ABSTRACT

An Algorithmic Solution to Mission Planning
via Auto-Routing Algorithms

James L. Furgerson, M.S.E.C.E

Mentor: Michael W. Thompson, Ph.D.

Mission planning for radar jamming escort missions is a tedious and complex problem to solve. For years this type of mission planning has taken many hours to solve and used multiple pilots to develop a solution. This thesis discusses the development of a MATLAB solution for auto-routing aircraft for a mission planning scenario. The unique contribution of this work involves the development and implementation of an auto-router algorithm called the Augmented Mission Planning (or AMP) algorithm. The AMP algorithm is developed by combining techniques for Jamming Acceptability Region (JAR) construction, weighted map creation from DTED data, and an augmented version of the A* path finding algorithm. This auto-router preforms within a typical mission planning system framework and we demonstrate the effectiveness of this approach for determining mission planning in a timely manner.

An Algorithmic Solution to Mission Planning via Auto-Routing Algorithms

by

James L. Furgerson, B.S.E.C.E.

A Thesis

Approved by the Department of Electrical and Computer Engineering

_____

Kwang Y. Lee, Ph.D., Chairperson

Submitted to the Graduate Faculty of
Baylor University in Partial Fulfillment of the
Requirements for the Degree
of

Master of Science in Electrical and Computer Engineering

Approved by the Thesis Committee

_____

Michael W. Thompson, Ph.D., Chairperson

_____

Scott M. Koziol, Ph.D.

_____

Bryon Newberry, Ph.D.

Accepted by the Graduate School

May 2016

_____

J. Larry Lyon, Ph.D., Dean

Page bearing signatures is kept on file in the Graduate School.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGMENTS

I would first like to thank my mentor and thesis committee chair Dr. Mike Thompson for the opportunity and all the support he as given during my graduate studies. Also, I wish to thank Dr. Duren for founding this research and allowing me the opportunity to continue it. I would also like to thank Willis Troy for his friendship and dedication to this research. Finally, thanks to the rest of the thesis committee: Dr. Scott Koziol and Dr. Byron Newberry for their time and interest in this work.

DEDICATION

To my wife and family. None of my accomplishments would be possible without their steadfast love and support.

CHAPTER ONE

Introduction

Mission planning is applied in a wide variety of applications and has been the subject of considerable research [see, for example [5]-[9]]. Auto-routing is an important tool for accomplishing mission planning goals. For example, in a paper by Vasudevan [9], auto-routing is used to plan navigation schemes for autonomous underwater vehicles, where obstacle avoidance is vital to a successful mission. Vasudevan uses case-based reasoning in order to develop new paths from old paths upon discovery of an obstacle. Auto-routing is also quintessential to missions requiring unmanned aerial vehicles, where fuel and airspace are of primary concern. A paper [7] on UAV mission route planning uses common pathfinding algorithms to develop paths for point to point navigation.

Most auto-routing problems involve path planning between a specified starting location and a final destination. The objective of the process is to find an "optimal" path between the starting point and the final destination taking into account possible obstacles and variation in the cost associated with possible routes.

This thesis considers a unique variation from the typical framework of most auto-routing problems. To understand the unique aspects of this problem we first begin by defining important terms. The term "protected entity" (PE) refers to an aircraft that will receive electronic jamming support from the "electronic attack" aircraft. The "electronic attack" (EA) aircraft has radar jamming capability that prevents detection for both the protected entity and itself from threat radars. The goal of an auto-router is to assist

mission planners in finding flight paths for both the PE and the EA that will produce a mission solution. This problem is unique because of the interdependency between the PE and EA paths. Without proper consideration of the PE path by the auto-router it will become problematic to find realistic EA paths. The novelty of the approach used in this thesis is that we incorporate knowledge of EA path constraints into our PE path planning algorithms.

The relationship between the PE and the EA for the problem under consideration is illustrated in the figure below. In this figure the EA is shown as transmitting electromagnetic energy from a strategic location in order prevent the threat radar from accurately determining the location of either aircraft. In this manner, the PE is able to fly paths that would otherwise result in detection. This situation is further complicated by the various jamming techniques available for the EA. The EA can typically provide preemptive or reactive methods for main-lobe, side-lobe and out-of-alignment jamming (6 different modes). A key element in auto-routing is to determine the possible locations of the EA that will facilitate effective jamming. The collection of coordinates that allow the EA to provide effective jamming for a give PE location is known as the Jamming Acceptability Region (JAR).

Auto-routing is an important element of Mission planning. An effective auto-routing tool allows mission planners to quickly investigate scenarios for individual flight paths that possibly comprise a larger mission. The time involved for mission planning is an important consideration. An auto-router tool that generates valid PE and EA paths is important for rapid mission planning.

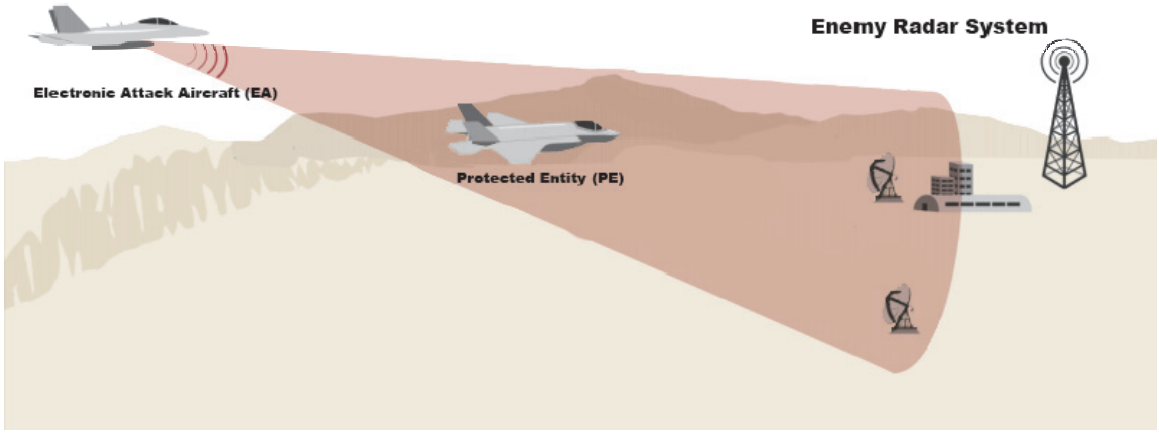


Figure 1: Mission Planning Example

In order to tackle this problem, the research team focused on developing a system in MATLAB to solve the four pillars of this mission planning problem, which are jamming, auto-routing, scenario modeling, and runtime.

Understanding how jamming is achieved was the first primary area of research. A foundation for modeling jamming techniques was developed from a 2008 Navy patent [1] for a system to assess jamming effectiveness. This patent introduced the idea of a Jamming Acceptability Region (JAR) which is an area calculated from the PE position relative to the threat radar. These JARs are used to develop EA paths that will jam the enemy radar for a given PE flight path. Development of JARs can become very complex especially as additional radar threats are introduced. In addition, the complexity of the problem increases to account for the six different types of jamming techniques to apply to any given threat. The active form of these jamming techniques are seen in Figure 2, which shows main-lobe (In), side-lobe (Side) and out-of-alignment (Out) jamming areas.

Chapters Two and Three of the thesis discusses the methods we employ for JAR construction along with strategies for generating PE paths that avoid JAR complications.

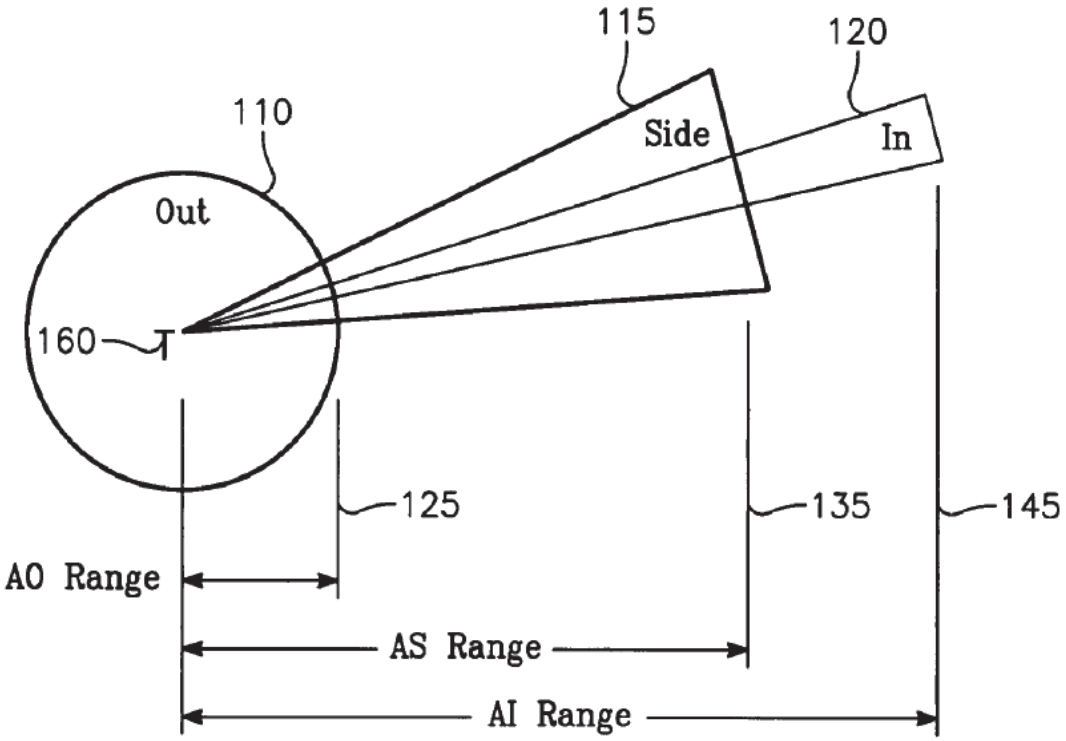


Figure 2: JAR Example [1]

The next pillar of this problem is path generation, or auto-routing, of the PE and EA. The EA path is dependent upon the JARs which are developed from the PE path, thus, the PE path must be generated first. The difficulty lies in creating favorable PE paths that lead to realistic JARs that lead to practical EA paths. This is accomplished in Chapter Three through the use of an algorithm we developed called the Augmented Mission Planning (AMP) algorithm, which is an augmented form of the A* (A Star) pathfinding algorithm..

The next issue in mission planning is accurate scenario modeling. Scenario modeling involves the accurate modeling of a mission in terms of geographic data, radar positions, realistic radar parameters and configurations, and tactical flight information. This area of the problem is difficult to accurately model because the data needed to produce accurate models is classified. However, we take the approach of simulating mission scenarios that reflect many of the challenges that mission planner will face using accurate models. Our approach is illustrated with simulated scenarios which can later be verified using realistic parameters. Scenario modeling is further described in the scenario modeling component section of Chapter Three and is implemented using Digital Terrain Elevation Data (DTED) rendering in a three dimensional MATLAB model.

Finally, runtime is an inherent problem to mission planning. Runtime is a computer science term that refers to the computational intensity of a program or algorithm, which is often measured by the time it takes to execute. Since mission planning is typically a very computationally intensive, auto-routing runtime can be so long that the system is rendered ineffective. Thus, when developing a mission solution algorithmically, a practical runtime is required. This means that all other aspects of mission planning must not only meet the constraints for proper solution generation but also meet runtime specifications in order to deliver practical mission solution. This will be addressed both throughout Chapters Three and Four.

Thesis Outline

Chapter One has laid out the pillars of the mission planning problem which this thesis aims to address. Chapter Two will provide background research related to mission planning. Furthermore, Chapter Two will address the topic of JAR generation and the

resulting necessary modifications required to implement auto-router algorithms. Chapter Three will cover the development of the mission planning system from the ground up, and detail all of its algorithmic components including map weighting, augmentation of the A* algorithm, parallelization, and multi-path generation. Chapter Four will contain the results of the system performance in terms of runtime and the qualitative assessment of the overall viability of the auto-router. Finally, Chapter Five will provide an overview of the work accomplished by this thesis and offer suggestions for future work.

CHAPTER TWO

Related Work

Mission planning for radar jamming escort missions is a rather difficult topic to research. While there are many resources that contribute indirectly to mission planning, (such as pathfinding, parallelization, scenario modeling, and even jamming) there are very few resources within “mission planning” literature that are directly applicable to this particular mission type. The lack of related work in this field is primarily due to the classified nature of this research area. The goal of this Chapter is to summarize existing research in the areas JAR generation and auto-routing. This important to our work since the mission planning software that we have developed incorporates these two components.

The fundamental resource that contributed most to this research was the US Navy patent mentioned in the introduction. This patent defines and describes the creation of a Jamming Acceptability Region (JAR) which, according to the patent is a "family of positions an EA may occupy and still provide effective jamming to protect the PE." [1] Figure 1 shows a general concept JAR taken from the patent, where the arrow is the PE path, the circle is the enemy radar detection range, and the collection of triangular shapes is the JAR.

