle. The resultant trajectory should be enhanced according to the result path angle of the aircraft. In this scenario, the elevation of each waypoint on the resultant trajectory is updated recursively according to maximum flight path angle $\gamma_{max} = 0.15\ rad$. Recursion in the climb enhancement algorithm halts when there is no change in elevations of waypoints of the trajectory. If there is no proper solution for climbing over the slope, the algorithm returns $N\ I\ L$. Climb enhancement of trajectory algorithm represented in Algorithm 13 and Algorithm 14.

Algorithm 13 Climb Enhancement of Trajectory

procedure enhance_climb $(trajectory,\ \gamma_{max})$
▷ $trajectory$: trajectory array which includes aircraft states
▷ γ_{max}: maximum flight path angle
$change\_flag \leftarrow true$
while $change\_flag$
$trajectory,\ change\_flag \leftarrow enhance\_climb\_helper(trajectory,\ \gamma_{max})$
end while
return $trajectory$

Algorithm 14 Helper Function for Climb Enhancement of Trajectory

procedure enhance_climb_helper $(trajectory,\ \gamma_{max})$
▷ $trajectory$: trajectory array which includes aircraft states
▷ γ_{max}: maximum flight path angle
$change\_flag \leftarrow false$
for $i = 1 : length(trajectory) - 1$
$p_1 \leftarrow trajectory[i]$
$p_2 \leftarrow trajectory[i+1]$
$h\_dist \leftarrow euclidean\_dist(p_1(1:2),\ p_2(1:2))$
$\gamma \leftarrow atan2(p_2(3) - p_1(3),\ h\_dist)$
if $\gamma > \gamma_{max}$
$prev\_ind \leftarrow -1$
for $j = i - 1 : -1 : 1$
$p_{prev} \leftarrow trajectory[j]$
$h\_dist_{prev} \leftarrow euclidean\_dist(p_{prev}(1:2),\ p_2(1:2))$
$\gamma_{prev} \leftarrow atan2(p_2(3) - p_{prev}(3),\ h\_dist_{prev})$
if $\gamma_{prev} < \gamma$
$prev\_ind \leftarrow j$
break
end if
end for
if $prev\_ind == -1$
return $N\ I\ L$
else
$change\_flag \leftarrow true$
return $update\_slope(prev\_ind,\ i+1,\ trajectory), change\_flag$
end if
end if
end for
return $trajectory, change\_flag$

The result of the climb enhancement operation on trajectory is represented with light blue line in Figure 4.25.

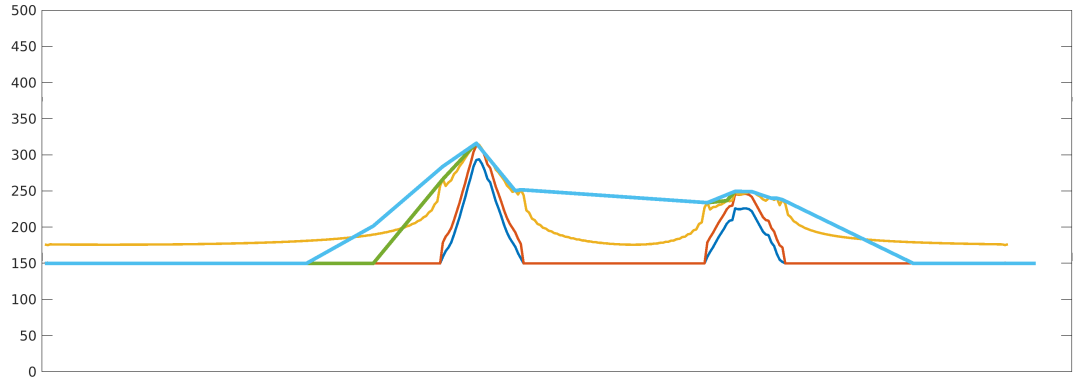

Figure 4.25: Light blue line: Result elevation signal of terrain collision avoidance algorithm

The whole picture of Voronoi diagram-based path planning algorithm with trajectory optimization, terrain collision avoidance with and without climb enhancement is represented in Figure 4.26 and 4.27 from different perspectives.

