3-26-2020

Timely Near-Optimal Path Generation for an Unmanned Aerial System in a Highly Constrained Environment

Kyle J. Matissek

TIMELY NEAR-OPTIMAL PATH GENERATION FOR AN UNMANNED AERIAL SYSTEM IN A HIGHLY CONSTRAINED ENVIRONMENT

THESIS

Kyle J. Matissek, Captain, USAF

AFIT-ENY-MS-20-M-271

DEPARTMENT OF THE AIR FORCE
AIR UNIVERSITY

AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

The views expressed in this thesis are those of the author and do not reflect the official policy or position of the United States Air Force, Department of Defense, or the United States Government.

AFIT-ENY-MS-20-M-271

TIMELY NEAR-OPTIMAL PATH GENERATION FOR AN UNMANNED AERIAL SYSTEM IN A HIGHLY CONSTRAINED ENVIRONMENT

THESIS

Presented to the Faculty

Department of Aeronautics and Astronautics

Graduate School of Engineering and Management

Air Force Institute of Technology

Air University

Air Education and Training Command

In Partial Fulfillment of the Requirements for the

Degree of Master of Science in Aeronautical Engineering

Kyle J. Matissek

Captain, USAF

March 2020

AFIT-ENY-MS-20-M-271

TIMELY NEAR-OPTIMAL PATH GENERATION FOR AN UNMANNED AERIAL SYSTEM IN A HIGHLY CONSTRAINED ENVIRONMENT

Kyle J. Matissek

Captain, USAF

Committee Membership:

Dr. Richard Cobb, USAF
Chair

Dr. David Jacques, USAF
Member

Dr. David Grymin, USAF
Member

Lt Col Michael D. Zollars, PhD, USAF
Member

Abstract

A current challenge in path planning is the ability to efficiently calculate a near-optimum path solution through a highly-constrained environment in near-real time. In addition, computing performance on a small unmanned aerial vehicle is typically limited due to size and weight restrictions. The proposed method determines a solution quickly by first mapping a highly constrained three-dimensional environment to a two-dimensional weighted node surface in which the weighting accounts for both the terrain gradient and the vehicle's performance. The 2D surface is then discretized into triangles which are sized based upon the vehicle maneuverability and terrain gradient. The shortest feasible path between the nodes of the two-dimensional triangulated surface is determined using an A* algorithm. An optimal path is then chosen through the unconstrained corridor to yield a quick near-optimal path solution in three-dimensional space. This technique requires prior knowledge of the terrain map and vehicle performance. The cost to traverse each segment of the map is independent of the starting position on the map and can be pre-calculated once the goal position is known. The proposed method allows for a rapid path solution from any start position to a goal position while satisfying all constraints. It was shown that employing the methodology herein resulted in near-optimal solutions in less than a couple seconds for the scenarios tested. The future work section proposes methods for improving the algorithms efficiency even further.

Acknowledgments

I would like to express my sincere appreciation to all the members of my research committee – Dr. Cobb, Dr. Jacques, Dr. Grymin, and Lt Col Zollars. Each of them was instrumental in working through this methodology and there is still a lot more potential! Pathfinding and optimization were newer topic areas to me, but they are areas where I have recently found a lot of interest. There were many times when I would brainstorm and begin to question pieces of my work in my head which would delay me from coding or putting something on paper. Each member showed confidence in me and provided the right mix of direction and laughter and expertise to keep me moving along. They have provided many professional and personal learning opportunities. Finally, thank you to my fiancé who has supported me and picked up the slack during the long nights of work. You all have contributed to my great AFIT experience!

Kyle J. Matissek

Table of Contents

Page

List of Figures

Page

List of Tables

Page

TIMELY NEAR-OPTIMAL PATH GENERATION FOR AN UNMANNED AERIAL SYSTEM IN A HIGHLY CONSTRAINED ENVIRONMENT

I. Introduction

Overview

The roles and corresponding requirements of unmanned aircraft systems (UAS) have been rapidly expanding in both the military and civilian industries. UAS are now recognized as critical assets across all levels of the Department of Defense (DoD) [1]. The high demand for UAS is driven by their ability to accomplish the dull, dirty, and dangerous missions, as reflected by the DoD Unmanned Systems Integrated Roadmap for fiscal year 2013-2038 [2]. As the use of UAS become more prevalent and more complex, the desire for autonomy of UAS also increases.

A current challenge in autonomy of small UAS is the ability to efficiently calculate a solution to a complicated route planning problem quickly, while minimizing desired parameters such as time, fuel consumption, etc. Flight through an urban environment or similarly highly constrained environment is a complicated problem. Once an aircraft determines an initial path solution and starts flying the solution, computations to improve that solution must not impede forward flight of the vehicle, since fixed-wing vehicles can't simply pause while calculating a solution and hovering expends more energy than forward flight. Therefore, efficiency and timeliness of a solution is extremely important to a small UAS vehicle that doesn't carry a lot of computing power.

This thesis aims to produce an algorithm that can be used to quickly calculate a path through a three-dimensional (3D) constrained environment by appropriately mapping it

into a two-dimensional (2D) constraint free space, solving the unconstrained optimization problem, and then converting the solution back to the 3D space. The mapping procedure removes the high cost associated with calculating a constrained path by placing a 2D surface over the constraints to account for them in a different manner, much like draping a blanket over an area full of objects; a path is then developed along this surface where all the constraints below will be avoided. The path is identified by searching for the lowest cost segments, which produce a corridor in which a flight path can be defined for a near-optimal solution very quickly. If successful, this type of algorithm would be highly valuable for applications such as flight through urban environments, low-level flight, or other terrain avoidance missions.

Background & Motivation

The Air Force Research Laboratory's (AFRL) 2020 Science and Technology (S&T) Vision is "Intelligent machines seamlessly integrated with humans-maximizing mission performance in complex and contested environments" [3]. To maximize mission performance, many tasks of UAS should be automated to afford humans more time to develop other solutions which may not be trivial. Tasks which are ripe for automation are those which follow a defined set of rules, such as navigation around physical objects.

A shortcoming with most of the current unmanned systems is their inability to autonomously navigate around constraints quickly and efficiently. Currently, if a user wants the UAS to travel from Point A to Point B and avoid constraints along the way in an efficient manner, they need to manually specify pre-defined waypoints which guide the vehicle around the constraints, or wait a long time for a solution to be calculated.

Navigation around objects may seem like a trivial problem to a human, or a GPS fed automobile navigation system giving directions down a defined road, but not all systems are restricted to operating on simple well-defined roads. Traveling down the well-defined road may not be the most efficient path, nor may the user want the vehicle to travel down conventional paths such as roadways. Determining the optimal route through a highly constrained 3D environment, such as an urban environment, is rather expensive in terms of computational power and time to solve. If a continuous, optimal solution is desired, long computation times are commonplace, and there is no guarantee that a solution will be found by the algorithm. Considerations such as vehicle dynamic limitations and non-linear constraints quickly increase the computational complexity of the problem.

An optimum path, as defined herein, is one which minimizes a cost metric along the path, such as minimizing the throttle, steering angle changes, or time to traverse the path. The optimum route is found from the solution of an optimal control problem. There are two challenges that are problematic to the optimal control problem: solution convergence and calculation time. If the problem is poorly defined, the solver may never converge to a solution. To increase the probability of convergence, most optimization solvers require an initial guess of the solution. The dilemma with this is, if a good estimate of the optimal solution was known, then the optimal solver may not be required in the first place. Additionally, the amount of time it takes to solve an optimal control problem increases exponentially with the number of non-linear constraints imposed. Each object that must be avoided is defined as a constraint to the problem. It is easy to see how an urban or mountainous environment consist of many constraints and cause the calculation time to rapidly grow.

The ability to determine a feasible near optimal path in near real time is in high demand. Vehicles that cannot autonomously and efficiently navigate and respond quickly will become obsolete - especially on the battlefield. This thesis aims to develop an algorithm which determines a near-optimal feasible solution quickly by specifying a path through a weight-mapped 2D corridor, discretized through a series of simply connected triangles defined herein as a Connected Simplex Corridor (CSC). Once identified, the path through the corridor can be further refined to yield a lower cost flyable solution.

Problem Statement

There is currently not a robust and efficient method for finding a near-optimal navigation solution through a constrained urban environment in near-real-time. Current algorithms are computationally expensive, and most are not very robust.

A *robust method* is one that is insensitive to small changes in the model – such as the ability to move the starting or ending location to anywhere on the map and still be capable of finding a solution. Non-robust methods are defined as being very sensitive to the parameters which define the problem, such as the inputs or constraints. A non-robust method may never converge to a solution or take a long amount of time to find a solution for a particular configuration.

An *efficient method* in this research is one which is computationally efficient, or one which requires using minimum computational power to solve the problem. A computationally efficient method is necessary to achieve the near real time solution for large complicated maps. Computational efficiency is necessary for a small UAS, since they are typically outfitted with low-power computers with minimal computation ability due to the desire to be small and lightweight.

The difference between an optimal and *near-optimal* or sub-optimal solution is the closeness of the near-optimal solution to the optimal solution. If the solution is the best it can possibly be, then it is the optimum solution. A near optimum solution may be only a small percentage more costly than the optimum solution. A near optimum solution is typically “good enough” for most applications. When solutions to non-linear problems are solved, it can only be claimed that the solution is optimal based on the tools used to acquire to solution and the approximations in the models. Unless analytically proven, there is no guarantee that a better solution exists.

A *near-real-time* solution in this research is one which is available as needed throughout the mission. Offering complete solutions within a few seconds is acceptable.