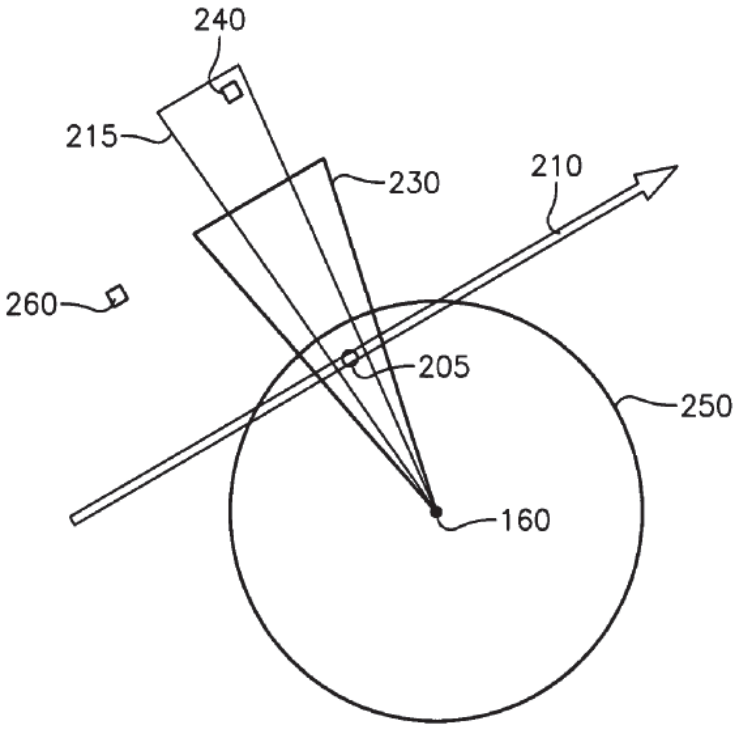


Figure 3: General Concept JAR with PE path [1]

This patent also detailed other scenarios that result in significantly more complex JARs, one of which is a very common circumstance. This scenario is the case of multiple enemy radar (see figure 4). When multiple enemy radar are introduced, JAR creation becomes far more complex and will change depending upon the jamming capabilities of the EA. Figure 4 shows the resulting JARs generated from the PE path, represented as the arrow. The shaded part of the figure is where the two JARs overlap, which can make or break whether a mission is possible or not. This overlap can result in impossible flight paths for a single EA and will require multiple EAs to provide effective jamming. This type of case should be avoided at all costs by the PE due to the new level of complexity for the constrains associated with the EA path. These types of scenarios presented by this patent aided greatly in understanding and developing JARs for our jamming model.

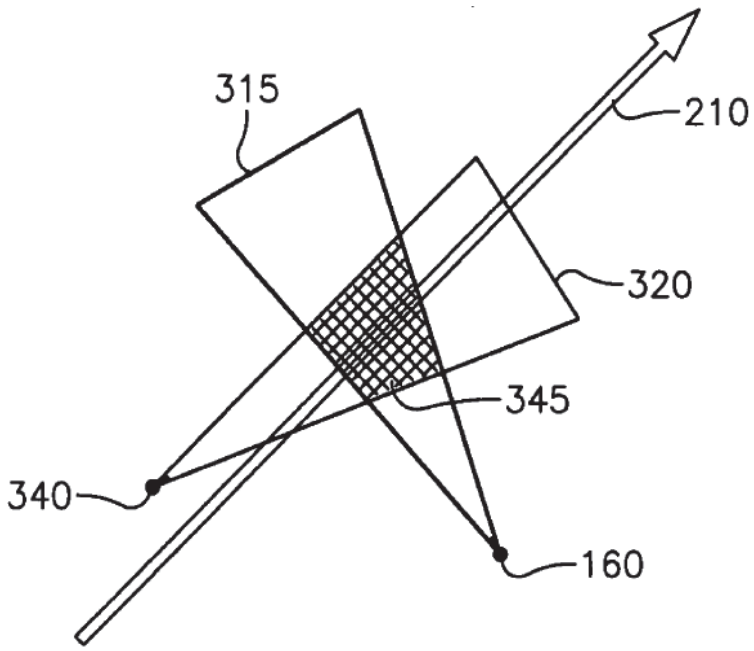


Figure 4: Complex JAR Example

It is also to note that this project's jamming model originated from a paper written by the mission planning team at Baylor [2]. This paper lays out, in great mathematical detail, how to derive a JAR based on radar parameters and the six different types of jamming techniques. These calculations were formed in much greater depth than the original patent, and provide far more accurate JAR generation methods. Figure 5 shows a JAR calculated in three-dimensional space using these methods. The Navy patent and the research by the mission planning team comprise the background to the JAR generation component (Chapter Three) and serve as key resources for this project.

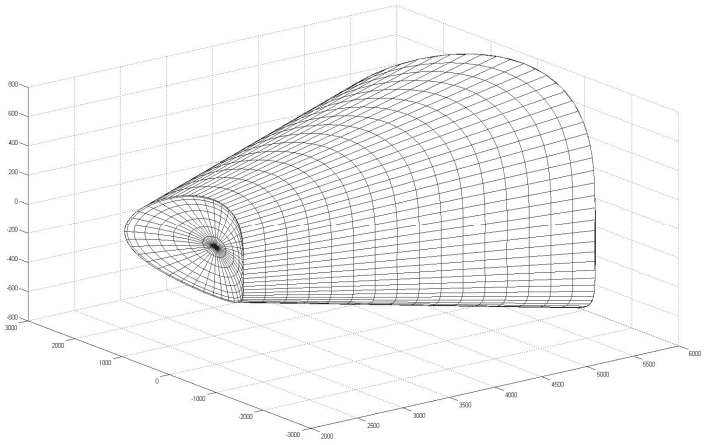


Figure 5: 3D JAR (Ellipsoid Model) [2]

Auto-routing (or path finding) is the process of algorithmically developing a path from a starting point to a desired destination, which is depicted in Figure 6. There are many different algorithms that accomplish this, therefore finding the most suitable one can be a challenging task.

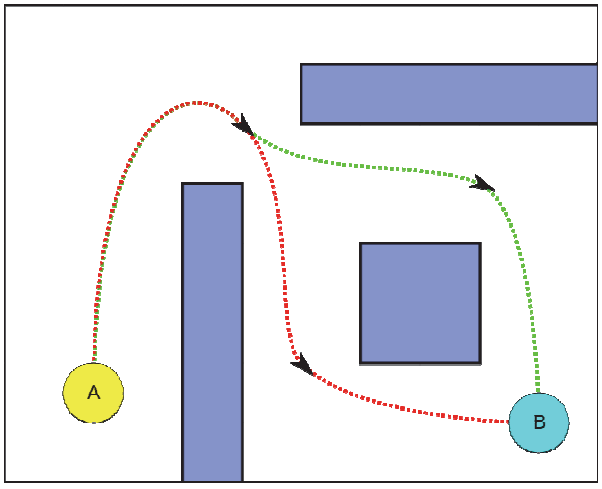


Figure 6: Auto-Routing Example

However, a dissertation on dynamic path planning, written by JP van den Berg [5], explores the use of the A* algorithm [3], as well as Dijkstra's Algorithm [4]. Dijkstra's Algorithm and the A* algorithm are node-based pathfinding algorithms that develop the shortest path from a start point to a target location. The difference between these two algorithms is that the A* algorithm uses a heuristic to guide its search based on Dijkstra's Algorithm [2]. Depending on the problem at hand, one of these algorithms is likely to outperform the other. Van den Berg addresses this and states that "Dijkstra's Algorithm finds shortest paths to all vertices in the graph, but often one is only interested in a shortest path to a specific goal vertex. In this case the A* method, which is based on Dijkstra's Algorithm, is favorable." Therefore, from van den Berg's work and other proponents of these algorithms [7,8], the A* Algorithm was selected to form the basis for this project's auto-routing solution, which is derived in the Algorithmic Foundation section of Chapter Three.

Thus, with the combination of sophisticated JAR creation and the utility of A* and Dijkstra's Algorithm, the foundation of this thesis has been laid. The next chapter uses the basic concepts of JAR generation and the A* algorithm to develop an auto-routing solution.

CHAPTER THREE

System Development

Due to the complexity of a radar jamming escort mission, many algorithmic components are needed in order to generate a mission solution. These components, which are laid out in Figure 7, are structured in order to provide a solution to each pillar of the mission planning problem (jamming, auto-routing, scenario modeling, and runtime). However, a fundamental algorithmic understanding is needed in order to fully explore each component of this system. Thus, in the following sections, the motivation for developing this system, the algorithmic foundation, and each component of the mission planning system will be discussed in detail, where the algorithmic foundation section will explain key building blocks to the auto-router system.

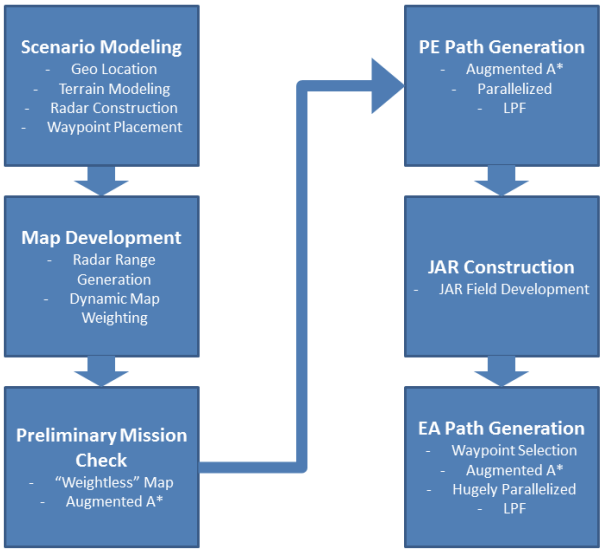


Figure 7: System Overview

Motivation

There are two primary motivations for an algorithmic solution to mission planning. The first is to be able to develop optimal mission solutions, and the results this could have on the field are substantial. For instance, creating optimal mission solutions will save resources due to shorter flight paths and relaxed jamming requirements. In addition, the primary benefit of optimal mission solutions is, of course, to help save lives. By developing better solutions pilots will be exposed to fewer risk factors. The second primary motivation is to develop methods in order to increase runtime of the mission planning process. By improving the auto-routing runtime, faster responses to enemy threats can be made.

Algorithmic Foundation

This section will explain the theoretical understanding and development of the foundation to the mission planning system. This foundation of pathfinding algorithms lead to development of the AMP (Augmented Mission Planning) algorithm, which is key to all the components of the system. The AMP algorithm is the base to the auto-router which drives three of the system components, (Preliminary Mission Check, PE Path Generation, and EA Path Generation) where the other three systems (Scenario Modeling, Map Development, and JAR Construction) set up the environment for the algorithm to run. This algorithm is truly the cornerstone of the mission planning system and will be derived in the following subsections.

Dijkstra's Algorithm

Dijkstra's Algorithm was one of the first pathfinding algorithms, which emerged in 1956. [4] It is a node-based search algorithm, where all nodal weights are calculated adjacently from a source node, in order to develop a weighted tree of costs and determine the least cost path. Dijkstra's will always find the best solution (shortest path), and with some improvements to calculation methods, has runtime complexity of:

$$O(|E| + |V|\log|V|)$$

where V is number of nodes and E is the number of edges [4].

While Dijkstra's Algorithm has been a great way to solve path finding problems in the past, it's use is limited because of the long runtime required for larger maps. Many new pathfinding algorithms have emerged with improved runtimes due to node selection heuristics. However, in worst case scenarios, these new pathfinding algorithms perform the same as Dijkstra's. This is due to the fact that most of the newer algorithms are based off Dijkstra's solution and end up having a runtime upper bound near Dijkstra's Algorithm's runtime.

Even with its limited use today, Dijkstra's Algorithm is fundamental to understanding pathfinding at its core. The concept of a node-based search is the basis for pathfinding and is led to its significant speed up. Also, Dijkstra's algorithm has laid a foundation for development of new pathfinding algorithms. For instance: A*, D*, and theta*, all have roots in Dijkstra's and came to fruition based on Dijkstra's principles. [5] Thus, for these reasons Dijkstra's algorithm was not selected for the basis of the mission planning solution. However, it is through the lens of Dijkstra's algorithm that the area explored for this project, the A* algorithm, was founded.