- Threats and geographical formations are represented with green-yellow surface according to elevation values.
- Output of Voronoi diagram-based path planning algorithm is represented with the dark blue line.
- Output of trajectory optimization process is represented with the red line.
- Output of terrain collision operation without climb enhancement is represented by the light blue line.
- Output of terrain collision operation without climb enhancement with respect to maximum flight path angle is represented by the green line.

It is easily seen that the path represented by the green line is safe in terms of colliding with threat ellipsoids and geographical formations and very short in regards to reaching the goal state.

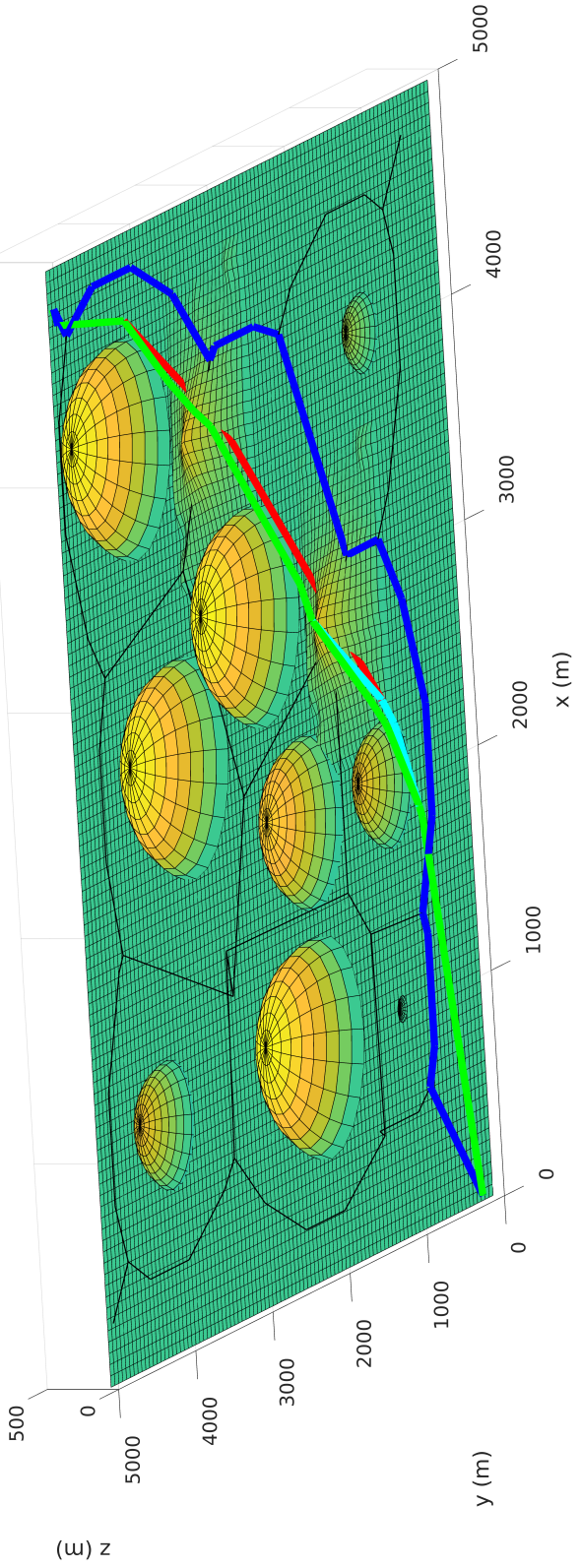

Figure 4.26: Optimized path with terrain collision avoidance

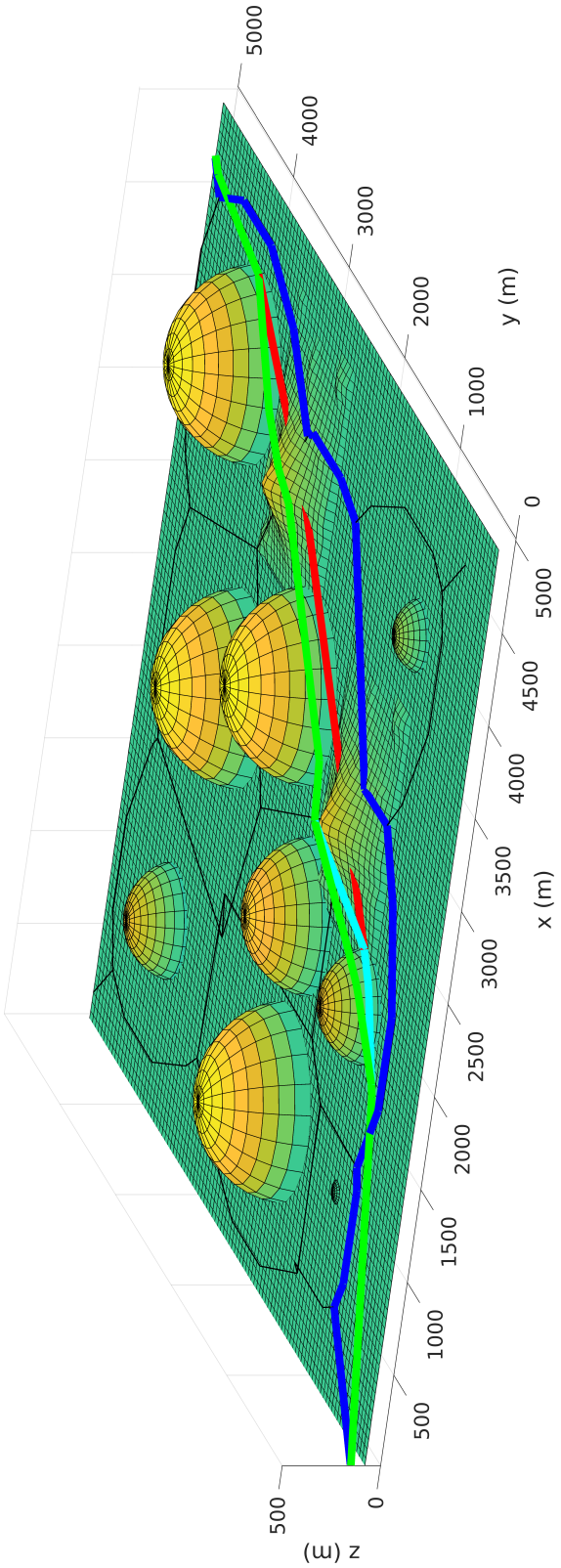

Figure 4.27: Optimized path with terrain collision avoidance

4.4 Results and Comparison of Proposed Methods