This thesis aims to provide a robust, near-optimal, path solution for navigation in a constrained urban environment in near-real-time.

Research Hypothesis

Both 3D digital map data and vehicle performance constraints can be mapped into a discretized 2D weighted map. The costs to traverse each section of the map can be precalculated. An algorithm can be used to quickly find a near optimum feasible path through the nodes of the 2D weighted map, which can then be mapped back to 3D space to yield a near optimum solution.

Investigative Questions

i. How should the discretization technique scale (temporal & spatial) based on terrain?

ii. How should the discretization technique scale (temporal & spatial) based on vehicle performance (caused by size/speed/maneuverability[1])?

iii. What method is the best for finding an optimal path through the equivalent 2D discretized surface?

iv. What are the tradeoffs of this proposed method as compared to alternative methods in use and proposed in the literature?

Methodology

This work will analyze the feasibility of designing an algorithm which can robustly and quickly attain a near-optimal path solution across a highly constrained terrain. Taking advantage of existing efficient 2D algorithms, the proposed method maps a 3D constrained environment to a 2D weighted map, which is discretized and weighted based on the environment and vehicle performance. An A* algorithm is then used to determine the lowest cost, near-optimal path, in near-real-time. The solution is then mapped back to 3D for implementation.

The digital terrain model (DTM) will consist of nodes that have known characteristics, such as a X location, Y location, and median Z elevation. The spacing between the nodes can be selected smartly to create a discretized map which ensures a feasible flight path is possible through the connection of arbitrary nodes in order to ensure a robust solution.

The mapping from 3D to 2D will be done by utilizing the 2D information and accounting for the third dimension (elevation of the terrain) by applying weights to the nodes which make up the map. The weights will be used to account for the difference in

[1] Maneuverability as used herein refers to the UAS max climb, turn, and bank rates

elevation between nodes, the vehicle performance between nodes, and any additional constraints such as keep out zones. A heuristic algorithm will then be used to determine the optimal path through the weighted nodes.

The solution from the proposed algorithm defines an initial solution through an unconstrained corridor. The solution can then be further optimized within the unconstrained corridor to yield a near-optimum solution.

Assumptions

The following list includes the assumptions made and applied to this method:

- Obstacle and Constraint Map is Provided in Advance (DTED, etc.)
- Start and End Position Lies on the Constraint Surface
- Static/Fixed End Goal
- Constant Velocity Aircraft
- Disturbances Are Not Included (Such as Wind)
- Flight Remains on the Constraint Surface
- Segment Costs Assume a Constant Climb or Descent Rate from Point to Point

Obstacle and Constraint Map is Provided in Advance – The fundamental premise that allows this technique to provide a quick solution is that information about the terrain and constraints are known before the planned flight. This allows the algorithm ample time to calculate the cost to travel each segment of the entire map based on constraint characteristics and vehicle performance. Data in the form of X, Y, Z will be used for this investigation; however, any terrain digital elevation model (DEM) which can be converted

to the form of X,Y,Z such as Digital Terrain Elevation Data (DTED), Shuttle Radar Topography Mission (SRTM) data, or LIDAR data, may be used.

Start and End Position is on the Constraint Surface –This research assumed that the vehicle starts and ends at locations on the constraint surface. If desired, it should be fairly trivial to add the ability to start off the surface and determine when and where to intersect and fly along the surface.

Static/Fixed End Goal - This technique assumes that the end goal position is not moving. Since the solution is provided a near-real-time, solutions can be quickly recomputed for changes in the goal state.

Constant Velocity Aircraft - Typically, aircraft are likely to adjust their speed to maneuver over and around different obstacles. This research assumes the aircraft desires to fly the route at a constant velocity which is not uncommon for small UAS. Some methods to handle a varying velocity are discussed, but not implemented.

No Disturbances (such as Wind) – This technique does not account for stochastic parameters, and disturbances such as wind. It is hypothesized that accounting for a constant uniform wind across the entire map may be accounted for easily and a constant non-uniform wind may be accounted for with slight increase in computational cost; however, a stochastic model or any time varying model would likely be more costly and is left as future research. For cases with strong winds, wind should be accounted for, even if the vehicle is capable of flying the designated path in the wind conditions present, since it should be ensured that the path chosen is the near-optimum path when the influence of the winds is accounted for.

Flight Along the Constraint Surface –If there was a catwalk, or some similar feature between two buildings, this model assumes a constraint surface which is above the catwalk. It would not calculate a path to traverse under the catwalk or other ‘tunnel’ like features.

Segment Costs Assume a Constant Climb or Descent Rate from Point to Point -The cost to travel from point to point on the map is calculated by assuming a constant climb or descent rate from point to point. The cost incurred by turning or initiating a change in climb or descent is not considered.

This algorithm also assumes the user wants to follow the contour of the constraint surface and fly at a minimum height above ground.

Uses of the Research

The work of this thesis has the ability to immediately provide the Air Force Research Laboratory with a method to find a near-real-time path solution, which may be integrated into further autonomy research. The less time the vehicle spends acquiring a solution, the more time and effort it can spend calculating other tasks, such as those required for Manned Unmanned Teaming (MUMT). MUMT is the capability of a human or human controlled machine to team with an unmanned machine to accomplish a task. The work may also be modified to be used in the AFRL open source multi-vehicle cooperative decision-making software called UxAS.

Maintaining a low flying altitude which traverses the surface may provide operational advantages such as radar avoidance, a decreased time for a ground personnel to observe and react to an overflying aircraft, and other aircraft collision avoidance may be realized.

Document Overview

Chapter II contains the findings from the literature review, which both identifies the current research gaps and provides a foundation upon which to build this research. Chapter III covers the methodology used in the research and discusses the validation of the results. Chapter IV shows simulation results of different scenarios and solutions. Chapter V presents and discusses the results. Chapter VI summarizes the results, draws conclusions, and discusses recommendations for future research.

II. Literature Review

Chapter Overview

The near-real-time, near-optimal path planning problem for Small Unmanned Aerial Systems (SUAS) is one that presents many challenges. An optimal path is one where the "sum of the transition costs is minimal across all possible paths leading from the start state to the goal state" [4]. In this research, the cost used will be the time to travel the path. As discussed below, algorithms currently exist which find the optimum path for a UAS in a 3D constrained environment; however, computation times are excessive and convergence to a solution is not guaranteed. Other algorithms can calculate a path quickly, but it may not be the optimum path.

As introduced in the methodology section for Chapter I, several tasks are required. Therefore, this chapter presents the literature review of this research which is broken into three sections: Maps, Search Algorithms, and Previous Work. The first section introduces the different map discretization techniques and different map characteristics. The second section discusses the algorithms commonly used for 2D pathfinding. The third section will cover some of the previous work which fueled this project.

Justification for Research

Modeling both the vehicle system and the environment with high fidelity models to attain a high-fidelity path solution leads to very long computation times. Similarly, oversimplifying the vehicle dynamics or the environment constraints leads to less than optimal solutions. Algorithms which can determine a near-optimal solution quickly are in high demand. One way to significantly reduce the computation time needed to generate a

near-optimal path is to provide a good initial guess of the near-optimal solution. Solutions from fast algorithms that find slightly sub-optimal solutions are ideal for supplying an initial guess of the near-optimal solution. The high-fidelity algorithm can then take the slightly sub-optimal solution and refine upon it to yield near-optimal solution.

Map Characteristics Common to Pathfinding

This section discusses characteristics common to pathfinding, such as different ways to measure distance across a map, how to discretize the map into subsets, how to account for different characteristics present in the environment, and algorithms commonly used to find paths across maps.

1) Distances

Three common distances used for pathfinding are the Manhattan distance, Euclidean distance, and Chebyshev distance. The Manhattan distance only allows movement in either the *x* or *y* direction for any moment in time, much like traversing the city blocks of Manhattan, NY. The Euclidean distance is the straight-line distance through space, which can be a straight line in *x*, *y*, and *z*, even though only *x* and *y* are shown in Figure II.1. Euclidean distance is the most common way to measure distance in aviation, since an airplane can fly a direct route. The Chebyshev distance is the straight-line distance from one point to another, but its length is determined by the absolute value of the maximum length of *x* or *y*. The Chebyshev distance is commonly used in logistics since it can account for restrictions such as speed limitations of independent axes on an actuator or an x-y crane. The amount of time it takes a crane to move one *x* unit and zero *y* units is the same amount of time it takes it to move one *x* unit and one *y* unit simultaneously, assuming

the motors' speed and side lengths were the same. Even if one axis achieves its goal state quickly, both axes must be at their goal state for the system to be at its overall goal state. Thus, the amount of time it takes the crane to travel along the most time-consuming axis is the same as the total time.