The A (A Star) Algorithm*

The A* algorithm is a traversing algorithm that determines a least-cost path for a given graph containing obstacles and cost criteria. A* is a derivative of Dijkstra's algorithm where both algorithms use node-based traversing techniques to develop a path from a start location to an end target. The primary difference is that A* uses heuristics in its path development to achieve a significantly faster runtime than Dijkstra's. These heuristics and even the search routine of the A* algorithm can be augmented to create entirely different path planning algorithms to solve more complex problems. Algorithms such as D*, IDA*, and Theta* are a few examples of A* augmentations. [9] [10] In this manner we developed our own A* augmentation to serve a Mission Planning purpose, which will be discussed in a later section.

A pathfinding routine.* As previously stated, the A* algorithm is a derivative of Dijkstra's algorithm in that it is a node-based search routine. However, the point of divergences between the A* Algorithm and Dijkstra's algorithm is in the way A* searches through nodes or points of a graph.

A* begins at the start location. The start location becomes the current node and the distance to all the neighboring nodes are then calculated (D_c). Next, the distance between each neighboring node and the end target location is calculated (D_t). The two distance calculations are summed for each node to achieve the cost (N_{cost}) of any given neighboring node.

$$N_{cost} = D_c + D_t$$

The neighboring nodes and their costs are then placed on an open list. This list acts as the branches the algorithm may pursue. The node with the least cost (N_{cost}) is then selected as the current node and removed from the open list. If the newly selected node is the end target, then the algorithm is completed, otherwise the previous current node is placed on a closed list. The closed list contains all the nodes that have been explored, which is what is used to develop the path.

$$Current\ Node = Open\ List\ Node\ with\ \min(N_{cost})$$

This cycle of calculating neighboring node costs (expanding the open list) and selecting the node with the least cost (expanding the closed list) will continue until the target node is reached. From that point the nodes on the closed list are parsed by cost and ordered to develop the path. However, in the case of an unsolvable graph or where no path exists between the start location and the target, the search will terminate when all searchable points are on the closed list. The entirety of the routine can be observed in the figure below, where the red dots represent the closed list, blue dots represent the open list, dark blue dots represent determined obstacles, and the green line is the generated path.

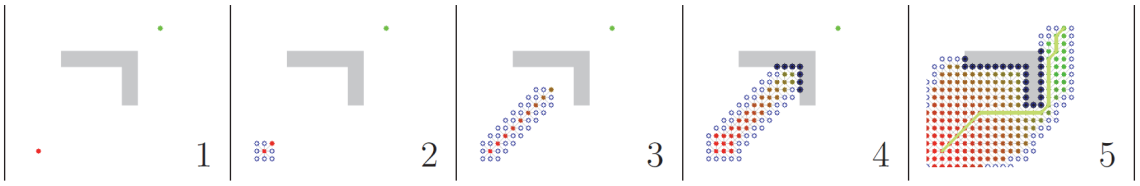


Figure 8: A* Pathfinding Process [13]

*A*algorithm properties.* In order to fully utilize and augment the A* algorithm some fundamentals of the algorithm must first be established. First and foremost, the A* algorithm (if implemented properly) will always find an optimal solution if one exists. Regardless of the graph input into the algorithm, the A* algorithm will find the best (shortest) path to the target. However, as the graph grows in size the run time of A* grows incredibly where, its worst case performance is denoted by the following expression.

$$O(|E|) = O(b^d)$$

$O(|E|)$ is the worst case runtime of the algorithm, d is the length of the shortest path, and b is the branching factor, or the average number of successors per state. Therefore, as the map grows in size and the length of the optimal path grows, the run time grows to the power of d.

Knowing this, how can a graph of an enormous size and complexity, the type that is seen in mission planning, be navigated without an excessive runtime?

The brute force way to solve this problem is to simply increase the processing power of the machine running the algorithm. However, if a super computer is not at your disposal, a better solution may be to augment the algorithm in order to solve a map "well enough". In many problems there is little difference between an optimized solution and a solution that is "good enough". Additionally, if an optimized solution takes three days to calculate and a "good enough" solution takes minutes to calculate, then the "optimized" solution may not be an optimal choice. The next section will cover how our augmented version of A* can be implemented in order to provide a satisficed solution to mission planning. [11]

The AMP Algorithm (Augmented Mission Planning)

The AMP algorithm is an augmentation of the A* algorithm. The AMP algorithm is designed to run on large weighted maps to develop a path that meets many criteria as quickly as possible. Therefore, the algorithm's augmentations from A* focus on speed up, weighted path planning, parallelism, multiple unique paths, path distance, and path smoothing. The following section will break down how the AMP algorithm focuses on these key topics and its performance versus A*.

AMP algorithm: speed up. A fundamental necessity of the AMP algorithm is speed. A* works well for reasonably sized maps and produces an optimal path. However, a map of a larger size causes the run time to dramatically increase. Since mission planning requires navigation of a map with a high resolution and substantial size, changes to the fundamentals of A* had to be made in order to meet run time requirements. This will be accomplished through a tradeoff between an optimized path and runtime. In order to accomplish this trade off, we must recall the node calculations of the A* algorithm.

$$N_{cost} = D_c + D_t$$

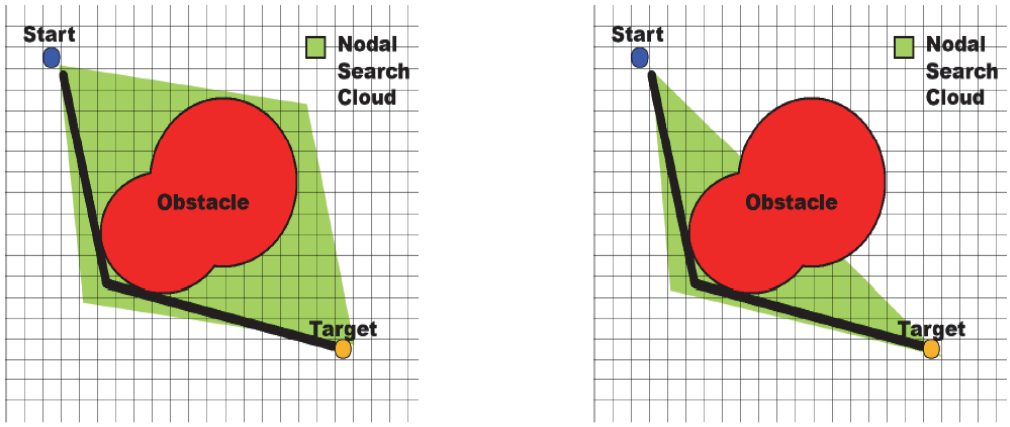
The path generated by A* is dictated by two main costs; D_c, distance to current node, and D_t distance to target. The synergy between these two weights is what helps build an optimal path rather quickly in comparison to Dijkstra's algorithm. However, in the case of A*, the D_c component is what keeps the algorithm in check by avoiding diagonal moves and weighing those against the D_t. However, in the case of a large map, the D_t value is essentially equal for all the nodes that are being evaluated. Thus, on large maps D_t appears to drop out entirely, leaving only a slight pull on where the algorithm's exploration direction, which is now almost entirely governed by the distance to the

surrounding node. This reduces the algorithm to a point near Dijkstra's algorithm, where the algorithm grows around its surrounding area until it happens upon the target. On a gigantic map this type of search can take hours to days depending on processor speeds. The solution is to reduce D_c's impact on nodal selection significantly by dividing it by a large constant C. Then we are left with an algorithm that will always work towards the target with minute difference in D_ts at each node, which gives the following equation. Note, MAP costs will be discussed in sections to follow.

$$N_{cost} = D_t + (M_{cost}) + \frac{D_c}{C}$$

This method produces a much faster way to generate a path, however, reducing the D_c component to such a degree does make a significant trade. A path will be generated much faster but, it is in no way guaranteed that this path will be an optimal path. In fact, a path generated using this method will most likely will not be an optimal path. However, in the field of mission planning, the maps to be traversed fit certain criteria that the optimal path tradeoff is insignificant. This is due to the lack of concave structures and obstacles being primarily elliptical.

The results of this new cost function that guides the AMP algorithm's heuristic can be seen in its nodal search cloud vs A*'s (Figure 9), where a nodal search cloud is the area of the graph the algorithm has explored in order to generate a solution. The more nodes explored results in longer runtime by the algorithm. It is clear that the augmentation explores fewer nodes and thus will require far less time to calculate.



(a) A* (b) AMP Algorithm

Figure 9: A* vs AMP Algorithm

Thus, the AMP algorithm runs significantly faster than it A* counterpart and reduces path generation time of Mission Planning problems significantly.

AMP algorithm: weighed path planning. Weighted path planning refers to planning a path around a graph, where areas of the graph are weighted with a heavier cost. For instance, if area 1 of a graph has a certain cost associated with it, while area 2 has twice the cost of area 1, then directing a path through area 1 would be much more desirable due to its lower cost. However, depending on the complexity of the map weights, the path shaping must be done correctly to generate a path that is desirable from a mission planning perspective. Thus, the second part of the AMP algorithm's cost function, map cost (M_{Cost}), is instantiated. The map cost is simple to understand but difficult to implement. The map cost is simply the cost of choosing a particular location within the map, which is given at the start of the path planning routine, i.e. the cost is built into the map not generated by the AMP algorithm. The map cost is added to the distance to target cost in order to create the total cost of the node. Deciding what values

to place on the map in order to generate a desired path is where the complexity of weights is introduced. This will be further explained in the MAP generation section.

AMP algorithm: parallelism. A significant constraint of the A* algorithm is its sequential form of path planning execution. Each nodal branch is determined by the previous node and so on, where it is impossible to fully parallelize the algorithm. The AMP algorithm is built on the same foundation of sequential execution, however due to specifics of the Mission Planning problem, parallelism can be achieved within solution generation. This is due to the use of waypoints throughout mission planning. Waypoints set certain locations that the path must go through, consisting of targets, start points, and endpoints. By breaking up the path generation into each "leg" (or path between waypoints), multiple instances of path finding can be run in parallel. Each leg is then conjoined to create the final path.

Figure 10 demonstrates this idea. Every "leg" is numbered and can be calculated simultaneously. If no parallelism was introduced, as the way points increase from path 1 to path 3 the run times would increase dramatically. However, calculating these paths in parallel would have a negligible difference in runtimes. Therefore, with the introduction of parallelism into the AMP algorithm, the runtime of path generation sees substantial decrease, even from hours to minutes in some complex cases.

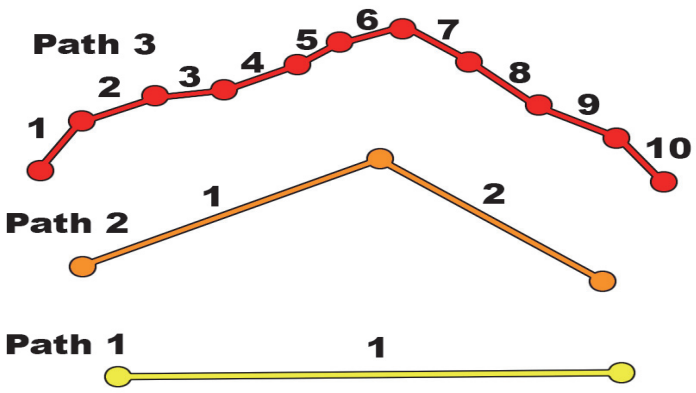


Figure 10: Parallel Example

Thus, for each N_W way points there are $N_W - 1$- instances of the algorithm created, which greatly reduces the AMP algorithm's runtime.

$$N_{PI} = N_W - 1$$

It should be noted that in the cases of very limited solutions or 1 unique solution to a given map, parallelism can cut run time down even more. If an instance of path finding is started at W_1 with and end goal of W_2 and another instance of path finding is started at $W_2$with an end goal of W_1, then the algorithm can connect in the middle to substantially reduce runtime. However, this is not commonly implemented because the cases that will receive a speed up from running the algorithm from both ends is rather specific and unlikely to occur, which is why this method is not incorporated within the AMP algorithm.

Therefore, through the use of waypoints, parallelism allows for a dramatic decrease in runtime for path generation. This specific quality of the AMP algorithm will be very useful in creation of both the PE and EA path. However, the EA path would be

almost non-calculable without this parallelism. These properties will be further explained within the PE and EA path generation sections.

Multiple unique paths. Another focus of the AMP algorithm is developing multiple unique paths. The reason for this is to avoid predictable paths in cases of the same mission being run multiple times. To accomplish multi-path generation, both map augmentation and path generation are required. Once a successful path is created, that path is then weighted and another iteration of the AMP algorithm is run. Therefore, the new weights change how the AMP algorithm navigates through the map and creates a new path. This process can continue for as long as desired. For each way point another iteration of path finding is instantiated and another branch is created, which has a serious effect on the number of unique paths that are generated. This effect is represented in the equation below where, P_{num} is the number of unique paths, I is the number of iterations run, and W is the number of way points.

$$P_{num} = I^{W-1}$$

For example, if there is a map with 4 way points that is run with 10 iterations of multi-path development, the output is one thousand unique paths. This is illustrated by figure 11. This process will be further expounded upon in the PE path generation section to discuss specifics of map augmentation and runtime.