Dynamically planning a mission is a mission-critical task in all military domain software applications, and it must be accomplished successfully as soon as possible. Therefore, a dynamic mission planning algorithm should behave in a deterministic manner. Furthermore, optimality of output trajectory is another consideration for the sufficiency of dynamic mission planning algorithm. Output trajectory should not be a sight worse than any pilot's preference. Both of the proposed methods are analyzed in terms of

- convergence rate,
- mean trajectory length, and
- execution time.

In terms of convergence rate, the Voronoi diagram-based path planning algorithm is overwhelmingly powerful than probabilistic roadmap algorithm because the observed convergence rate is 100% independently of variables of probabilistic roadmap algorithm, num and $threshold$.

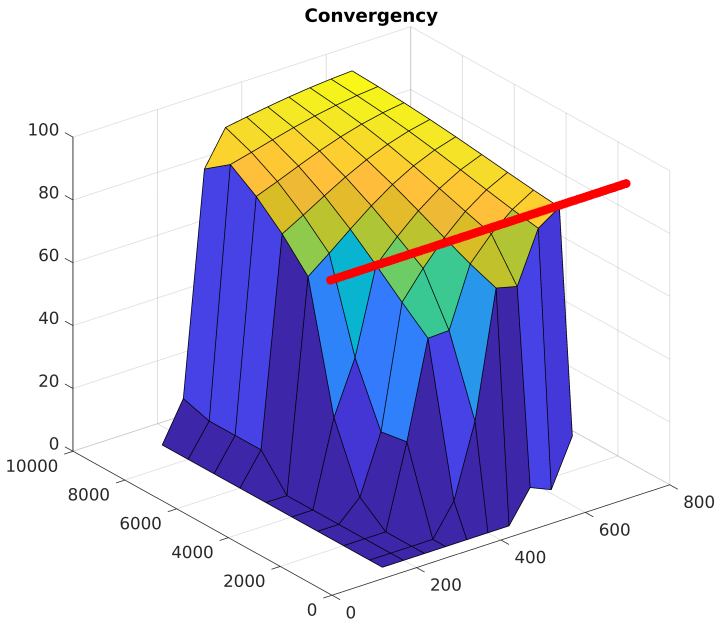

Figure 4.28: Convergence rate analysis and comparison

In Figure 4.28, the surface represents observed convergence rates of the probabilistic roadmap-based method, and the red line represents the convergence of the Voronoi diagram-based method (100%).