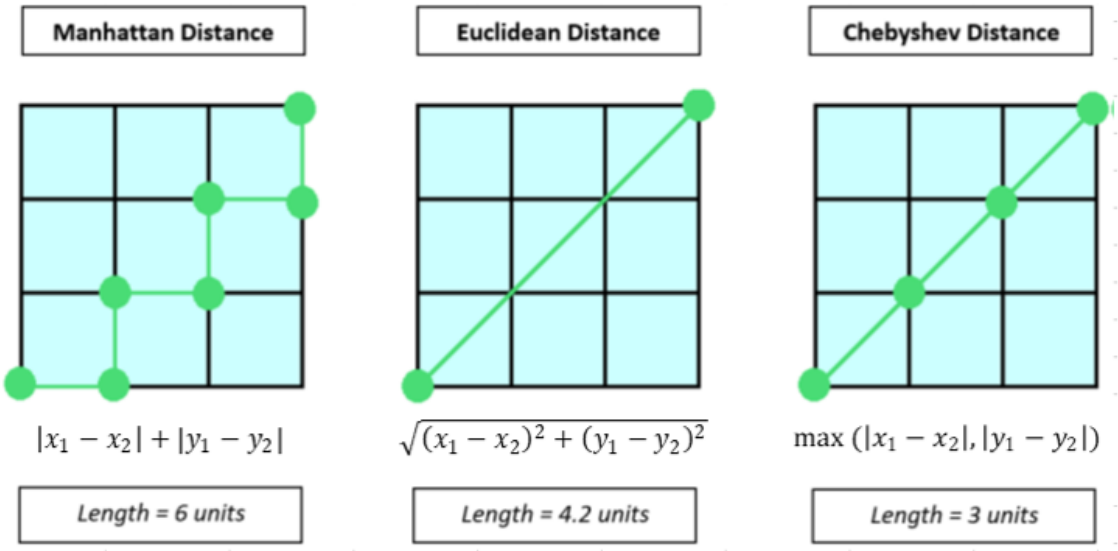


Figure II.1 Manhattan, Euclidean, and Chebyshev Distance Comparison [5].

2) Discretization Techniques

The discretization of a surface highly influences the fidelity and timeliness of the solution. Discretization is the process of turning something continuous into discrete parts or subsets, as commonly seen in mathematics with finite difference equations. This allows a large complicated region or equation to be represented by a subset of simple regions or equations.

It is desired to utilize the lowest number of segments, or coarsest discretization, which accurately describes the region. The discretization should be scaled based upon the

complexity of the environment and not simply the size of the environment. For example, if one region of the map is a nearly continuous flat plane which could be modeled with a few key points, then it would be inefficient to model such a region with many more points than necessary, since it would not provide us with any more useful information about the region. Modeling the region with three points (which is the minimum number of points to create a surface) would be more desired than using a finely meshed grid, since the fine grid provides little more information about the region than the three points do.

2.1) Grids

The most well-known discretization method is the uniform grid. Grids consist of an area broken into square or rectangular sections which are the same size throughout the map. The diagrams in Figure II.1 are examples of a grid and paths across them. Often times, a grid discretization for a region of complex terrain is not an efficient way to model the terrain when attempting to use the least number of points. For example, equal spacing might cause nodes around a steep mountain peak to be located on either side of the peak but not on the peak, which would result in a poor model of the peak. The major disadvantage of grid discretization is the inability to account for complex terrain by using a minimum number of nodes. Using a grid discretization often leads to more nodes then necessary to model the space, thus slowing the computation time of the search algorithm.

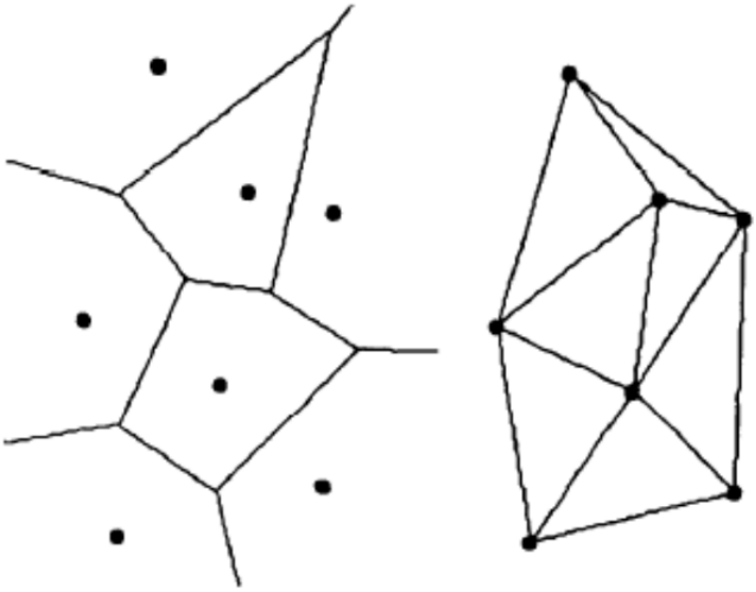
2.2) Voronoi Diagram

One of the earliest known discretization methods is a Voronoi discretization. A Voronoi diagram is created by drawing lines equidistant between each set of nodes, as shown in Figure II.2 [6]. A Voronoi diagram may be useful to understand which node is

the closest to the current position if the position on the diagram is known. A Voronoi diagram is also easily used to maintain a minimum distance away from all obstacles, which is indicated by the lines drawn between the nodes.

2.3) Delaunay Triangulation

Another common method of discretization is triangulation. Triangles are useful for the discretization of complicated shapes, which is why their 3D counterpart, the tetrahedron, is commonly used in finite element analysis (FEA). The most common form is Delaunay triangulation. Delaunay triangulation is the straight line dual of the Voronoi diagram [6]. The lines of the triangles in the Delaunay triangulation would intersect perpendicular to the Voronoi diagram lines. A set of nodes can be connected in many ways to form a surface of triangles; however, the points of a triangle in Delaunay triangulation lie on the perimeter of a circle which has its center at the vertex of a Voronoi polygon. The connection of three points to create triangles for a Delaunay triangulation is not arbitrary.



A Voronoi Diagram (left) and a Delaunay Triangulation (right) show the difference in discretization for the same nodes.

Figure II.2 Voronoi Diagram and Delaunay Triangulation [7]

2.4) Constrained Delaunay Triangulation

A common discretization method used when a space contains path constraints (obstacles such as buildings, mountains, etc.) is the constrained Delaunay triangulation method. It is undesirable to have paths which pass through constraints since they would be infeasible paths and should not be considered. Constrained Delaunay triangulation connects the edges of the triangles to corners of constraints, as shown in Figure II.3. This technique allows the enclosed constraints to be modeled as a series of polygons which can then be removed from the search space; thus, ensuring that the available paths are always free of path constraints. Constrained Delaunay triangulation characterizes a complicated and constrained region by a relatively small number of paths since no paths lead to the edge of a constraint unless the edge was a corner to the constraint. The number of triangles which

form the map is based on the number of constraints and not necessarily the size of the map. An undesirable trait of Delaunay triangulation is the number of triangles required to replicate a circular constraint. MATLAB has a built in Delaunay triangulation function "delaunay(X,Y,Z)" which can generate a vector of triangle vertex coordinates and triangle numbers that represent the discretization, for an input vector of *x*, *y* and *z* coordinates.

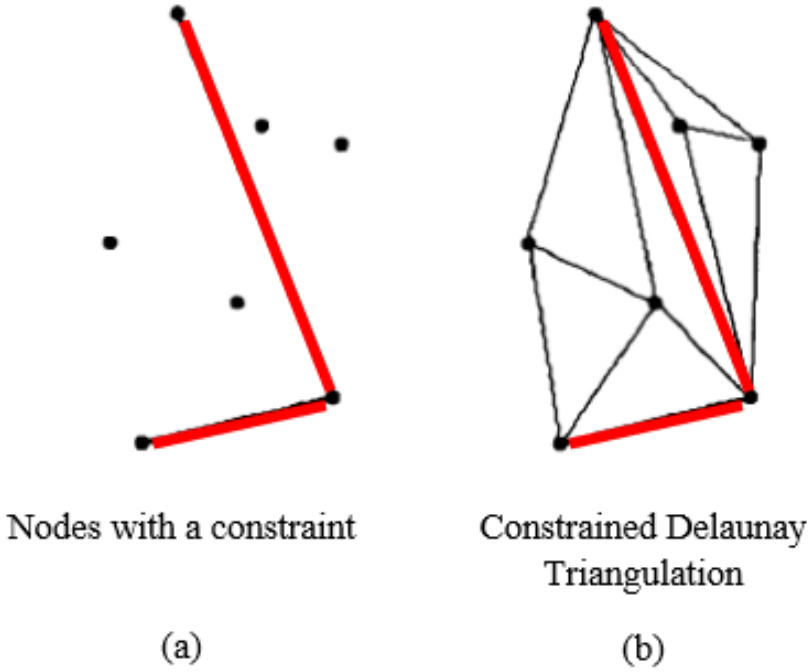


Figure II.3 Constrained Delaunay Triangulation [7]

2.5) Visibility Graphs

Another discretization method which is very useful but is computationally costly is visibility graphs. Visibility graphs create paths between nodes which are visible to each other, meaning a straight line can be drawn from one node to another without intersecting a constraint [8]. The vertices are connected when no obstruction between the two vertices

exists. Paths are not created between two nodes that are not visible to each other. Visibility graphs can be very powerful if they are precomputed; otherwise, they are computationally expensive and slow. Figure II.4 shows the unobstructed paths which are fewer than the number of paths that would be attempted if all the nodes were connected directly to each other. This saves on computation time because far fewer paths are considered.