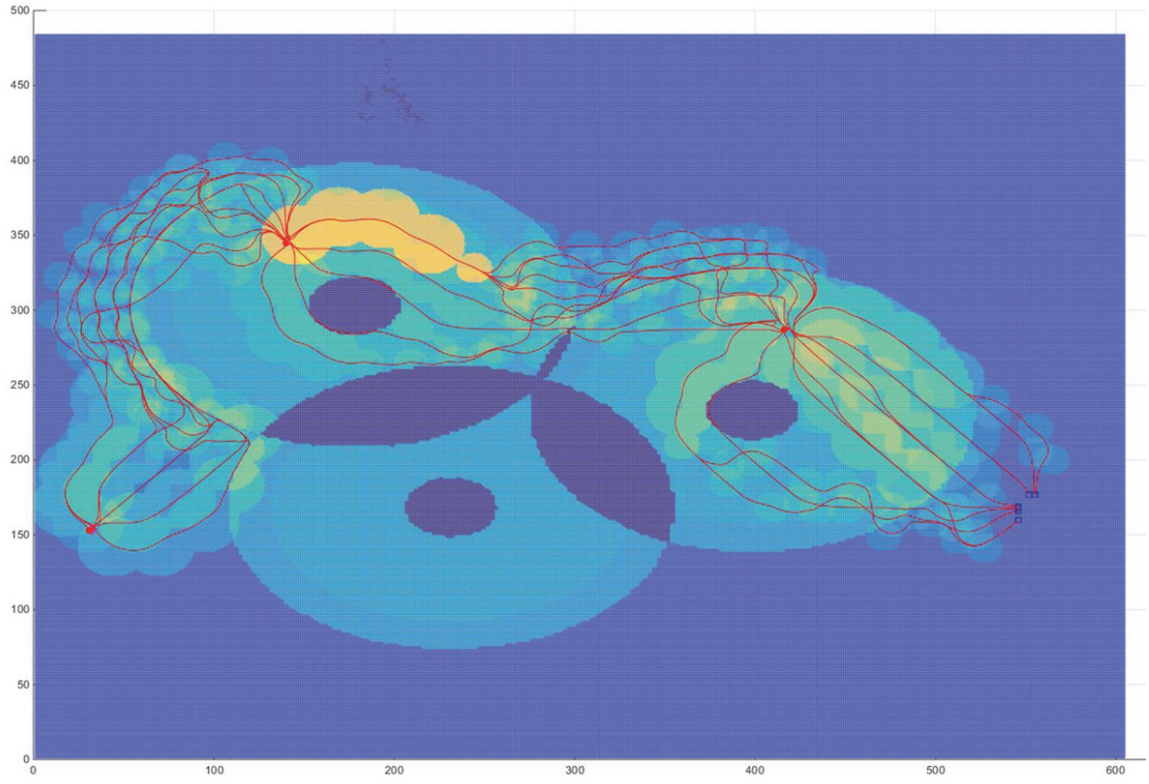


Figure 11: Multi-path AMP Output (1000 Unique Paths)

The Mission Planning System

The following sections will detail the focus of this thesis; the mission planning system. Each section will describe the algorithmic components that make up the system and how they function together to form a mission solution.

Scenario Modeling

As stated in the introduction, scenario modeling is a rather difficult area to research. Battle scenarios are a highly classified area and little information is available to the public. However, through the use of DTED (Digital Terrain Elevation Data), geolocation coordinates, and enemy radar range construction, successful scenario modeling can take place with the future intent of improvement.

DTED is a great solution to landscaping in scenario modeling. DTED is essentially a data base that consists of elevation values over the entire surface of the earth. Thus, any area of the earth's surface can be loaded and used to generate a landscape. The accuracy of these values changes with the different levels of DTED (see Table 1).

Table 1: DTED Level Resolution

DTED Level	Resolution (m)
0	~ 900
1	~ 90
2	~ 30

Only certain levels of DTED are available to the public, however the government has higher levels of DTED which they use internally. Therefore, by building the scenario modeling component upon a standard that can be easily modified for resolution improvement, the system can be easily implemented by the Navy.

Fortunately, DTED can be easily imported by MATLAB through its' own importer. With this tool in hand the landscape can begin to take form in order to provide robust scenario modeling of the mission environment. Gathering the data for a given mission is accomplished through geolocation coordinates. By traversing the DTED data base via latitude and longitude, the desired surface of the earth can be loaded and modeled. Figure 12 shows DTED of a section of the California coastline using MATLAB's functionality.

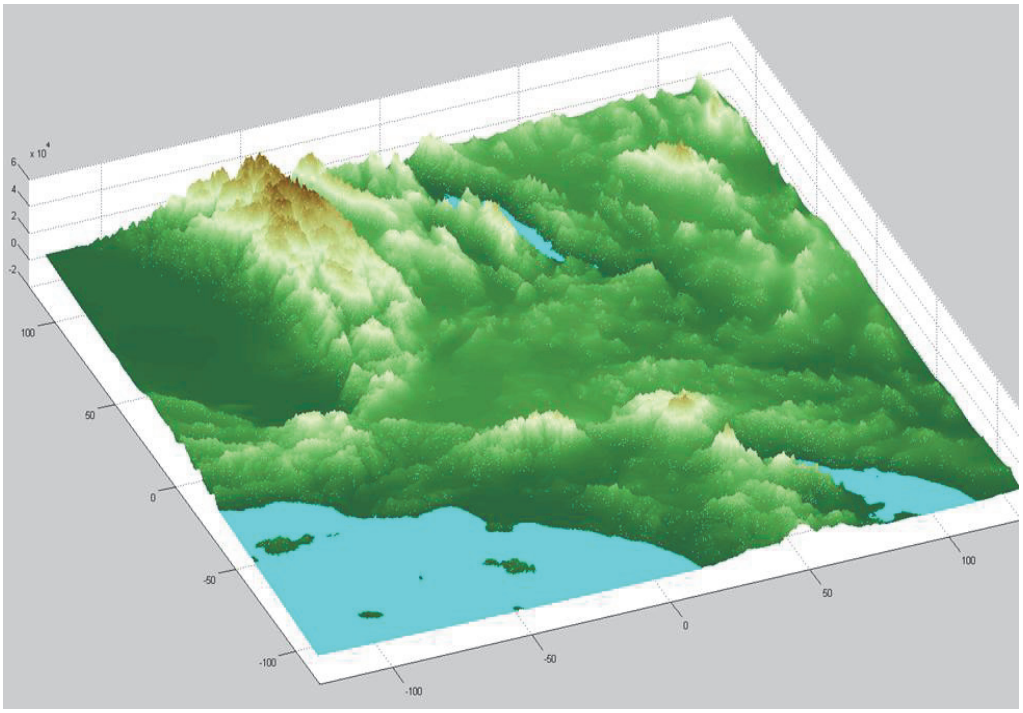


Figure 12: DTED Levels

Enemy radar construction is yet another area where one is hard pressed to find information. While it is difficult for the academic world to generate enemy radar ranges and radiation patterns, the Navy has the tools to do so. Hence, generation of range and radiation patterns are not a goal of this thesis. However, one characteristic of enemy radar range that is able to be modeled is the "lethal zone" [2]. This zone is an area around the radar where the EA's jamming for the PE is no longer effective. This produces concentric circle ranges, where one is the enemy radar range and the other is the lethal zone (see Figure 13). Incorporating the lethal zone into the scenario modeling process is absolutely key. Since these cannot be derived, (due to lack of public information) an estimate will serve as a good placeholder until more accurate information can be input.

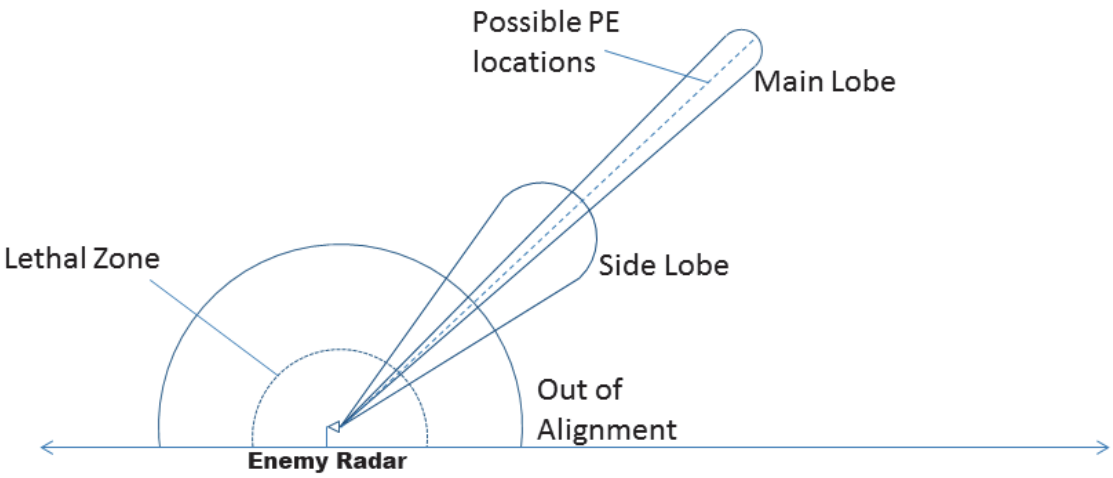


Figure 13: Radar Range and Lethal Zone

Nevertheless, enemy radar placement within a mission is an aspect of scenario modeling that can be explored. Typically, radar jammer escort missions are performed in an area of high detectability. This means that multiple enemy radar are on-site and their ranges overlap. Also, placement of enemy radar tend to be either clumped together in order to protect a particular location, or spread along a line in order to protect a border or coastline. With these characteristics in mind, better scenario modeling can be accomplished by simply placing enemy radar locations in the same type of patterns and allowing radar ranges to overlap.

The final aspect of scenario modeling concerning this system is waypoint positioning. A waypoint is a location that must be reached throughout the course of a mission. Typically a flight path for a mission is divided up into sequential waypoints that are used as references to orchestrate a mission, where each waypoint dictates where the EA/PE should be at a given time. Since waypoint selection is obviously classified information, development was focused on evaluating waypoints at varying levels within

enemy radar range. This was done in order to assure that the system performed properly with any mission, regardless of waypoint placement.

Map Generation

Map generation is the process of using all the scenario modeling aspects and incorporating them into a single environment that is primed for the AMP algorithm: the map. The scenario modeling component's output, landscape, enemy radar range and location, and waypoint positioning, are used to develop a matrix of weighted values that the AMP algorithm must navigate. These weights are crucial to the solution of the mission. This is because the map weights will shape how the AMP algorithm solves the mission and, in turn, affects the optimization of the end result. The following sections will examine how the map is created from the scenario modeling output and how the map is weighted to facilitate a desirable result.

The scenario modeling component's output is used to set the foundation of the map. Firstly, the latitude and longitude, as well as the landscape, generated by the DTED, is used to set the bounds of the map. These bounds are the map size, or the area in which a mission will take place. They are used to create a matrix of the same size as the points contained within the DTED over the desired longitude/ latitude area. It should be noted that the resolution of the map matches the resolution of the DTED, meaning that the level of DTED sets the resolution of the MAP. From there, the radar locations and waypoints are placed on the map by the longitude, latitude, and altitude desired. The ranges of the radar are then added to the map by calculating them in three-dimensional space. Again, enemy radar range calculation is not an area for public research and thus not an area of research for this thesis. However, a shell model (seen in Figure 14) can adequately

represent a radar range and be substituted with proper values in classified environments at a later date.