For the probabilistic roadmap-based method, mean trajectory length is analyzed in terms of the number of samples generated and the maximum Dubins distance between the edges of PRM graph. According to the results of the probabilistic roadmap-based method, the minimum trajectory length is observed as $6882.8\ m$ with 8000 samples and taking threshold as $550\ m$ after 100 trials. Additionally, the mean of all trials with all independent variables (num and $threshold$ combinations) is about $8107\ m$. For example, the trajectory length in the case represented in Figure 3.5 is $7982.7\ m$. In Figure 4.29, the surface represents mean trajectory length values of probabilistic roadmap-based method for independent variables (num and $threshold$), the blue line represents of non-optimized trajectory's length as $8225.3\ m$, the red line represents of optimized trajectory's length as $7169.3\ m$, and finally, the green line represents of the resultant length of terrain collision avoidance applied optimized trajectory as $7192.6\ m$. All the line representations belong to the analysis of the proposed Voronoi diagram-based method.

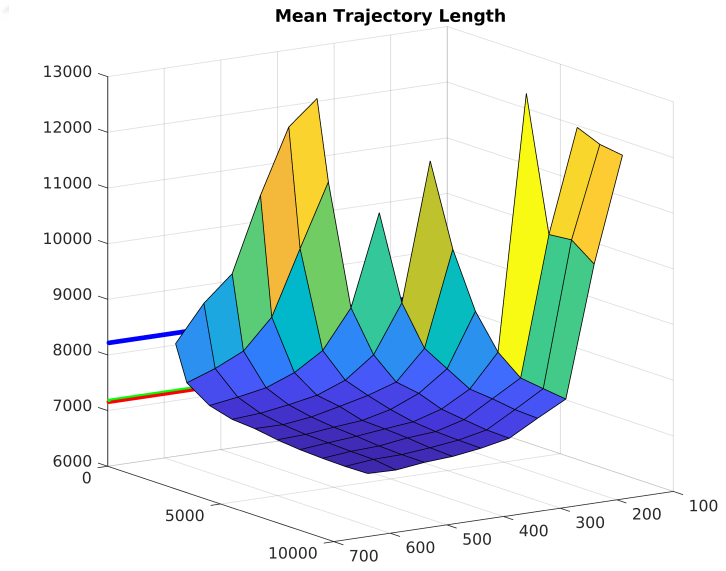

Figure 4.29: Mean trajectory length analysis and comparison

It is easily seen that the mean trajectory length performance of Voronoi diagram-based

path planning is worse than the average of the results obtained for the probabilistic roadmap-based method. However, the mean is decreased to $7169.3\ m$ (12.8%) with the proposed trajectory optimization method. This is not better than the best result of the proposed probabilistic roadmap algorithm but can be a complete substitute by considering convergence rate and execution time performances of the proposed Voronoi diagram-based path planning algorithm.

Since dynamic mission planning is a mission-critical task that should be accomplished immediately, execution time analysis of proposed algorithms has vital importance. In Figure 4.30, average execution times of each combination of the number of samples generated ($samples$) and maximum Dubins distance between the edges of PRM graph ($threshold$) is represented as surface, and the average execution time of Voronoi diagram-based path planning algorithm with trajectory optimization and terrain collision avoidance is represented with the red line.