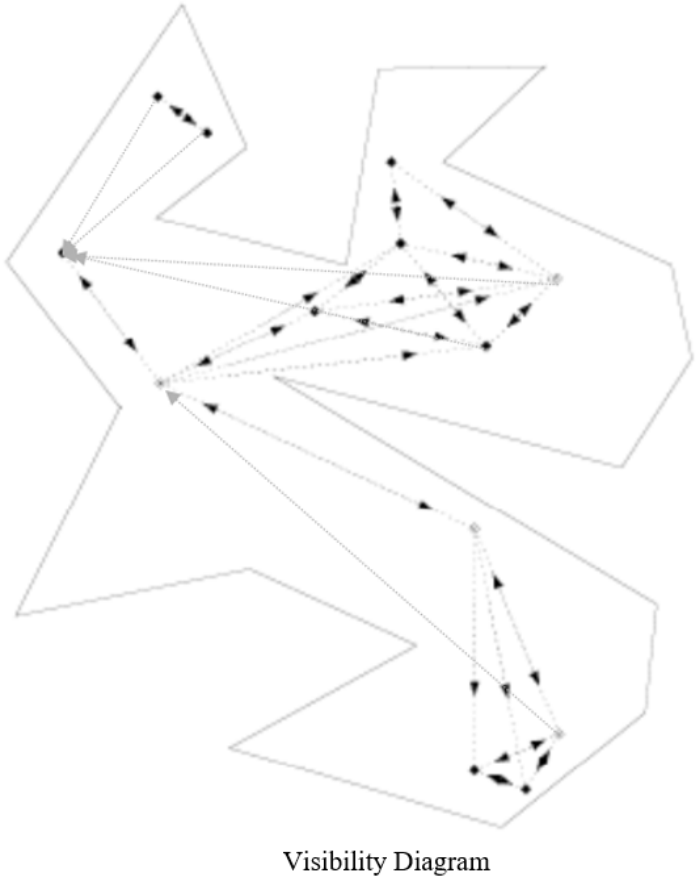


Figure II.4. An Example of a Visibility Diagram [8].

3) Directed vs. Undirected Maps

Certain maps have restrictions on the directions of travel allowed along certain paths. If there is a path on a map requiring restrictions on the allowable direction of

travel, then a directed map would be appropriate. A directed map is one which restricts the direction of travel along a path, similar to one-way streets in a city. An undirected map does not restrict the direction of travel along a path. Directed maps ensure that only paths which travel in the allowable direction are evaluated by pathfinding algorithms.

4) Weighted Regions

A way to account for characteristics about the environment is by applying weights to nodes or regions on a map. The weights indicate the intensity of the characteristic being modeled. The weight (or cost) to traverse the area can be high or low dependent on terrain characteristics.

Weighted maps are commonly used in path planning to account for attributes of the terrain such as the difficulty to travel the path due to terrain roughness or steepness [9]. A weighted map can be used to account for the difference in cost (such as time) to traverse the segment. It will likely take more time to travel a distance across mountainous terrain than it would to travel the same distance on flat ground, so the mountainous terrain would have a higher cost of time. In addition, the direction of travel on the mountain is significant. Traveling up the slope has a different weight then traveling down the slope.

Some methods even apply a negative weight to account for characteristics, but it is more common for weights to be only positive in value. It does not make sense to use negative weights for most instances. For example, if the object being measured is the time to fly the path, then it would not make sense for a path segment to take a negative time to fly the path.

Weights can be very useful to account for characteristics which are difficult to account for otherwise, and is the technique exploited herein when representing 3D terrain as 2D weighted maps.

5) Constraints

Constraints are points or regions on the map which the algorithm must abide by certain parameters, such as traveling at a certain speed or avoiding the region all together. One way a constraint can be modeled is as a node with a weight to associate the constraint to the node. If the constraint is an impassible region, an infinite weight is applied to prohibit the algorithm from selecting the node as a node for the path. When constraints are a hard constraint, meaning that the solution should not violate the constraint, then they are typically removed from the search space entirely by using a method such as constrained Delaunay triangulation. Constraints such as the physical terrain, aircraft performance, and keep out zones are considered in this methodology.

Pathfinding Algorithms

Once the space is appropriately discretized and characterized with appropriate weights, then an algorithm to search the space for the best path is needed. Algorithms used for pathfinding can generally be broken into two categories, non-heuristic and heuristic. Early algorithms were non-heuristic, but as time went on, it was obvious that the algorithm should be able to utilize information about the final destination if its final destination is known. The heuristic term helps the algorithm by indicating if the path is propagating toward the goal state or not [4].

1) Non-Heuristic

Non-heuristic search algorithms search the space blind to the location of the final destination. The only information they use is the information obtained while expanding out from the starting position.

1.1) Dijkstra's Algorithm

Dijkstra's algorithm was designed by Edsger Dijkstra in 1959 for the traveling salesman problem and is perhaps the most well-known pathfinding algorithm [10]. The objective of the traveling salesman problem is to minimize the cost to visit all nodes (the cheapest path to visit all of the salesman's clients). Another common variant of Dijkstras algorithm places a starting point and finds the lowest cost path to reach each individual node. The starting node is called the parent node and the node that is expanded next is called the child of that parent node. Each node, besides the starting node, has a parent node. New children nodes are created upon each iteration (also known as a generation). The algorithm branches out from each node upon each iteration, but only expands (or produces children) from the cheapest node, leaving the more expensive node for consideration upon next iteration. The cost of the nodes which were skipped in the previous iteration is tracked and those nodes are expanded upon when they are the cheapest node to expand when compared to the other nodes being considered [10]. For the example in Figure II.5, the expansion from A to C will not be considered until point C is being expanded. This is because the cost of 4 is more than the cost of the 1 plus 2 incurred by choosing the route between nodes A, B then C.

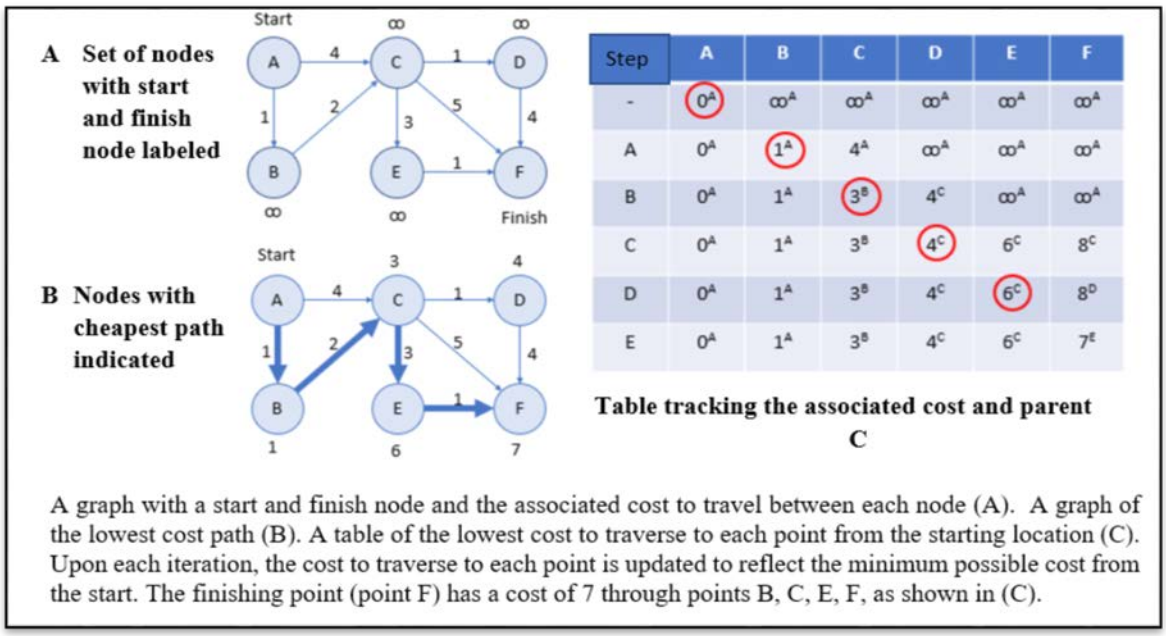


Figure II.5 Dijkstra Algorithm Search Example [5]

1.2) Rapidly-Exploring Random Tree (RRT)

The Rapidly-Exploring Random Tree algorithm searches the space randomly by initially propagating outwards and then growing “branches” off each limb. This is like a spider-web expanding across a map. The branches first reach out across the graph, then each limb starts branching off to increase the density of the limbs across the map.

Heuristic Search Methods

When information about the goal state is known, it is common to use a heuristic to speed up the search. Heuristics provide information about the goal state to the algorithm and can be used to determine the progress toward the goal upon each iteration. The most commonly used heuristic is the Euclidean distance. Algorithms which include a heuristic

typically add the heuristic cost multiplied by the heuristic weight ($h * w$) to the operating cost (g) to achieve a total cost (f) as shown in Equation 2.1.

$$\text{TotalCost} = \text{OperatingCost} + \text{Heuristic} \tag{2.1}$$

Applying a heuristic cost into an algorithm helps drive the paths toward the goal state. A heuristic weight can be utilized to magnify or attenuate the influence of the heuristic.

1) Heuristic Weight Factors

A heuristic weight effects the timeliness of solving for a solution and for most environments it effects the optimality of the solution, therein effecting the optimality. Changes in the heuristic weight changes how fast the algorithm will drive straight toward the goal. A low weight factor on the heuristic will yield near-optimal solutions that are still faster than using no heuristic at all. A large weight factor on the heuristic may decrease the amount of time it takes to find a solution, but the solution may be sub optimal [11]. A large heuristic weight would allow for the calculation of a solution in less time, but the path would likely be more of a straight-line path and is likely to be less optimal than a path obtained by using a mild heuristic weight.

A situation when the weight of the heuristic in the A* algorithm does not affect the optimality of the solution is when the start and end point is on either side of a flat terrain with consistent cost. For this unique case, the solution would be the same straight-line path for an algorithm using any heuristic weight; however, a higher weighted heuristic algorithm would find the solution faster than the lower weighted heuristic algorithm. For complex terrain with segments of varying cost, a higher weighted heuristic would yield a more

straight-line path than a lower weighted heuristic. More information about weighted A* algorithms can be found in [11].