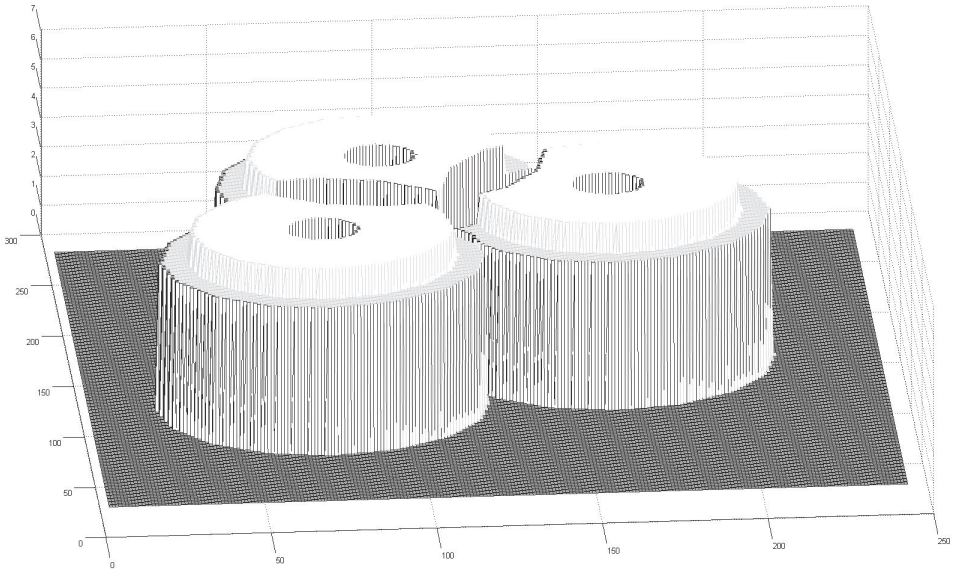


Figure 14: Enemy Radar Range

The cross-section of the landscape data and three dimensional radar range are then taken at the desired altitude of the mission, which is the same altitude as the waypoints. This cross-section, composed of all the scenario modeling aspects, is what makes up the weighted map, where the only areas that have values at this point are the lethal zones associated with the enemy radar, and any landscaping obstacles, such as mountains. These are given negative values on the map and are considered as obstacles to the AMP algorithm. This rather barren map is the weightless map that is used by the mission check system component, which will be explained in the next section. The final step in map creation is to weight the enemy radar range and free space in order to facilitate the AMP algorithm. Free space receives a base value of 1 and the enemy radar range is a multiple of the free space weight. In order to facilitate speed and paths that do not fly into enemy

radar range to save distance, a high value was assigned to the enemy radar range weight of 10 times that of free space. This value makes penetrating into enemy radar range a very high cost and forces the AMP algorithm to only enter enemy radar range when a waypoint is contained within the enemy radar range. A fully realized map is show in Figure 15 in graphical form

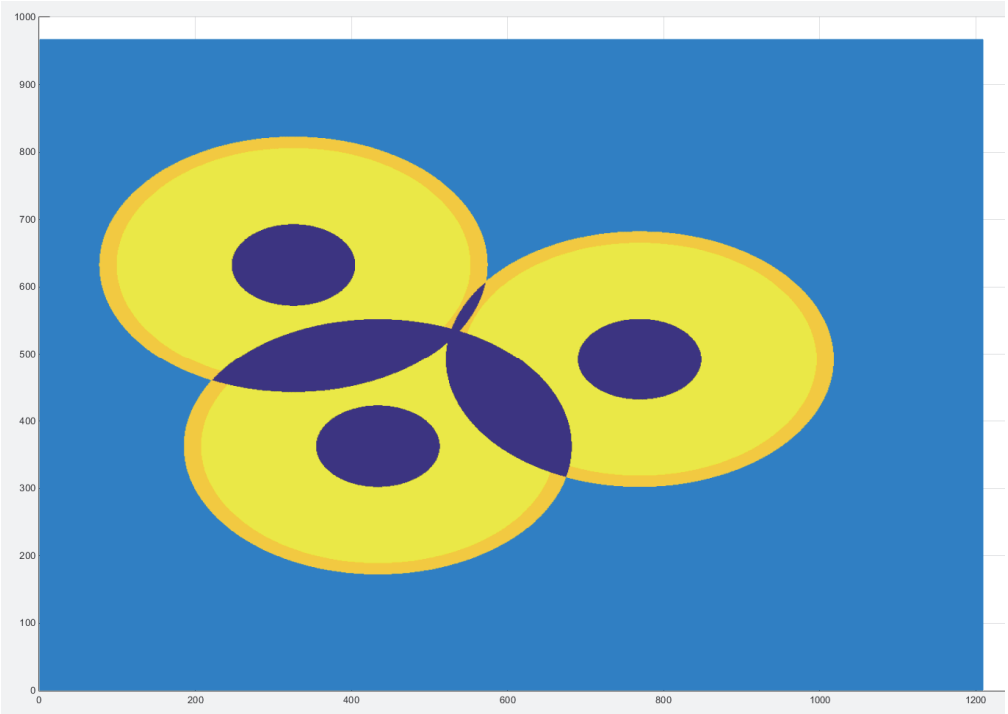


Figure 15: Map Example

Preliminary Mission Check

The preliminary mission check component of the mission planning system is very simple. The preliminary mission check's job is to determine whether or not the mission can be solved. This provides a great service to the system because this process is fast, and in the case of an unsolvable mission, provides a massive runtime reduction. By

predetermining mission possibility, the algorithm can move into the next phase of mission planning, confident that a solution exists and no time will be wasted.

The mission check is performed by running the AMP algorithm on a weightless map, where only the waypoints and obstacles, or lethal zones, are kept. Because these obstacles are the only absolute limitation of the flight path, the obstacles and waypoints are the sole aspects of the map that must be incorporated to check mission possibility. Figure 16 show a mission check in action and the weightless map that it performs on.

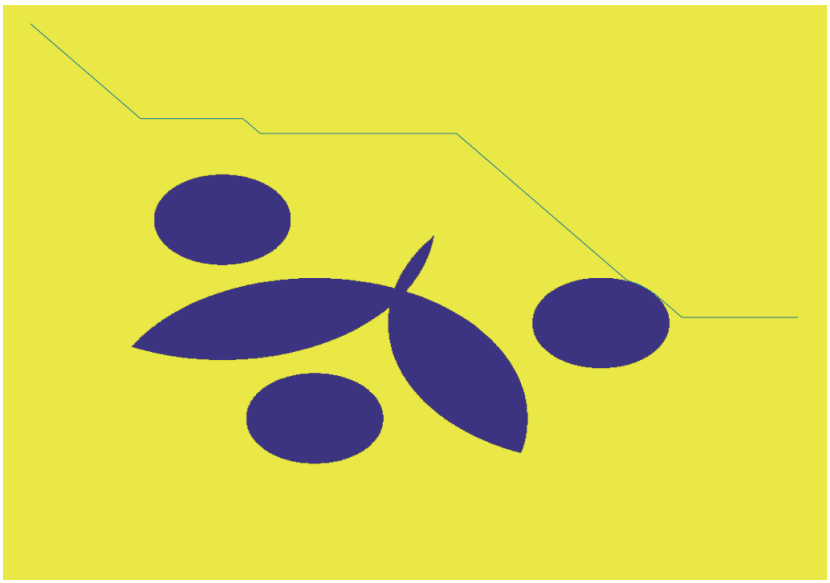


Figure 16: Mission Check on Weightless Map

As one would expect, with the removal of the weights, a solution can be generated in a fraction of the time it would take on a weighted map. A quick binary digit will represent whether the mission is solvable or not. If the mission check is successful the algorithm will move on to create the PE path, but if the mission check fails the mission planning system will terminate.

PE path Generation

Thus, with the weighted map (incorporating the scenario modeling data) in place, and an affirmative mission check, the mission planning system will now begin to calculate the PE path. This will be accomplished with the AMP algorithm that will, in parallel, calculate each leg between all waypoints. Algorithmic details of this process can be found in the Algorithmic Foundation section in Chapter 3. The results of this process can be seen in the PE path generated for four waypoints in Figure 17.

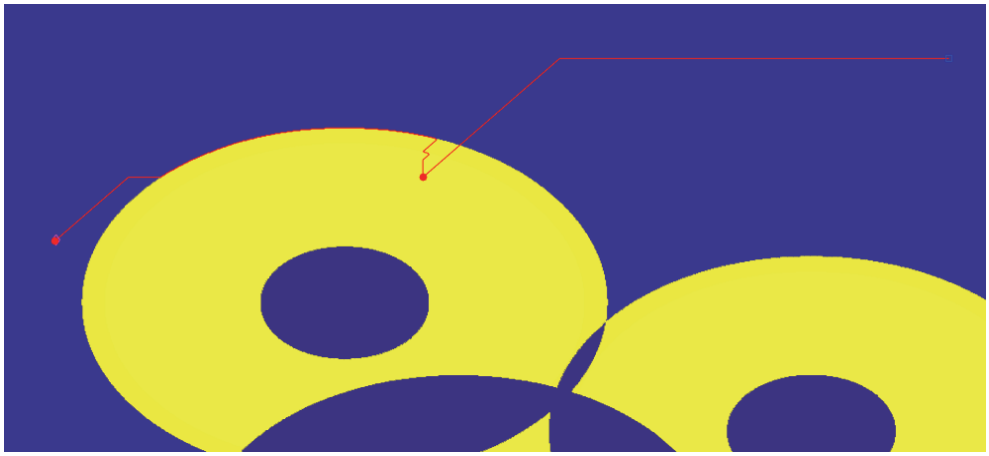


Figure 17: PE Path w/ No Post-Processing

While the PE path generated is an adequate solution and meets runtime requirements, there is a slight problem that requires some attention. The PE path that is generated is rather jagged. This is due to the type of search that the AMP algorithm uses. Because the algorithm is a node-based search, the path between each point on the map is a straight line, which makes curves appear as stair steps. This is a problem for two reasons. The first is that this jagged path does not accurately model a plane's flight path in the slightest. If this path is to be designed as a solution and not an approximation, it must accurately depict a real flight path. The second problem stems from JAR creation. If the

JARs for the EA flight path were created from this jagged stair step path, the result would be an incredibly inaccurate mess that would totally break the system. JAR creation will be further discussed in a later section, but it is important to note the effect the PE path has on the resulting JARs.

The solution to this jagged stair stepped path is the simple post processing technique of low-pass filtering. In this case low-pass filtering is implemented though a weighted averaging (α) of the current coordinate (x_i) with the previous (x_{i-1}), which is seen in the equation below.

$$LPF(x_i) = \alpha x_{i-1} + (1-\alpha)x_i$$

The weight (α) determines the operation of the filter. In the case of the stair step curves seen in this mission system setting α relatively high (around 0.9) aided in correcting the path greatly. Figure 18, shows the filter operating on a stair stepped sinusoid. The result is a smooth sinusoid with a small amount of loss.

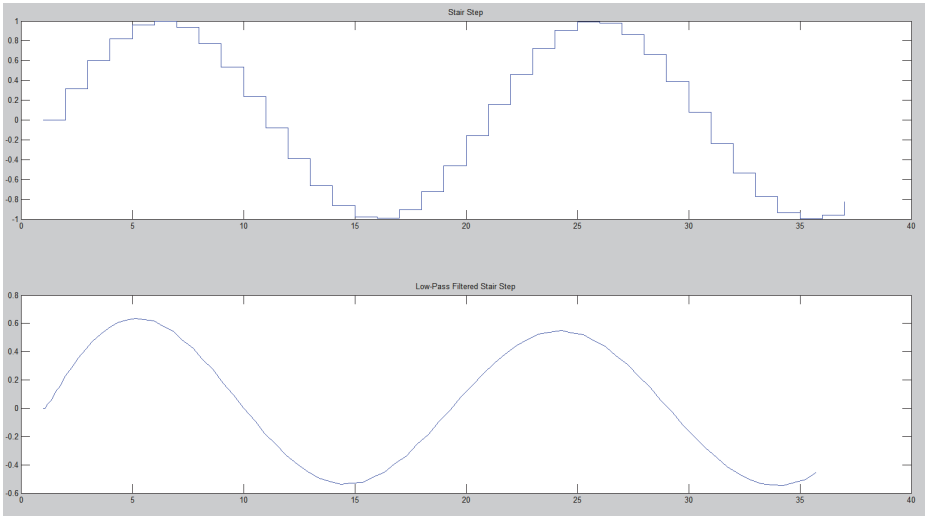


Figure 18: LPF Applied to Stairstep Function

Applying this same low-pass filtering technique to the PE path achieves far more desirable results (See Figure 19). With these changes, the end result for the PE path meets both requirements of accurate simulation of a flight path and the necessary consistency for JAR creation.