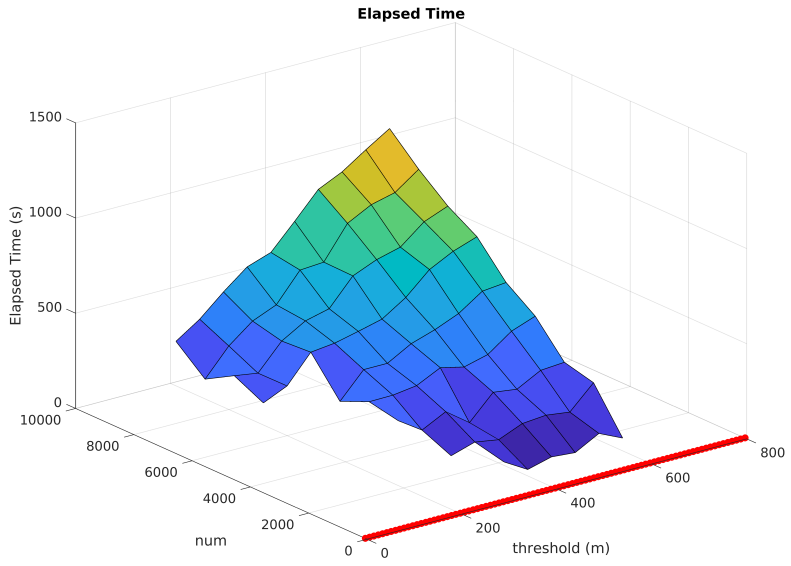

Figure 4.30: Execution time analysis and comparison

The average execution time of Voronoi diagram-based path planning algorithm with trajectory optimization and terrain collision avoidance is $0.60\ s$. Since the best execution time obtained with the probabilistic roadmap-based method is $62\ s$, the execution time of the Voronoi diagram-based method is at least 100 times less than the proba-

bilistic roadmap-based method despite new features, i.e., trajectory optimization and ground collision avoidance. Technical specifications of the computer used for experiments are listed in Table 4.1.

Table 4.1: Technical Specifications of Computer

Resource	Specification
Memory	15.5 GiB DDR3
Processor	Intel® Core™ i7-3630QM CPU @ 2.40GHz × 8
Operating System	Ubuntu 18.04.5 LTS
OS Type	64-bit

By using ROSplane simulation environment, 25 different flight simulations are done for 25 different outputs of the proposed Voronoi diagram-based method. In Figure 4.31, each color represents deviations from waypoints for separate flight simulations, and the thickest blue line represents the average deviation from each waypoint in all flight simulations.

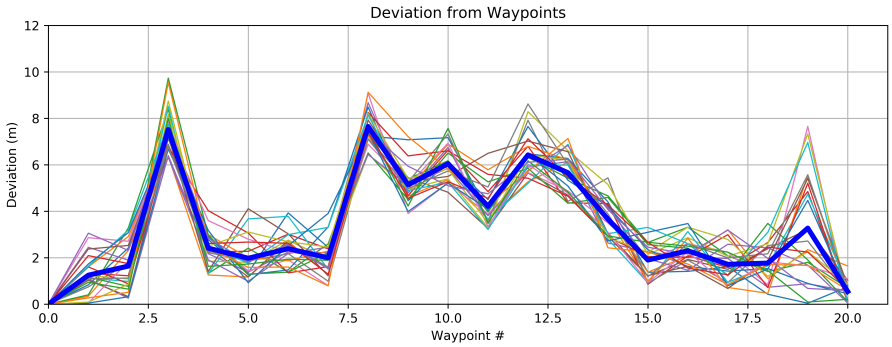

Figure 4.31: Deviation from waypoints of 25 outputs of the Voronoi diagram-based method

Differently from waypoint deviation from waypoint analysis for the probabilistic roadmap-based method (represented in Figure 3.10), there is a pattern here. Because the Voronoi diagram-based algorithm finds almost the same trajectory except for the dummy goal state, which is one state before the release point of the weapon, and it is different for each output of the proposed algorithm. It can be easily seen that the max-

imum deviation does not exceed $10\ m$, and the average deviation in whole waypoints is observed as $3.308\ m$. Results show that generated trajectories can be followed easily by a fixed-wing aircraft by taking its flight characteristics into consideration.

CHAPTER 5

CONCLUSIONS

Dynamic mission planning is a crucial requirement for a military aircraft in order to carry out a mission with moving threats and targets and targets of opportunity. Deviating from previously planned waypoints in a mission can lead to safety problems in a dynamic battlefield, and the pilot's decisions should be supported by aircraft.