1.1) A* Search Algorithm

Combining Dijkstra's algorithm with a heuristic led to the creation of the A* search algorithm (pronounced A-Star). The A* algorithm was designed in 1968 by Peter Hart, Nils Nilsson, and Bertram Raphael from the Stanford Research Institute to find a path for their robot [12]. A* commonly uses the Euclidean distance as the heuristic cost, which is added to the operating cost to attain the total cost. The initiation and order of the expansion of nodes is the same as Dijkstra's algorithm, but it is influenced by the heuristic which becomes more costly if the nodes are propagating away from the goal state. While Dijkstra's method tends to propagate in all directions choosing the lowest cost upon each iteration, the A* algorithm propagates toward the goal state and very rarely propagates away from it. The A* algorithm is considered an "informed search" and a "best-first search" method since it searches the space until a solution is found [13].

The smoothness of the exact path which the A* creates on the map is highly dependent on the discretization method used. The A* algorithm is commonly used in video games due to its ability to generate a near-optimal solution very quickly. This allows video game characters to traverse the map by using seemingly intelligent movements with as little hesitation as possible [12].

1.2) Theta* Algorithm

A modified version of the A* algorithm is the Theta* algorithm. The Theta* algorithm works exactly like the A* algorithm, except upon each iteration of Theta*, the

child node looks to their grandparent node to see if any constraints prohibit the connecting of the two nodes directly. If no constraints prohibit the connection, then the parent node is removed from the path and the child's grandparent node becomes the new parent [14]. No additional nodes between the starting point and the corner of the constraint in Figure II.6 exist because each child was able to connect to the original node without breaking constraints. After all iterations, only 3 nodes exist because the path between them is unconstrained.

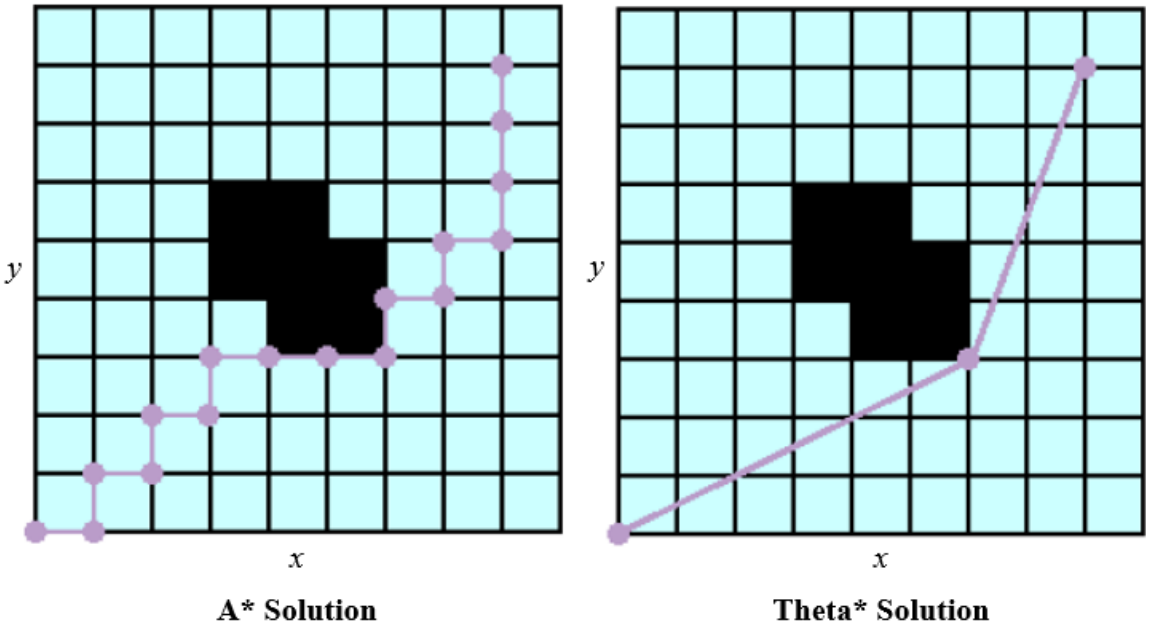


Figure II.6 A* vs. Theta* Path Solution Example [5].

One disadvantage of the A* and Theta* algorithms is their tendency to expand all points in a U-Shaped obstacle before navigating around the obstacle. The algorithm is forced to search within the U-shaped obstacle because the heuristic is pulling the solution into the U. Methods have been developed which analyze the space and close off U-shaped

areas which don't contain the goal to prevent algorithms from exploring unnecessary U-shaped regions [11].

Recent Related Work

Work has previously been done attempting to solve problems similar to the one referenced in Chapter I herein; however, the solutions are either only offered for the 2D space or neglect to discuss feasibility of the vehicle to travel the path along the 3D contoured surface.

1) Other

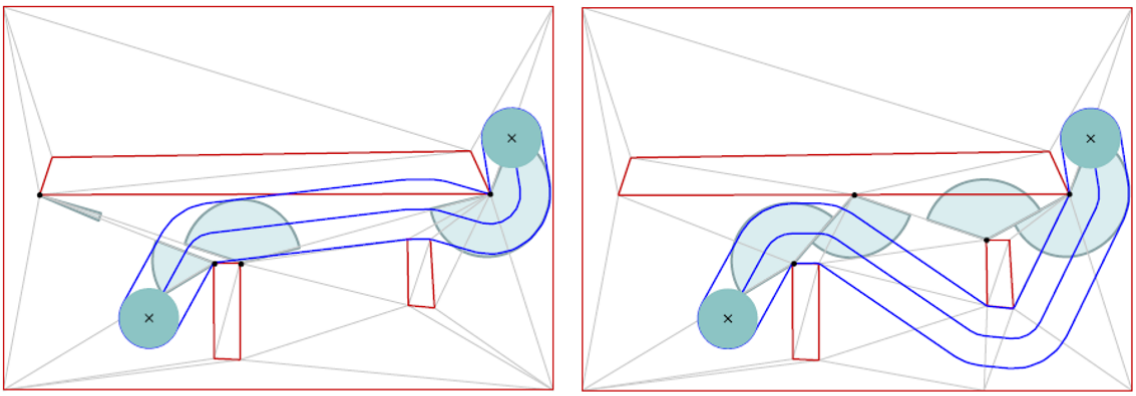
In 2007, Geraerts and Overmars published the Corridor Map Method (CMM). The CMM creates a system of collision-free corridors around the static obstacles in a 2D environment [15]. High-quality complex paths can be planned within the corridor in real-time. In 2010, Geraerts published on the Explicit Corridor Map (ECM). The ECM identifies corridors in a 2D environment which yield the shortest path, largest amount of clearance, or any path in between by using a Generalized Voronoi Diagram [16]. Figure II.7 shows a visualization of the ECM generation process and path solution. He was able to achieve real-time path results for an agent whose movement is defined by the steering angle.

Figure II.7 Explicit Corridor Map Generation Visualization [16]

Identifying a feasible corridor which yields the shortest path across a 3D surface is one of the objectives of this research where this 2D corridor process may be leveraged.

In 2010, Kallmann developed a method which converts a Constrained Delaunay Triangulation (CDT) to a Local Clearance Triangulation (LCT) to easily allow for local clearance checks [17]. The CDT in Figure II.8 passes a clearance check in which the

nodes of each vertex are checked to see if they lie within the clearance distance of other nodes; however, it can be seen that the path violates a constraint. The LCT in Figure II.8 corrects for situations such as this one by placing additional vertices along the constrained edges. The LCT path solution does not pass the local clearance check in the problem region and correctly routes the path around the infeasible section of the map.



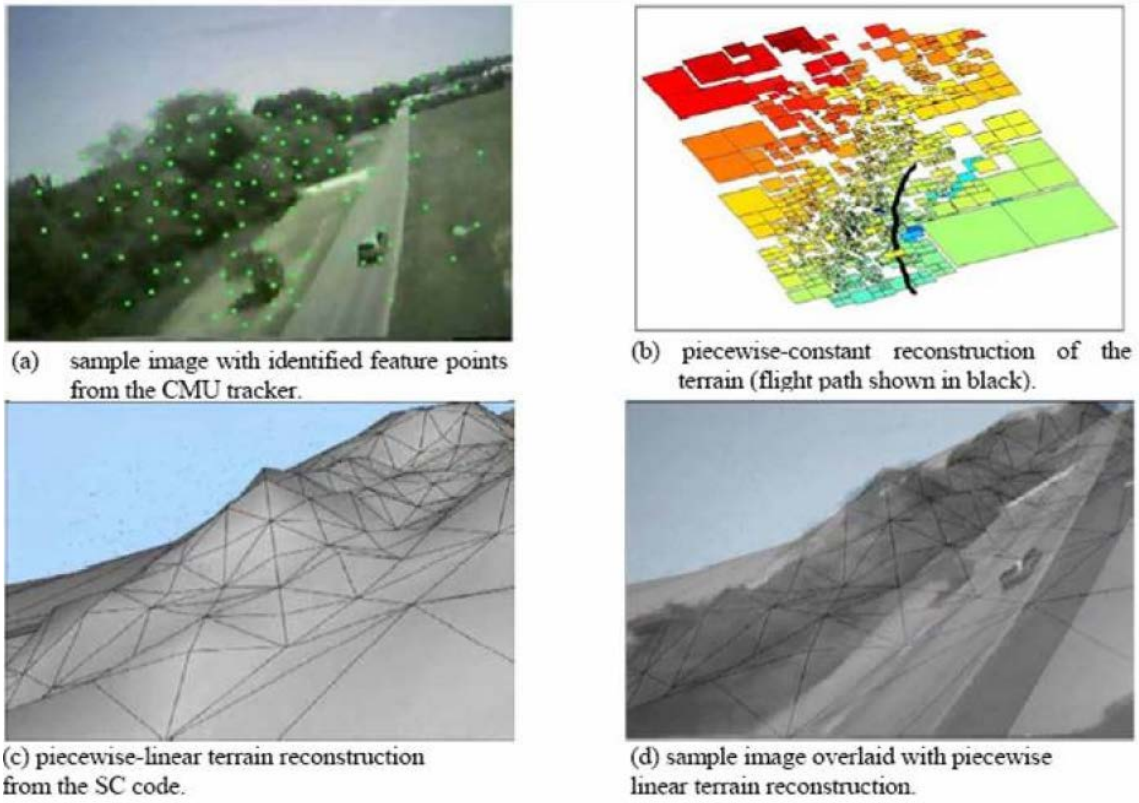
CDT LCT

Figure II.8 An Example of a Constrained Delaunay Triangulation and a Local Clearance Triangulation [17]

The LCT discretization allows for quick clearance checks which leads to quick identification of feasible paths. The methodology of the LCT discretization may be helpful in the discretization of the 3D map. Additional information about navigation meshes and real-time dynamic planning for virtual worlds can be found in [17, 18, 19, 20, 21]; however, this research proposes an incorporation of the constraints directly into the discretization instead of planning around them.