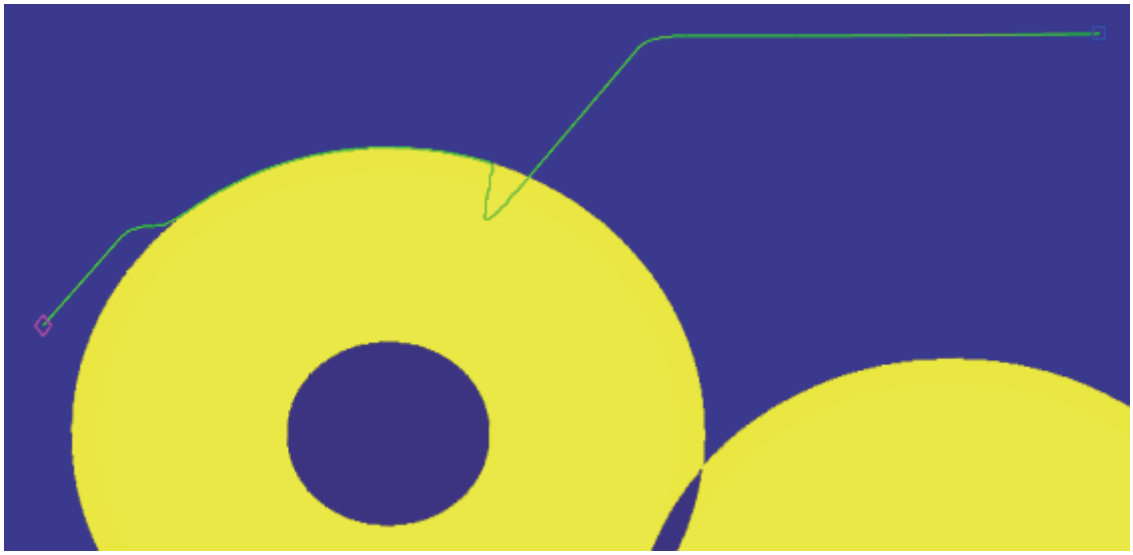


Figure 19: PE Path with LPF

The final aspect of PE path generation is generation of multiple unique paths. As previously stated, this process is achieved through augmenting map weights after multiple iteration of the AMP algorithm. The weights are distributed along the path in large circles that condense to a fixed smaller size as the path continues (Figure 20)

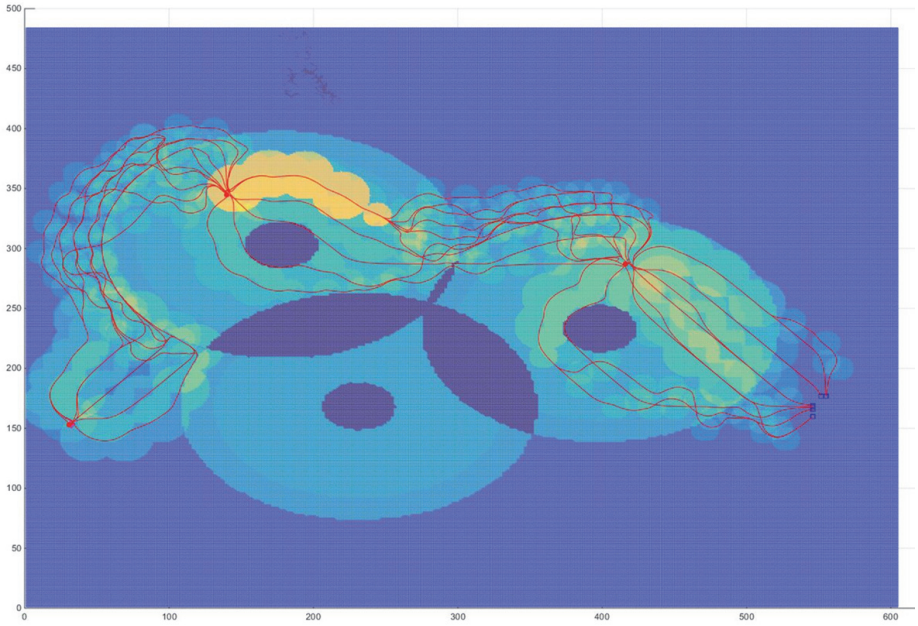


Figure 20: Condensing Circles for Multi-path

There are three reasons as to why condensing circles were chosen to weight a given path. The first is to influence the path to deviate at the start, which is achieved by the large circles at the beginning of a path. By applying a weight over a larger area at the first part of the path, the AMP algorithm will be forced to maneuver around or through the weighted zone, which will cause a new path to be created. The second reason condensing circles were chosen was to allow a new path to cross the old path if necessary, without the potential of the new path converging to the old path. The small gaps between the weighted circles allow this to take place. Because that area has no additional weight from the previous path, the new path is not discouraged to cross in that gap area. The final reason condensing circles were chosen is due to the fact that they provide nice curvature and overlap. The curvature of the circles provide smooth transitions for the path to take, while polygonal shapes would have sharp edges to traverse. Additionally, the overlap between circular map weights apply well. When the

weights begin to overlap from multiple runs of multi-pathing, the combined weights form fairly smooth overlapping areas that will keep the generated path from making abrupt changes in direction. All three of the reason for using condensing circles can be seen in Figure 21, which shows both the paths generated and the weights applied by condensing circles. In this figure the dark blue areas are lethal zones and weighted areas' magnitudes are equivalent to the brightness of color.

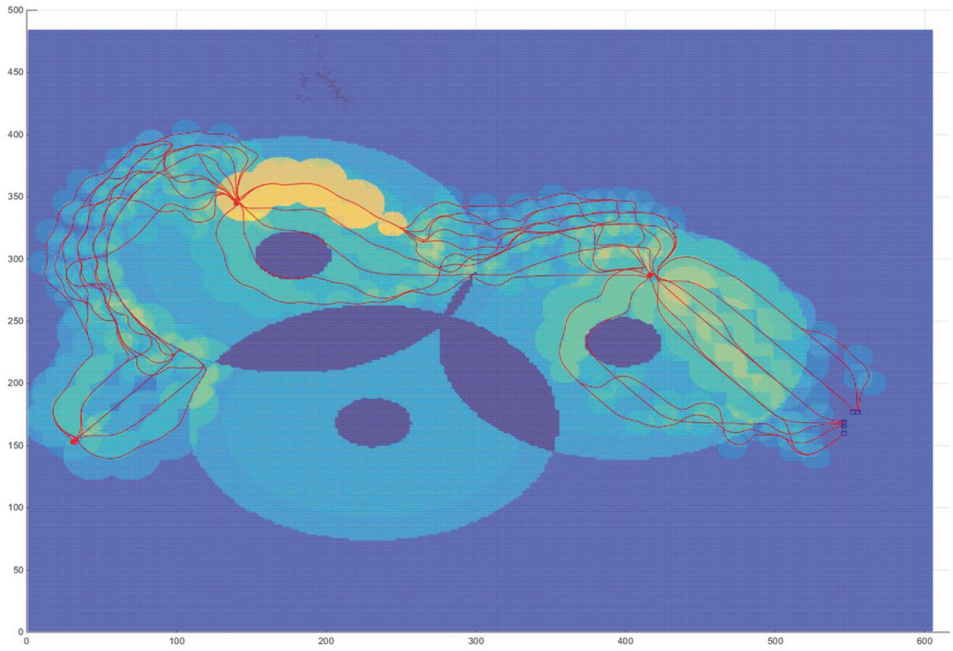


Figure 21: Multi-Path Development (1000 Unique Paths)

The PE path can be generated via the AMP algorithm and post processed with a low-pass filter. The system's next priority is to calculate JARs for EA path development.

JAR Construction

JAR construction is a complex area of research. There are many parameters from both the enemy radar and EA that greatly affect the resulting JAR. However, the paper by the mission planning team at Baylor on JAR generation [2] derives how to fully develop JARs given the enemy radar and EA parameters. The area of this paper that is of particular interest is how JAR generation is accomplished at a particular altitude via intersection using three different jamming techniques: preemptive main lobe, side lobe, and out of alignment jamming. This is achieved by modeling the JARs as geometric intersections of the detection radar pattern with the altitude plane. This paper derives this process of JAR generation in great detail and is the same method used in this system. Figure 22 shows the JARs calculated for a given PE path using the above JAR generation method.

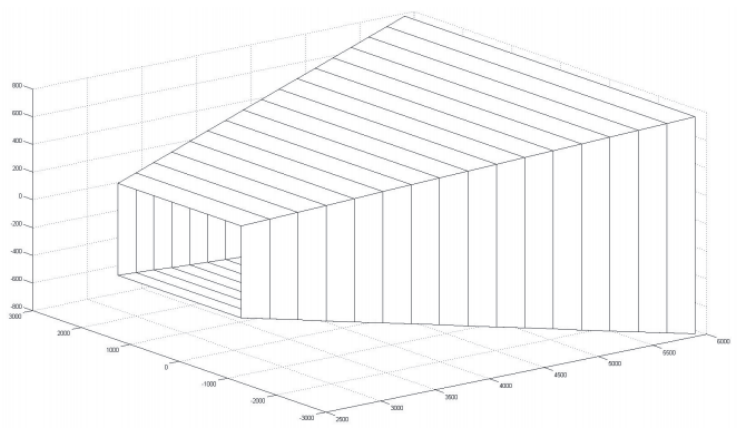


Figure 22: JAR Generation from Geometric Intersection (Rectangular Model)

EA Path Generation

Finally, with JAR construction completed EA path generation, the last phase of the mission planning system can take place. On the surface EA path generation is very

similar to PE path generation. They both use a parallelized instance of the AMP algorithm and both navigate waypoints across a map. In addition, they also require post processing via low-pass filtering. However, the difference is that EA path generation has hundreds, if not thousands, of waypoints, where these waypoints are selected from hundreds, if not thousands, of JARs.

As previously mentioned, each point on the selected PE path that is within the enemy radar range has its own JAR associated with it. With higher resolution paths and longer lengths of paths within enemy radar range, more JARs will be created. For each JAR created a waypoint must be selected within the JAR in order for the EA to adequately jam for the PE. Waypoint selection is done simply by finding the point within a JAR that is the closest to the current EA waypoint and the next PE path location. This is accomplished by calculating the sum of the distance between the current JAR location, the current EA waypoint, and the next PE path location. The EA waypoint selection equation below illustrates this relationship, where W_n is the waypoint to be selected, $J_{W_{n-1}}$ is the vector of distances from the JAR to the previous waypoint, and J_{PE+1} is the vector of distances from the JAR to the next PE point

$$W_n = \min(J_{W_{n-1}} + J_{PE+1})$$

The location with the minimum value will be the desired waypoint. This process is run on all JARs and creates a constellation of waypoints for the AMP algorithm to solve. Figure 23 shows the EA waypoint constellation on (left) and the EA path generated (right, black plot). Note that the shape of the constellation of waypoints resembles the EA flight path shown with the black plot. The figure on the left is enlarged so that waypoint JARs can be seen.

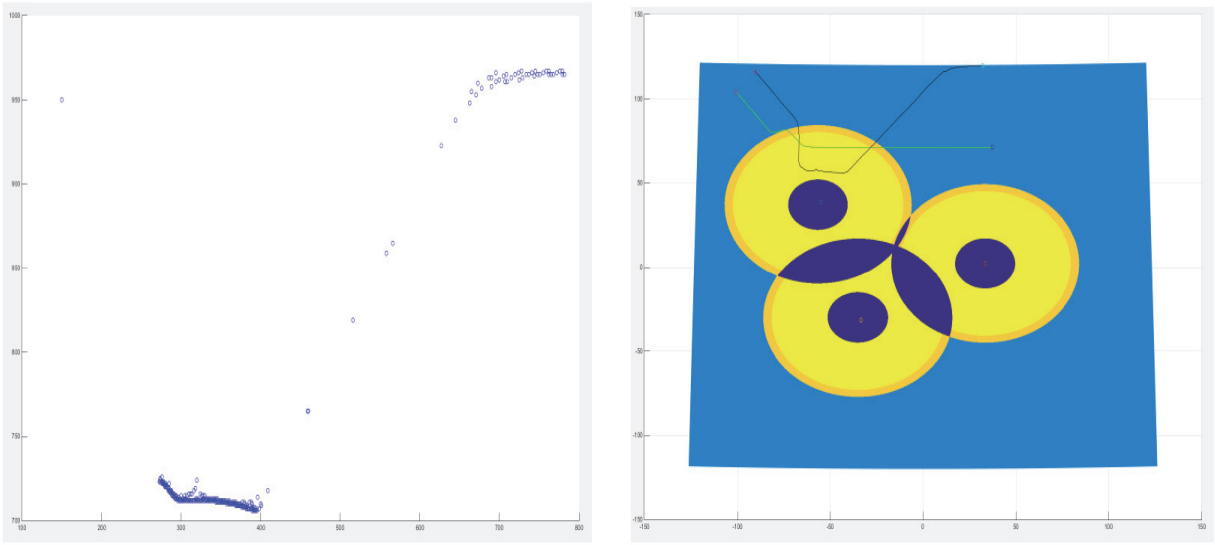


Figure 23: EA Waypoint Constellation and EA Path

At this point, massive parallelization will take place, because there are hundreds to thousands of waypoints to navigate. While this may seem like a daunting task, it does not take substantial run time to execute using a standard computer system. This is due to the fact that many of the legs are only a small distance apart and require little-to-no pathfinding. However, the reason the AMP algorithm must be run for each leg of the constellation is because the distances of the waypoints are not guaranteed and require little-to-no-path finding. For instance, when a PE path passes in and out of multiple enemy radar ranges, the resulting JARs could be generated far apart. For this reason the AMP algorithm is still required for each leg because a lethal zone or another enemy radar may be between two EA waypoints.

With the navigation of the EA waypoint constellation and low-pass filtering of the EA path, the solution to a given mission is completed. Both the PE path and EA path have been generated and meet all the waypoint and jamming requirements. The next

chapter will focus on the results of each component of the mission planning system and the overall results of the system as a whole.