Artificial intelligence applications make a difference in the military domain, especially when it is used in military avionics because speed in air missions is drastically high. Running artificial intelligence on avionic computers has many key requirements: computation power, network speed, the performance of data links between military vehicles, and safety considerations. All of them can be grouped into two parts: computational power and safety considerations. Military avionics should have smaller size, weight, and power consumption and affordable cost, then having high computational performance with size, weight, and power consumption constraints are not easy [55]. Furthermore, all avionics software has to be certified according to guidelines like RTCA/DO-178C. The impact of change should be analyzed, and a variety of actions should be taken when avionics software is modified according to RTCA/DO-178C. For this reason, a dynamic air-to-surface mission planning scenario should be generic and compatible with different air-to-surface weapons, including future ones. Predictive launch acceptability region approach is a solution for compatibility of dynamic air-to-surface mission planning with different air-to-surface weapons and provides flexibility and safety for specifying a goal aircraft state for releasing weapons.

A dynamic air-to-surface mission planning approach is proposed in the probabilistic roadmap-based method using predictive launch acceptability region queries. How-

ever, outputs of the algorithm need trajectory optimization, and the algorithm does not take geographical formations into consideration. Furthermore, observed convergence rates and execution time performance require enhancements. Because of these reasons, a Voronoi diagram-based path planning method is proposed in addition to the probabilistic roadmap-based method, solutions for its weaknesses are developed, the output of the Voronoi diagram-based algorithm is optimized by a novel approach, and finally, a Hilbert transformation-based terrain collision avoidance method is developed.

The Voronoi diagram-based path planning algorithm provides 100% convergence rate, which is much better than the sampling-based probabilistic roadmap algorithm. Furthermore, the Voronoi diagram-based method is at least 100 times faster than the probabilistic roadmap-based method, although it has trajectory optimization and terrain collision avoidance functionalities. Mean trajectory length of the Voronoi diagram-based method is 11.3% smaller than the average of experiment results of the probabilistic roadmap-based method; however, it is not better than the best observation of the probabilistic roadmap-based method. Twenty-five flight simulations are done for twenty-five different outputs of the Voronoi diagram-based method, and maximum deviation does not exceed $10\ m$, and the average deviation in whole waypoints is observed as $3.308\ m$ which is far smaller than the safety margin value ($20\ m$) used in terrain collision avoidance method.

In this thesis, efficient and applicable air-to-surface mission planning approaches are proposed by considering safety-critical software development and certification considerations. In future work, research will be conducted by considering air-to-air threats for air-to-air missions as well, and a human-machine interface may be proposed for the future method.

REFERENCES

[1] Sun, W. G., Ye, H. B., Wang, K., & Song, Y. J. (2012). Design of Autonomic Logistics Information System Based on System Integration Model. Advanced Materials Research, 472-475, 2650–2654.

[2] Pereira, F. D., Ordonez, E., Sakai, I. D., & de Souza, A. M. (2013). Exploiting parallelism on keccak: Fpga and gpu comparison. Parallel & Cloud Computing, 2(1), 1-6.

[3] Software Considerations in Airborne Systems and Equipment Certification. RTCA document DO-178C, 2011.

[4] L. Rierson, Developing Safety-Critical Software: A Practical Guide for Aviation Software and DO-178C Compliance. Boca Raton, FL: Taylor & Francis, 2013.

[5] Moir, I. (2019). Military avionics systems. John Wiley & Sons.

[6] K. A. Rigby, Aircraft Systems Integration of Air-Launched Weapons. Chichester, West Sussex, United Kingdom: Wiley, 2013.

[7] AS5653B, High Speed Network for MIL-STD-1760, 03 January 2014.

[8] Wilburn, J., Perhinschi, M., Wilburn, B., Enhanced Modified Voronoi Algorithm for UAV Path Planning and Obstacle Avoidance, (2013) International Review of Aerospace Engineering (IREASE), 6 (1), pp. 54-63.

[9] J. Gruszecki & F.L. Basmadji (2007) Rough pilot decision-support algorithm, Aviation, 11:3, 31-36, DOI: 10.1080/16487788.2007.9635967

[10] Flight Training Instruction: Strike T-45 MPTS and IUT (2017). Department of the Navy Chief of Naval Air Training. CNATRA P-1209 (Rev. 02-17).