In 2013, Prazenica et al published an article entitled “3-D Implicit Terrain Mapping and Path Planning for Autonomous MAV Flight in Urban Environments” [22]. In their work, they utilize LIDAR measurements to map the environment and use a

receding horizon algorithm to compute kinematically-feasible paths. They utilize an algorithm from the University of South Carolina to discretize the environment where the size of the discretization was based upon the complexity and features of the environment. The discretized environment can be seen in Figure II.9.



(a) sample image with identified feature points from the CMU tracker.

(b) piecewise-constant reconstruction of the terrain (flight path shown in black).

(c) piecewise-linear terrain reconstruction from the SC code.

(d) sample image overlaid with piecewise linear terrain reconstruction.

Figure II.9 Discretization of an Environment Using LIDAR [22]

The hypothesized discretization developed herein is similar to that used in Figure II.9 image (c); however, herein the discretization scale will also be constrained by the performance of the aircraft over the terrain gradient.

In 2014, Zhan et al. published an which seemed to achieve path solutions by utilizing a similar methodology as proposed herein [23]; however, ensuring the feasibility of the discretization and thus that the A* solution was feasible (a key feature), was not

discussed in the article. Additionally, there were other omissions in the article that made the replication of their results impossible.

The article references the use of an ant colony algorithm and genetic algorithm for comparison against a A* algorithm. Use of an ant colony algorithm is typically not the best choice for a timely solution against the A* algorithm for a problem in which the solution is a small segment of the overall map. The type of genetic algorithm used is also not mentioned in the article, more specifically the heuristics used in the algorithm are not identified, and thus were not able to be verified.

Furthermore, the article claims to yield real-time results; however, the size of the map is inconsistently reported as both 2,500,000 Km^2 and 2,500,000 m^2. The map of Km^2 size is a reasonable size, but is referenced only once while the map of size m^2 is referenced throughout the article. Once the large keep out zones depicted in the environment are added to the environment, the number of nodes available to explore are greatly reduced – thus possibly falsely giving the reader the impression that the algorithm is suitable for real-time.

The shortcomings of Zhan et al.'s work, specifically the lack of detail about the discretization of the surface which ensures a feasible solution, provided sufficient justification to continue the research proposed herein.

In 2016, Muñoz et al proposed the use of a 3D any-angle path planning algorithm (3Dana) on 3D surfaces to determine an optimal path for a Mars rover across a digital terrain model [24]. The method achieved solutions which were slightly improved from an A* solution across similar terrain; however, the results were not achieved in near-real time. This research seeks to utilize a similar methodology to find solutions in real-time.

Throughout all of the literature review, one thing is clear – there currently does not exist methods to robustly identify a near-optimal path in a highly constrained 3D environment in near-real time. Optimal control solvers are time consuming when setting up the problem, do not robustly provide a solution, and are expected to take longer than the methodology proposed herein. There exist many efficient graphing methods for path finding in the 2D environment and they have been used and enhanced year after year, as seen in [19, 20, 21]. When utilizing 2D methods for 3D problems, one must ensure the solution is feasible to maneuver in the third dimension. Simply moving an object through 3D space in the direction of a goal is not the same as finding the near-optimal path through the 3D space, especially a space with many constraints.

Recent work at the Air Force Institute of Technology attempted to solve the challenge addressed in Chapter I. The 2D work of Zollars fueled the inspiration for the proposed methodology.

2) AFIT

In 2018, Zollars published a dissertation at AFIT which focused on finding robust path solutions for a UAS in a constrained urban environment. He focused mostly on establishing a path in a 2D environment. His method discretizes the map using a constrained Delaunay triangulation, which removes the constraints from the map. He then finds an optimal path through the midpoints of the discretized triangles using a A* algorithm. The connection of triangles which the A* algorithm passes through identifies what he calls a Connected Simplex Corridor (CSC). The CSC can then be used to find an optimal path relatively quickly because the CSC is free of constraints. Figure II.10 shows an example of a CSC through an urban environment with an optimal path through the CSC,

where the path is a Dubins path that abides by a minimum clearance radius around the constraints. His work can be found in [25].

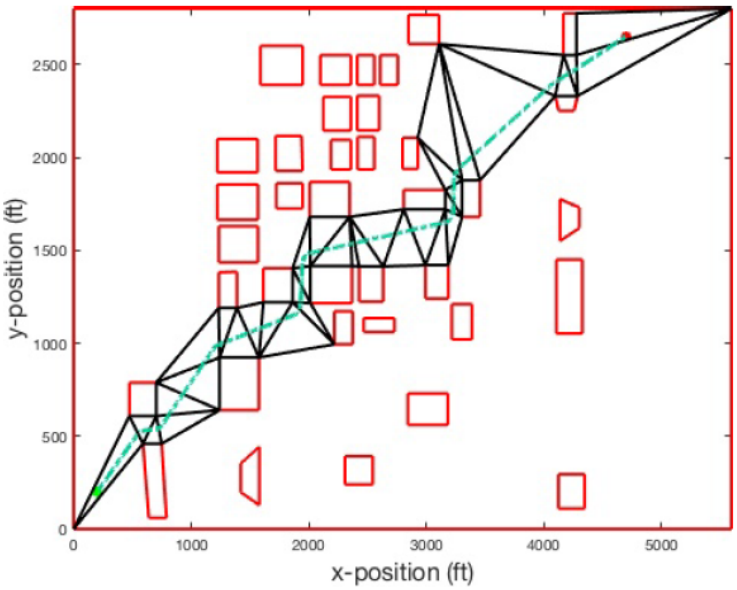


Figure II.10 A Connected Simplex Corridor (CS) [25]

Toward the end of his work, Zollars worked on a 3D path planning algorithm which uses simplexes (or 3d triangles) to discretize the space. He placed nodes at the midpoints of his simplexes in order to use an A* algorithm and find the shortest path in 3D space. The simplexes which the A* algorithm passes through could then be connected to create a CSC. He then utilized barycentric coordinates to identify the space within each simplex, utilized the initial path solution as a guess, and broke the problem into segments to solve for the optimal path. His method proved robust, but was not very fast.

This research seeks to benefit from his findings leverage his CSC method across a contoured surface.

In 2019, Rivera published a thesis at AFIT which found paths for a UAS through a constrained environment for real-time application. He creates a digital map with constraints overlaying the buildings and objects from the overhead imagery. The map is then discretized using constrained Delaunay triangulation. Nodes are placed in the center of each triangle and mid-edges of each triangle. MATLAB's Dijkstra algorithm is used to find the shortest path from the starting node to the finishing node. Due to his map size, he is able to calculate a solution in less than a second. He then utilized the solution as waypoints around the buildings and flew the trajectory. Rivera's work can be found in [5]. Rivera's work provides a 2D solution, this research seeks to produce a 3D solution.

Summary

The optimality of the solution to a path planning problem depends on the discretization of the space. The timeliness of the solution depends on the search algorithm. The timeliness and optimality of the search algorithm depends on the weights of the heuristic. Clearly the discretization method, algorithm, heuristic weight, and method of data management impact the optimality and timeliness of the path solution.

It is proposed in this research that a triangular discretization over a 3D surface can be mapped to a weighted 2D surface. The cost to traverse between nodes can be precalculated for any region, irrespective of the start and goal state positions. The heuristic cost can be added once the goal state location is known. An A* algorithm can be applied to the 2D weighted surface to find a near-optimal path. The corridor created by the triangles which the path solution traverses through in 3D space can be used to identify a corridor to create an optimal trajectory through, allowing for more straight-lined segments and

smoothed path corners. This method should provide an initial guess quickly and a bounded region to solve for an optimal solution, which will allow for a timely near-optimal solution. The work which has been conducted thus far by other researchers provides ample observations and methodologies which build upon.

Next, Chapter III will introduce the methodology which was adopted to solve the timely near-optimal path finding problem.

III. Methodology

Chapter Overview

This chapter presents the methodology used to solve the problem presented in Chapter I. As a solution to the problem proposed is a method which can quickly and robustly calculate a near-optimal solution. The method incorporates constraints into the map through a weighted discretization. An A* algorithm then determines the optimal path through the weighted mid-edge nodes. The mid-edge point path is then refined within the unconstrained corridor to yield a near-optimal feasible solution.

The terrain is modeled by creating a 3D triangular discretized surface which reasonably represents the contours of the terrain. The discretization method creates a surface of triangles which the vehicle may travel along free of constraints. This is similar to an ant traveling along the surface of a blanket laid over a bunch of objects which create humps in the blanket. There are no terrain constraints necessary to consider as long as the travel is along the surface.