CHAPTER FOUR

Results

This chapter will discuss the results of each component of the mission planning system and the results of the mission planning system as a whole. The goals for the system are to provide a solution to the pillars of the mission planning problem, which are effective jamming, proper auto-routing, scenario modeling, and runtime. Each component will be measured by its effectiveness within these four pillars in order to determine the overall results of the system and individual components. An emphasis will be place on the runtime results of each component due to the importance of runtime within mission planning. Please note that all of the following results were taken from a 1000x1300nmi mission size.

Scenario Modeling Results

The scenario modeling component of the system performed well in terms of both accuracy and runtime due to the nature of DTED. Because DTED is a commonly used standard with high levels of accuracy, the resulting terrain reflected is accurate. Also, since higher levels of DTED are used internally by the government, the system can easily facilitate higher resolution landscapes in their hands. Another property of DTED that lead to a successful system component is its size. DTED can be loaded quickly and does not require massive processing. The specifics of this system's performance are seen in Figure 24, which displays the DTED rendering of the mission as well as a runtime of 2.195s for the scenario modeling component. It is clear that the system executes quickly and the

terrain generated is accurate to the level of DTED. However, higher levels of DTED would allow for this system to achieve greater resolution and improve the system accuracy as a whole. Having access to these levels would further the goals of this work. However, because these levels are classified the current DTED available will suffice.

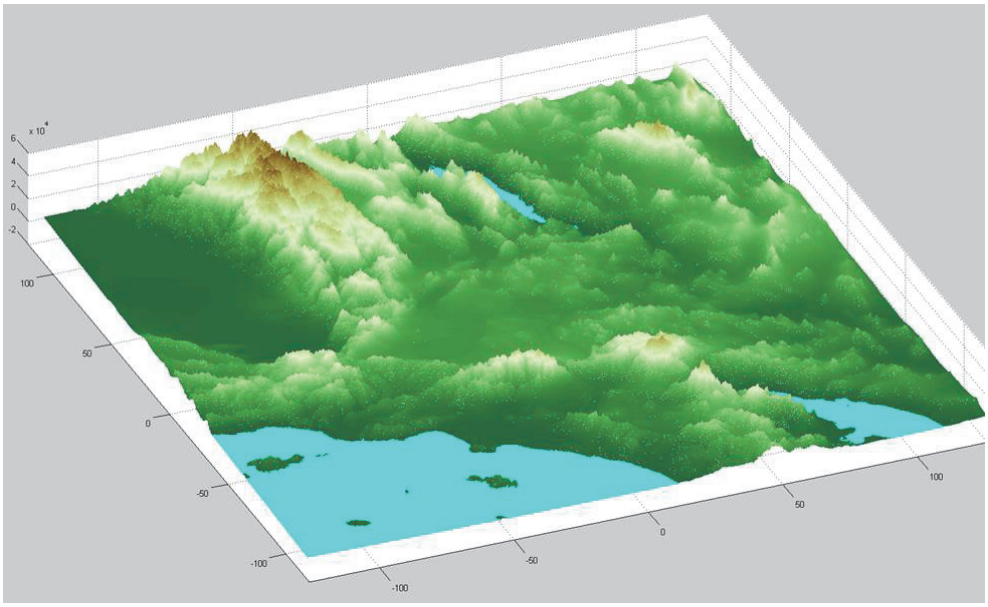


Figure 24: Scenario Modeling Result, Runtime = 2.195s

Map Generation Results

The primary purpose of map generation is to create an environment for the AMP algorithm that will facilitate a desirable path. This was accomplished through combining the scenario modeling component's output with the predefined threat radar longitude and latitude coordinates to form a matrix with properly weighted values. The map generation component is vital to auto-routing and the weights set by this process greatly affect the resulting path. This system component performed quite well as it generates the desired weights for the range of the map and has a short runtime even on large maps. The results

of the map generation component can be seen in figure 25, where this component had a runtime of 3.298s.

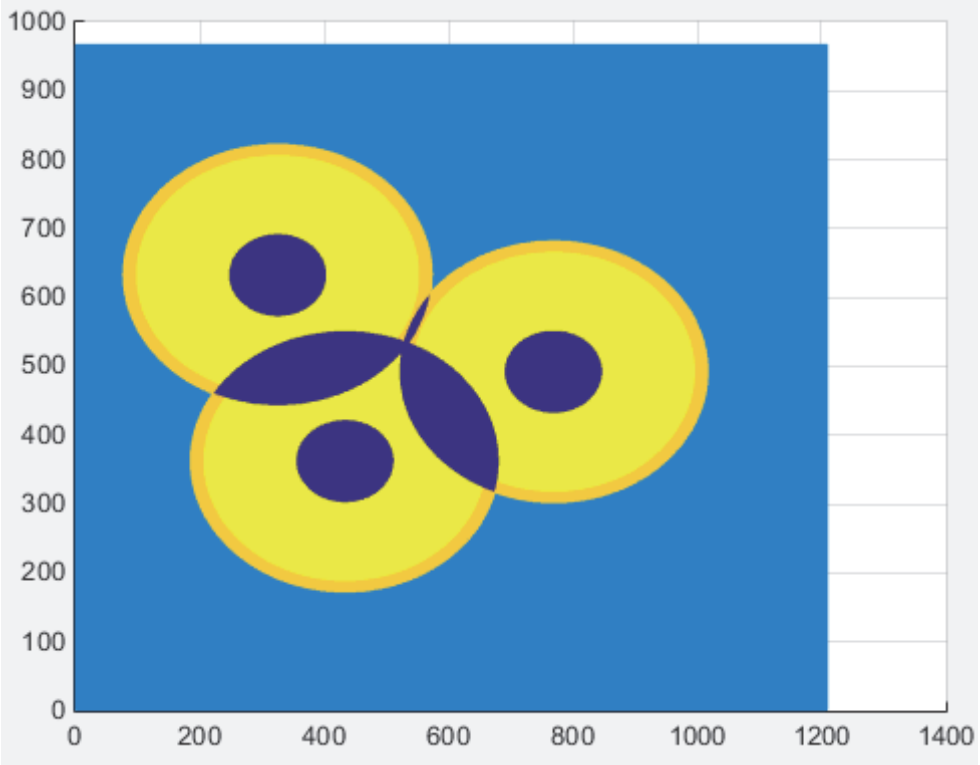


Figure 25: Map Generation Result, Runtime = 3.298s

Preliminary Mission Check Results

The preliminary mission check is a simple implementation of the AMP algorithm upon a weightless map. Because this process requires navigation of a large weightless map, runtime for the mission check is not negligible (in this case about 15.4832s). However, the check does run expediently enough to still be beneficial to the system. By spending a small portion of time on the mission check, hours of calculation can be avoided by predetermining if a mission is not solvable. Figure 26 shows the results of the mission check and the runtime required.

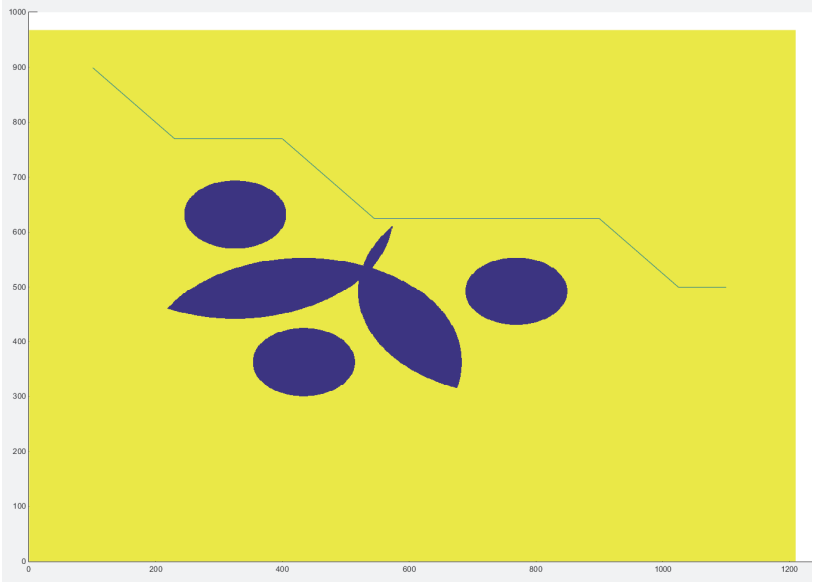


Figure 26: Preliminary Mission Check Result, Runtime = 15.4832s

PE Path Generation Results

PE path generation is one of the more computationally intensive components of this system. This is due to the complex nature of navigating a large weighted map between multiple waypoints. The longer runtime is partially mitigated by the parallelism of the AMP algorithm, however, runtime is still in the order of minutes for a single path. In the case of multi-path development, PE path generation can be cumbersome depending on timing requirements and the number of desired paths. Figures 27 and 28 shows the results for single and multi-path PE generation, where single-path generation had a runtime of 22.840s and multi-path had a runtime of about 4 hours. While these results clearly point to PE path generation as one of the longest run times in the system, the process is still more expedient in comparison to the current state of many other mission planning systems.

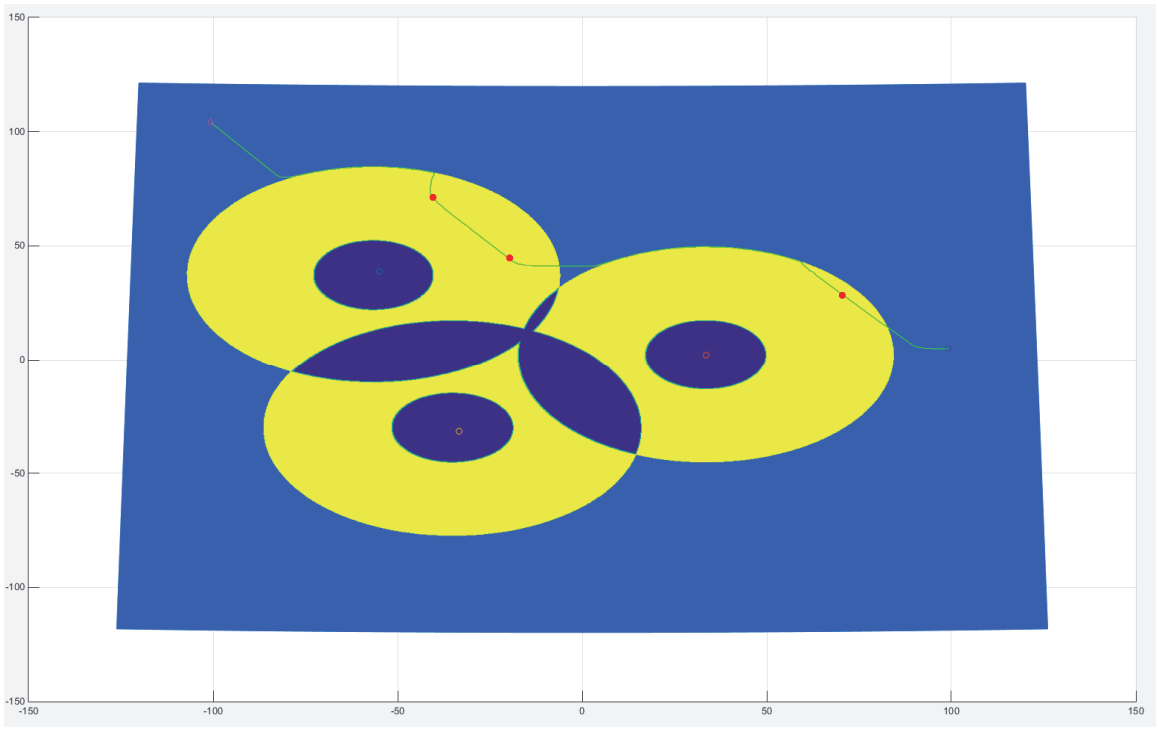


Figure 27: PE Path Generation Result, Runtime = 22.840s

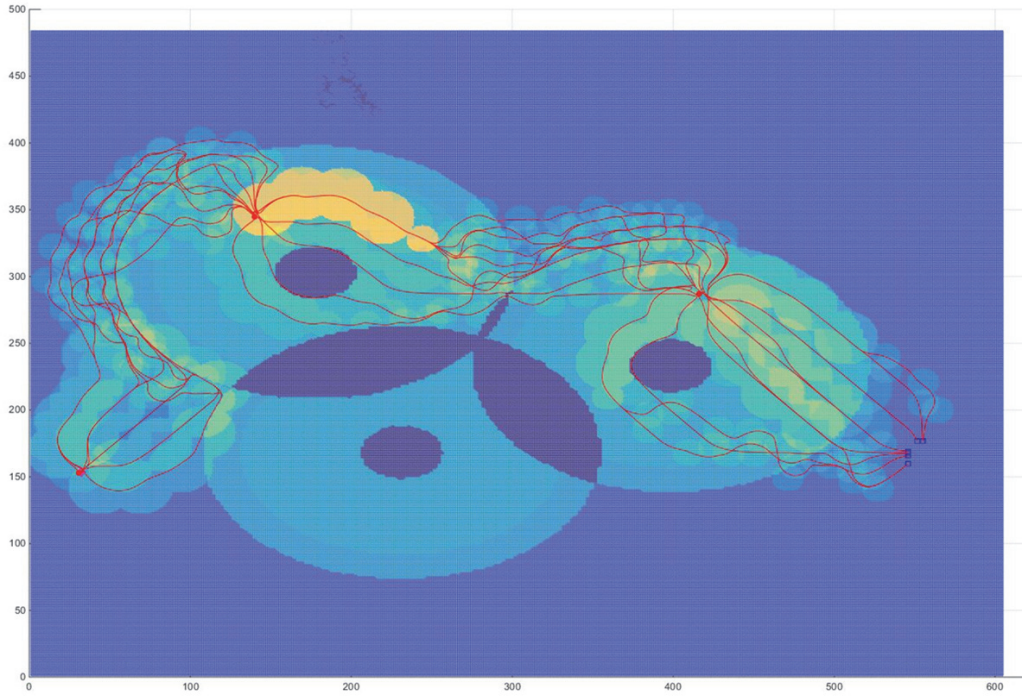


Figure 28: Multi-Path Result (1000 Unique Paths), Runtime = ~4hr

JAR Construction Results

The JAR construction component performed well in both runtime and accuracy. The JARs created by this system aligned properly with the given PE path and also executed quickly. Despite longer PE paths, JAR construction still performed quickly (13.987 s) and provides a fast accurate output to the final stage of mission planning. Figure 29 shows the resulting JARs using the rectangular intersection method.