[11] Edwards, K. L. (2004). Air–to–ground targeting — UAVs, data links and interoperability (project Extendor). The Aeronautical Journal, 108(1088), 493–504. doi:10.1017/s0001924000000324

[12] Yoon, K. S., Park, J. H., Kim, I. G., & Ryu, K. S. (2010). New modeling algorithm for improving accuracy of weapon launch acceptability region. 29th Digital Avionics Systems Conference. doi:10.1109/dasc.2010.5655454

[13] Elder, R. L. (1986). An examination of circular error probable approximation techniques. AIR FORCE INST OF TECH WRIGHT-PATTERSON AFB OH SCHOOL OF ENGINEERING.

[14] MIL-STD-1553B, 21 September 1978.

[15] MIL-STD-1760E, 25 October 2007.

[16] ANSI INCITS 356-2002 Fibre Channel - Audio Video, 25 November 2002.

[17] Fiber Channel FC-AE-1553 [S], Revo.8,April 6.2006

[18] Migneault, G. E. (1980, May). Software reliability and advanced avionics. In Proceedings of the May 19-22, 1980, national computer conference (pp. 715-720).

[19] Software Considerations in Airborne Systems and Equipment Certification. RTCA document DO-178B, 1992.

[20] Youn, W. K., Hong, S. B., Oh, K. R., & Ahn, O. S. (2015). Software certification of safety-critical avionic systems: DO-178C and its impacts. IEEE Aerospace and Electronic Systems Magazine, 30(4), 4–13. doi:10.1109/maes.2014.140109

[21] L. E. Dubins, “On curves of minimal length with a constraint on average curvature, and with prescribed initial and terminal positions and tangents,” American Journal of Mathematics, vol. 79, no. 3, p. 497, 1957.

[22] Software Tool Qualification Considerations. RTCA document DO-330, 2011.

[23] Model-Based Development and Verification Supplement to DO-178C and DO-278A. RTCA document DO-331, 2011.

[24] Object-Oriented Technology and Related Techniques Supplement to DO-178C and DO-278A. RTCA document DO-332, 2011.

[25] Formal Methods Supplement to DO-178C and DO-278A. RTCA document DO-333, 2011.

[26] N.-H. Quttineh and T. Larsson, “Military aircraft mission planning,” Optimization Letters, vol. 9, no. 8, pp. 1625–1639, 2014.

[27] Ozdemir, M. R., Cevher, L., Ertekin, S., ”AI-based Air-to-Surface Mission Planning using Predictive Launch Acceptability Region Approach”, 2021 International Conference on Military Technologies (ICMT), 2021, Brno, Czech Republic, June 8-11, 2021. 10.1109/ICMT52455.2021.9502830

[28] Department of Defense, & Joint Chiefs of Staff. DOD Dictionary of Military and Associated Terms. Independently published, 2019.

[29] Howie Choset, Kevin Lynch, Seth Hutchinson, George Kantor, Wolfram Burgard, Lydia Kavraki, and Sebastian Thrun. Principles of Robot Motion: Theory, Algorithms, and Implementation -Errata. 2010.

[30] P. Cheng and S. M. LaValle. Reducing metric sensitivity in randomized trajectory design. In IEEE/RSJ International Conference on Intelligent Robots and Systems, pages 43–48, 2001.

[31] F. Aurenhammer. Voronoi diagrams—A survey of a fundamental geometric structure. ACM Computing Surveys, 23:345–405, 1991.

[32] Davis, Jennifer & Perhinschi, Mario & Wilburn, Brenton & Karas, Ondrej. (2012). Development of a Modified Voronoi Algorithm for UAV Path Planning and Obstacle Avoidance. AIAA Guidance, Navigation, and Control Conference 2012. 10.2514/6.2012-4904.

[33] M. Candeloro, A. M. Lekkas, J. Hegde and A. J. Sørensen, "A 3D dynamic Voronoi diagram-based path-planning system for UUVs," OCEANS 2016 MTS/IEEE Monterey, 2016, pp. 1-8, doi: 10.1109/OCEANS.2016.7761427.

[34] Candeloro, Mauro & Lekkas, Anastasios & Sørensen, Asgeir. (2017). A Voronoi-diagram-based dynamic path-planning system for underactuated marine vessels. Control Engineering Practice. 61. 41-54. 10.1016/j.conengprac.2017.01.007.

[35] Hvamb, K. (2015). Motion Planning Algorithms for Marine Vehicles.