The terrain constraints and any additional constraints are accounted for by weighting and positioning of the nodes. Traditionally, modeling terrain in optimization problems yields non-linear constraints, which exponentially increase the amount of time it takes to solve the problem. Solving an unconstrained optimization problem is much faster than a similar constrained optimization problem. Upon every iteration or trial of a solution, the solver must ensure it satisfies all of the constraints. One can easily see how this can become very costly if multiple constraints are checked upon each iteration. Eliminating the terrain constraints and any additional constraints from the problem, especially non-linear

constraints, by accounting for them through a penalty or weighting factor simplifies the problem [9].

An A* algorithm is used to quickly find a path solution across the discretized and weighted map. The A* algorithm will allow for a quick solution along the mid-edge of the triangles, and in turn a quick specification of the flight path.

The initial mid-edge point path solution identifies a connection of triangles referred to as the Connected Simplex Corridor (CSC), which identifies an unconstrained corridor. Once the CSC is defined, the initial mid-edge point solution path can be refined to find the optimal path through the CSC. The time to calculate the optimal path is expected to be much less than traditional methods due to the reduction of constraints and feasible initial path solution, which is close to optimal. The method just described is depicted in Figure III.1, and is further described in detail in this chapter.

Process Description

The process is outlined in Figure III.1 and each step is described in detail later in this chapter. This research uses MATLAB 2018b for programming and execution.

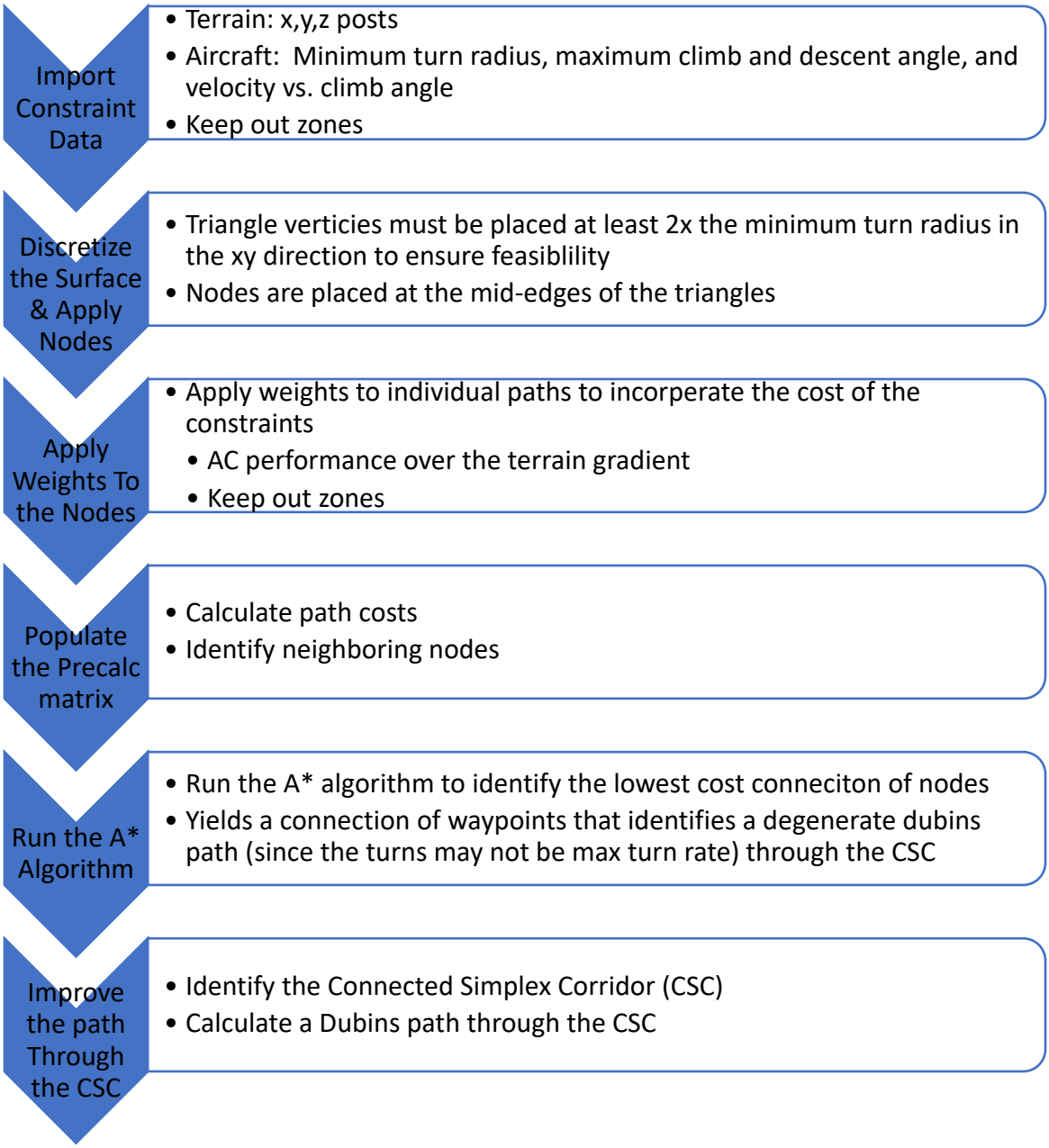


Figure III.1 Diagram of Methodology

1) Terrain Data and Constraints

Prior information about the constraints (such as terrain, buildings, aerial vehicle performance, and keep out zone) is necessary for the proposed algorithm to work in near-real-time. The algorithm pre-computes the costs associated with the constraints in order to allow for a near-real-time solution. The entire process is very fast, relative to traditional optimal path finding techniques; providing known information such as the terrain and vehicle performance beforehand drastically reduces the amount of time required to calculate a solution.

The environmental position and elevation information can be provided through means such as Digital Terrain Elevation Data (DTED) or imagery from an aerial vehicle or satellite which provides 3D information. The data provides the maximum elevation for a given x and y position. Since only the surface data is utilized, this algorithm will not choose paths which traverse under objects, such as a bridge or overpass. Terrain information is used for the discretization, weighting, and path generation steps. This research simulated DTED by using the MATLAB shapes command to produce an evenly spaced 3D mesh of points in the x, y, z reference frame. The Cartesian coordinate frame was chosen for simplicity as this algorithm is mostly for proof on concept.

The following aircraft performance parameters are required to ensure a feasible path: maximum climb and descent angles, airspeeds at various climb angles, minimum turn radius while performing a maximum climb. The parameters directly influence the discretization of the map and the node weights. The minimum turn radius is used in the discretization method, while the other performance information is used in the weighting of the nodes. Other constraints to the flight path, such as the location of keep out zones can

be accounted for by the weighting of the nodes within the specified radius of the keep out zone. Incorporation of the aircraft parameters and constraints are described next in the discretization section.

2) Discretization and Node Placement

Once all of the information about the terrain and aircraft performance is obtained, the first step is to smartly discretize the space into sub-surfaces and place nodes on the surface.

The way in which the space is discretized is a critical part of this research. The discretization of the surface determines the placement of the nodes, and the nodes must be able to be connected and create a feasible flight path within the corridor defined by the discretized segments.

The algorithm developed uses a triangular discretization of the space. Triangles allow for the best representation of complex terrain when compared to other discretization shapes, since the minimum number of points necessary to create a plane in 3D space is three. An example of a triangular discretized surface is shown in Figure III.2.

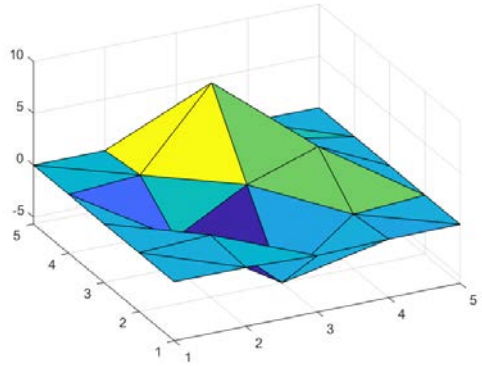


Figure III.2 Triangular Discretization of MATLAB Peaks Terrain

The nodes were placed at the mid-edges of the triangles on the discretized surface so that a path created by connecting the nodes would follow the surface contour and stay toward the middle of the triangles it passes through. The triangles which the path travels through can be connected side by side to create a CSC that is free of constraints. The nodes were placed on the mid-sides of each triangle to ensure this characteristic. An example of mid-edge node placement can be shown in a 2D view in Figure III.3

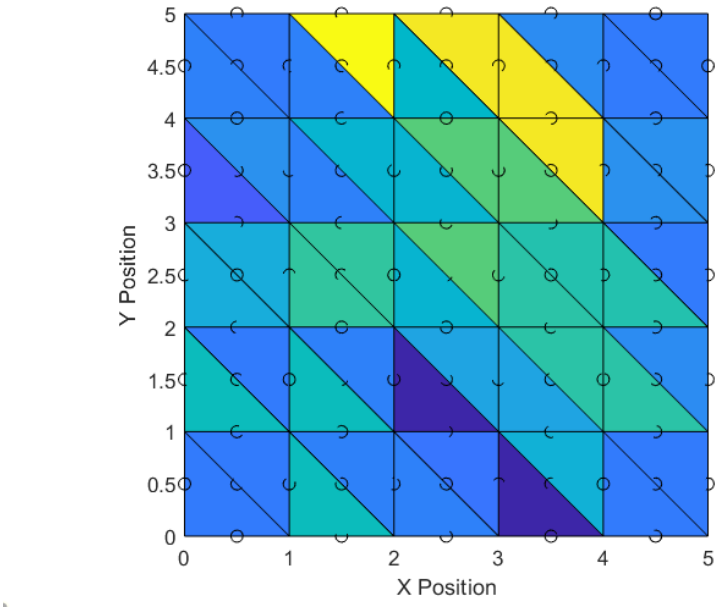


Figure III.3 Discretized Surface with Mid-Edge Nodes

Nodes are restricted to connecting to their immediate neighbors on adjacent triangles to ensure the path follows along the contour of the surface and does not cut through or skip over the top of the surface. Neighboring nodes are those which the node in reference shares a triangle with.