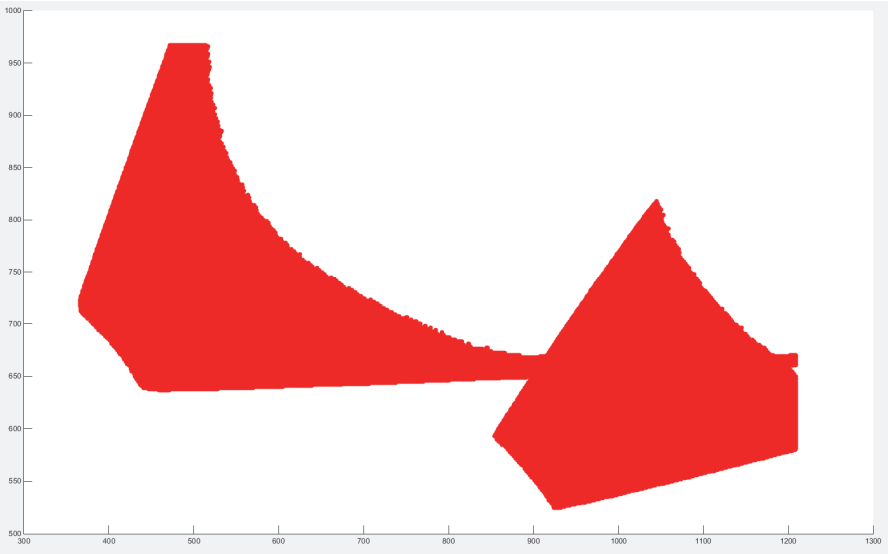


Figure 29: JAR Construction Result, Runtime = 13.987s

EA Path Generation

EA path generation sits alongside PE path generation in terms of computational intensity due to the nature of EA path generation, which requires hundreds to thousands of paths to be generated. EA path generation does not experience gigantic run times due to calculating these paths in parallel. Additionally the majority of these paths are short and require little path planning. The results of this process can be seen in Figure 30 with a runtime of 278.934s.

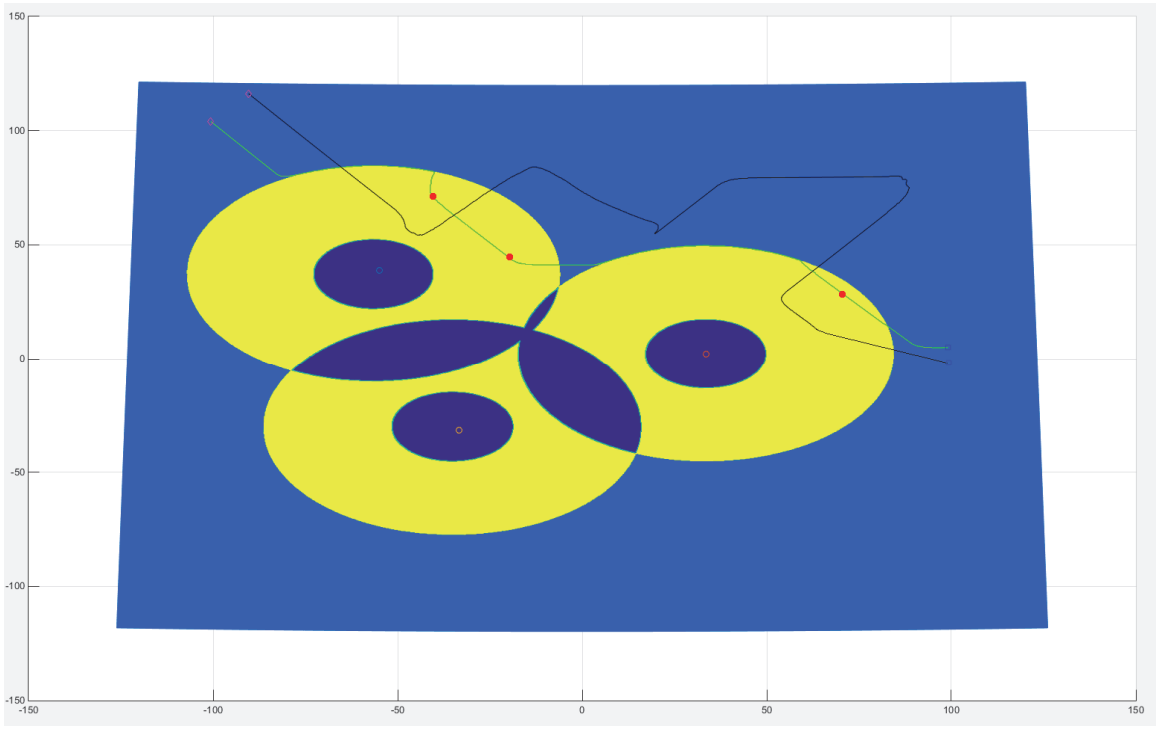


Figure 30: EA Path Generation Result, Runtime = 278.934s

Mission Planning System Results

The results of the mission planning system that have been discussed in this thesis are contained in the MATLAB output in figure 31. The overall system performance result is successful. The accuracy of the system is sound, due to its foundation on DTED, and the overall system runtime is satisfactory. The following is a benchmark of all the processes involved in the system and their performances.

Function Name	Function Type	Calls	Total Time	% Time	Time Plot
nav_ea_waypoints	function	1	424.185 s	75.5%	
gen_ea_waypoints	function	1	59.729 s	10.6%	
createEA_JAR	function	1	17.389 s	3.1%	
nav_waypoints	function	1	16.189 s	2.9%	
mission_check	function	1	15.540 s	2.8%	
get_EA_Fields	function	1	13.291 s	2.4%	
createEA_MAP	function	1	6.500 s	1.2%	
displayjars_rsolov3	function	1	3.370 s	0.6%	
createmap	function	1	2.220 s	0.4%	
plotpath3	function	1	1.840 s	0.3%	
plotpath2	function	1	0.340 s	0.1%	
plotpath	function	1	0.340 s	0.1%	
smooth_paths	function	3	0.120 s	0.0%	
buildthreatmodels	function	1	0.060 s	0.0%	
Totals			561.823 s	100%	

Figure 31: Mission Planning System Benchmarks

While this is only a foundation solution, the groundwork has been laid in order to create a more dynamic system to solve radar jamming escort missions. The next chapter will conclude this work and discuss possible enhancements to the current system, as well as future work in the field of mission planning for radar jamming escort missions.

CHAPTER FIVE

Conclusion

This thesis has developed a MATLAB based tool to assist with mission planning for radar jammer escort missions. The approach we take in developing this tool integrates the use of DTED information, a novel approach for map creation and weighting, and features a modified A* path planning algorithm. The MATLAB framework was tailored for the AMP algorithm, an augmented version of the A* pathfinding algorithm, which was specifically tuned for generating mission solutions. The AMP algorithm generated satisfactory PE paths that are then used in JAR construction to allow for the final phase of the system. Next, the EA path was then derived from the JARs based on automated waypoint selection and a parallelized AMP algorithm. The end result is a demonstrated capability for generating successful PE and EA paths, where the EA is providing jamming for both aircraft properly in order to fulfill the desired mission. The following sections will discuss briefly possible enhancements to the mission planning system, as well as future work in the field.

While this system does provide a satisfactory mission solution, there are areas of the system that can be enhanced to provide a more favorable output. The first enhancement that could be made is to use higher resolution DTED. This will produce a higher resolution map and, in turn, a higher resolution output. A higher resolution output will require a longer runtime, but because the system is built on algorithms favoring speed, the increased resolution should not cause the system to become ineffective. It

would also be desirable to more thoroughly evaluate the capabilities of our system using realistic EA and enemy radar parameters. While these values are classified, incorporating them into the system would allow for more accurate radar ranges and jamming models. Finally, implementation of more advanced JAR generation techniques would provide more accurate EA paths. Since this model uses the rectangular intersection method as opposed to the more accurate elliptical and ellipsoid intersection methods, improvement can be achieved by using one of the latter two methods. This would come at the expense of increased runtime.

There is significant future work that could be done to improve this tool. In this particular case there are two key research areas that could potentially push this thesis' solution to the next level. The first is the adoption of a global and local planner model, which is seen in J.P. van den Berg's dissertation [5]. This path planning model works by having two planners generating solutions, where one operates on a large scale and guides the local planner that is calculating a path at a much higher resolution. The second idea is to incorporate the ability to run the system adaptively in real-time for the purpose of reacting to a changing mission environment and guide the EA and PE paths simultaneously. This will require a significant amount of research, and most likely, specific hardware to accommodate the computational complexity. The adaptability of this system could be derived from Berg's work [5] in conjunction with Trower's thesis [6] on porting high level path planning algorithms to FPGAs, which could be used to develop a hardware solution.

In summation, with the system developed in this thesis and the future work in the field, solutions to the Navy's mission planning problems are within reach. This research

hopes to aid in the development of mission solutions in order to expedite the process of mission planning and to develop solutions that will help save lives.

BIBLIOGRAPHY

[1] J. Dark, J. Buscemi, and S. Burkholder, "Aircrew Aid to Assess Jam Effectiveness," Patent US007 427 947B1, Sep. 23, 2008.

[2] W. Troy, J. Stone, and M. Thompson, "A case study on modeling jamming acceptability regions for escort mission planning," 2015 IEEE Aerospace Conference, 2015.

[3] P. Hart, N. Nilsson, and B. Raphael, "A Formal Basis for the Heuristic Determination of Minimum Cost Paths," IEEE Transactions on Systems Science and Cybernetics IEEE Trans. Syst. Sci. Cyber., vol. 4, no. 2, pp. 100–107, 1968.

[4] E. W. Dijkstra, "A note on two problems in connexion with graphs," Numer. Math. Numerische Mathematik, vol. 1, no. 1, pp. 269–271, 1959.

[5] J. P. van den Berg, "Path Planning in Dynamic Environments," dissertation, 2007.

[6] J. Trower, "Accelerating Path Planning Algorithms with High Level Synthesis Tools and FPGAs," thesis, 2012.

[7] W. Xinzeng, C. Linlin, L. Junshan, and Y. Ning, "Route planning for unmanned aerial vehicle based on threat probability and mission time restriction," 2010 Second IITA International Conference on Geoscience and Remote Sensing, 2010.

[8] K. Tulum, U. Durak, and S. K. Yder, "Situation aware UAV mission route planning," 2009 IEEE Aerospace conference, 2009.

[9] C. Vasudevan, "A novel navigational planning scheme for autonomous underwater vehicles," Proceedings of OCEANS'94.

[10] A. Stentz, "Optimal and efficient path planning for partially-known environments," Proceedings of the 1994 IEEE International Conference on Robotics and Automation.

[11] H. A. Simon, "Rational choice and the structure of the environment.," Psychological Review, vol. 63, no. 2, pp. 129–138, 1956.

[12] A. Nash and S. Koenig, "Theta* for Any-Angle Pathfinding," Game AI Pro 2 Collected Wisdom of Game AI Professionals, pp. 161–172, 2015.

[13] Subh83, Astar progress animation. 2011.