[36] Madhavan Shanmugavel, Antonios Tsourdos, Brian White, Rafał Żbikowski, Co-operative path planning of multiple UAVs using Dubins paths with clothoid arcs, Control Engineering Practice, Volume 18, Issue 9, 2010, Pages 1084-1092, ISSN 0967-0661, https://doi.org/10.1016/j.conengprac.2009.02.010.

[37] Zhang, Yu & Chen, Jing & Shen, Lincheng. (2013). Real-time trajectory planning for UCAV air-to-surface attack using inverse dynamics optimization method and receding horizon control. Chinese Journal of Aeronautics. 26.1038-1056.10.1016/j.cja.2013.04.040.

[38] L. J. Clancy, Aerodynamics. Harlow, Essex: Longman Scientif. & Techn., 1986.

[39] "MAM-C Smart Micro Munition," Roketsan. [Online]. Available: https://www.roketsan.com.tr/en/product/mam-c-smart-micro-munition/. [Accessed: 14-Feb-2021].

[40] R. D. Blevins, Applied Fluid Dynamics Handbook. Malabar, FL: Krieger Pub., 2003.

[41] S. Maxemow, "That's a drag: The effects of drag forces," Undergraduate Journal of Mathematical Modeling: One + Two, vol. 2, no. 1, 2013.

[42] "ROSplane," GitHub. [Online]. Available: https://github.com/byu-magicc/rosplane. [Accessed: 14-Feb-2021].

[43] M. Quigley et al., "ROS: an open-source Robot Operating System," ICRA Workshop on Open Source Software, 2009.

[44] G. Ellingson and T. McLain, "ROSplane: Fixed-wing autopilot for education and research," 2017 International Conference on Unmanned Aircraft Systems (ICUAS), Miami, FL, USA, 2017, pp. 1503-1507, doi: 10.1109/ICUAS.2017.7991397.

[45] I. Lugo-Cárdenas, G. Flores, S. Salazar and R. Lozano, "Dubins path generation for a fixed wing UAV," 2014 International Conference on Unmanned Aircraft Systems (ICUAS), Orlando, FL, USA, 2014, pp. 339-346, doi: 10.1109/ICUAS.2014.6842272.

[46] R. W. Beard and T. W. McLain, Small Unmanned Aircraft: Theory and Practice. Princeton University Press, 2012.

[47] McLain, T., Beard, R. W., and Owen, M., “Implementing Dubins airplane paths on fixed-wing UAVs”, Handbook of UnmannedAerial Vehicles, 2014, pp. 1677–17

[48] L. E. Kavraki, P. Svestka, J. -. Latombe and M. H. Overmars, ”Probabilistic roadmaps for path planning in high-dimensional configuration spaces,” in IEEE Transactions on Robotics and Automation, vol. 12, no. 4, pp. 566-580, Aug. 1996, doi: 10.1109/70.508439.

[49] J. Jackson, G. Ellingson and T. McLain, "ROSflight: A lightweight, inexpensive MAV research and development tool," 2016 International Conference on Unmanned Aircraft Systems (ICUAS), Arlington, VA, 2016, pp. 758-762, doi: 10.1109/ICUAS.2016.7502584.

[50] N. Koenig and A. Howard, "Design and use paradigms for Gazebo, an open-source multi-robot simulator," 2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) (IEEE Cat. No.04CH37566), Sendai, Japan, 2004, pp. 2149-2154 vol.3, doi: 10.1109/IROS.2004.1389727.

[51] Joseph Needham, Science and Civilization of China, V.3, 1959, Caves Nooks, Taipei, 143.

[52] Lee, D.T., Schachter, B.J. Two algorithms for constructing a Delaunay triangulation. International Journal of Computer and Information Sciences 9, 219–242 (1980). https://doi.org/10.1007/BF00977785

[53] Greenberg, H. (2013, January 26). Mount Saint Helens DEMs. http://gis.ess.washington.edu/data/raster/thirtymeter/mtsthelens/index.html.

[54] Alan V. Oppenheim and Ronald W. Schafer. 2009. Discrete-Time Signal Processing (3rd. ed.). Prentice Hall Press, USA.

[55] Chand, Naresh & McNeill, Kevin Kumar, Srikanta Eteson, Bruce. (2008). An approach to reducing SWAP and cost for avionics high-speed optical data networks. 1 - 7. 10.1109/MILCOM.2008.4753047.