2.1) Spacing Between Vertices

It is desirable to use the coarsest discretization (or largest triangles possible) which accurately represents the space. This will allow for a minimum set of triangles to create a corridor and path. The spacing between the nodes should be influenced by the aircraft performance and the terrain.

The aircraft performance influences the discretization by requiring the vertices of the triangles to be separated by a distance greater than or equal to 2 times the minimum turn radius of the aircraft.

$$\tilde{r} = \frac{v}{\dot{\theta}_{max}} \tag{3.1}$$

The minimum turn radius $\tilde{r}$ is calculated by dividing the velocity v by the maximum turn rate $\dot{\theta}_{max}$. This ensures that the aircraft can make a feasible turn through the triangle no matter the aircraft orientation angle when entering the triangle. This is shown in Figure III.4.

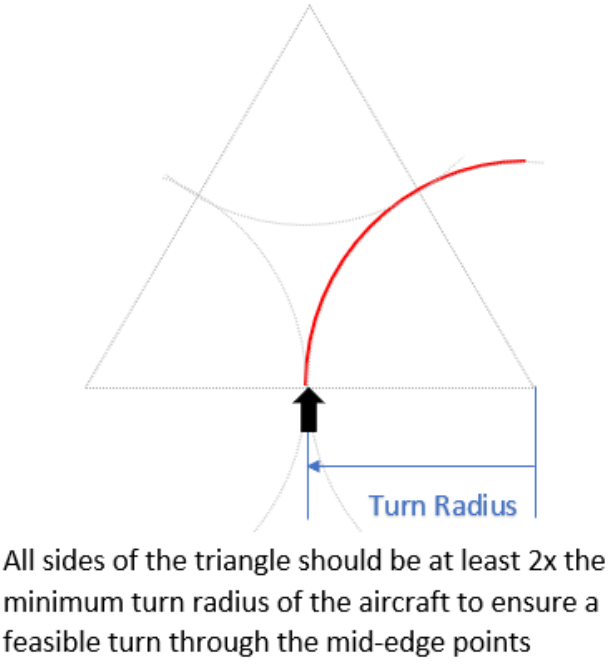


Figure III.4 Triangle Edge Sizing to Ensure Feasibility

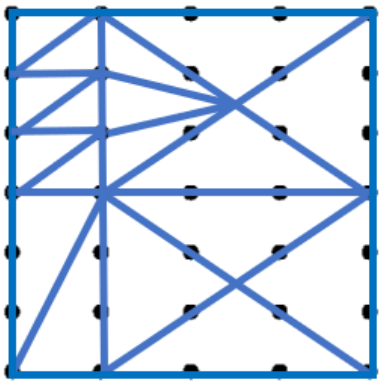
Only uniform triangular discretization was used for this research due to research time constraints; however, a non-uniform discretization can still yield a feasible discretized space as long as the spacing between the vertices abide by the turn radius constraint requirement. It would be beneficial to implement a non-uniform discretization in regions where the gradient exceeds the maximum aircraft performance and to represent the differences in terrain complexity. Implementing a non-uniform discretization in regions which the gradient exceeds aircraft performance capabilities would enable the algorithm to find paths which zig-zag up the side of the mountain, much like most roads (switchbacks) are designed on the side of a mountain.

Implementing a non-uniform discretization to better model regions of varying complexity would enable more efficient paths around the terrain. For the constant velocity

case, modeling a simple terrain region with a coarser discretization is beneficial, since it reduces the number of nodes in the search space, thus reducing the number of path options. For the varying velocity case, it is beneficial to model areas with coarser and finer discretization, appropriate to the terrain complexity. The discretization could be made finer for regions in which there is significant terrain complexity and slowing down could yield an improved path. An example of such a terrain is an urban environment, where buildings may be closely spaced together and flying up and over the buildings is more costly than slowing down and going between buildings.

The only requirement of the discretization for the method used in this research is the minimum post spacing to handle the aircraft turn radius, as well as the alignment of mid-side edges of the triangle.

Figure III.5 shows an example of a varied triangular discretization in which the upper left portion of the map may contain a more complex environment and the right half of the map contains less complex terrain where larger flat segments can be used to represent the terrain. The size of the triangles in the discretization are proportional to the minimum aircraft turn radius which is required to ensure a feasible path. The region in Figure III.5 required the lengths of the sides of the small triangles to be at least twice the minimum turn radius of the aircraft. The minimum turn radius to ensure a feasible maneuver does change from triangle to triangle in Figure III.5; however, as long as the most restrictive turn radius is achievable, then the entire region is feasible.



This method requires that the mid-side edge point of the discretized triangles edge be shared (In other words, each triangle must share two vertices with the neighboring triangle)

Figure III.5 Example of a Varied Discretization Size

2.2) Downsizing High Fidelity Data

Terrain data provided at a higher fidelity than necessary can be simplified to provide only the level of fidelity at which the data is useful based upon the vehicle's performance. For example, if data were given at increments of .5 in the 2D plane, but the aircraft could only perform a turn with a radius of .4, then the spacing between vertices should be .8 or greater to ensure feasibility. In this example, a spacing of 1 between vertices would be chosen since it is the next largest spacing. If a vertex spacing of 1 is chosen, then there exists the possibility that the nodes between the vertices protrude through the surface created by the vertices spaced apart a distance of 1.

This algorithm ensures that any points between the vertices of the triangulated surface are present on or below the surface. If a point were to protrude through the surface created by the new vertex spacing, then the algorithm raises the elevation of the entire surface the

distance of the point with maximum protrusion. The resulting surface is shown in Figure III.6

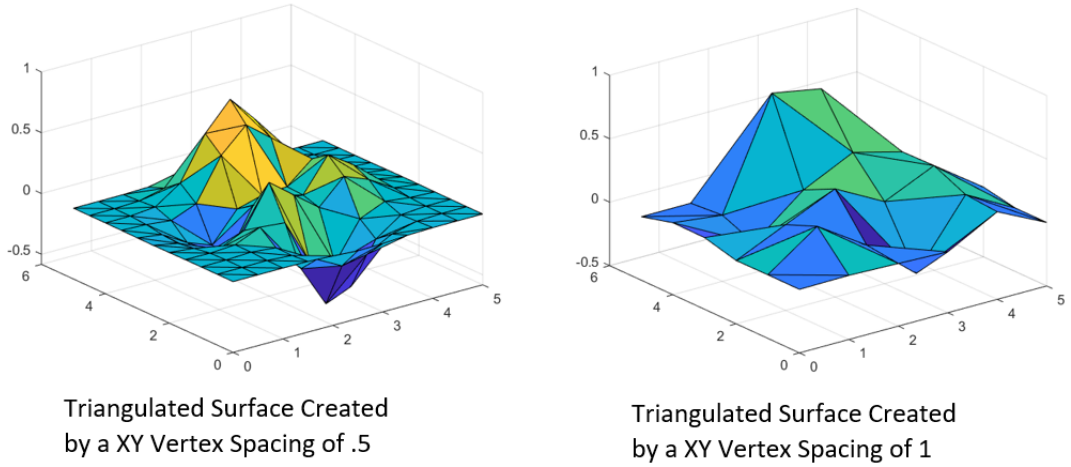


Figure III.6 Surfaces of Different Vertex Spacing

The ability to reduce the high-fidelity terrain information to the level usable by the vehicle reduces the time to calculate a solution.

2.3) Node Placement

Nodes were placed at the mid-edge points on the sides of each triangle which make up the discretized surface. This ensures clearances on either side of the path and ensures the path follows the contour of the triangles and does not cut through the surface. Additionally, the number of neighboring nodes from the mid-edge position is consistently 4.

If nodes were placed on the vertices of triangles, then the path between nodes would follow along the edge of two triangles. The question would then become, “which triangle is chosen to create the CSC?” If both triangles were chosen to create the path, then the CSC would have increased complexity because the path could look like a path with triangles attached to the sides of it. A more complex corridor would increase the computation time

and the solution is unlikely to be improved. Additionally, the number of triangles which can connect to a vertex is not consistent. Thus, the number of neighboring nodes to consider when expanding a node is not consistent. Three triangles or any number of triangles can be used to model a hill where the peak of the hill is the vertex of the triangles.

Also considered were nodes placed in the center of the triangles. Such placement could create paths which protrude through the constrained surface when nodes on either side of an elevated area were connected. For this reason, this node positioning was not chosen.

2.4) Node Numbering Scheme

Numbering of the nodes is needed in order to easily save the information about each node in a vector format. This is similar to finite element schemes which label each node with a number.

A node numbering scheme was needed to provide directionality information about the path, since paths can lead to or away from a point. The numbering allows for the algorithm to quickly identify the triangle which it just came from and its two neighboring nodes on the triangle which is to be expanded. Each node has four neighbors, but only two of the four neighbors need to be opened upon expansion. The algorithm works most efficiently if it quickly knows what two of the four nodes are on the new triangle being expanded. An example of the node numbering can be shown in Figure III.7.

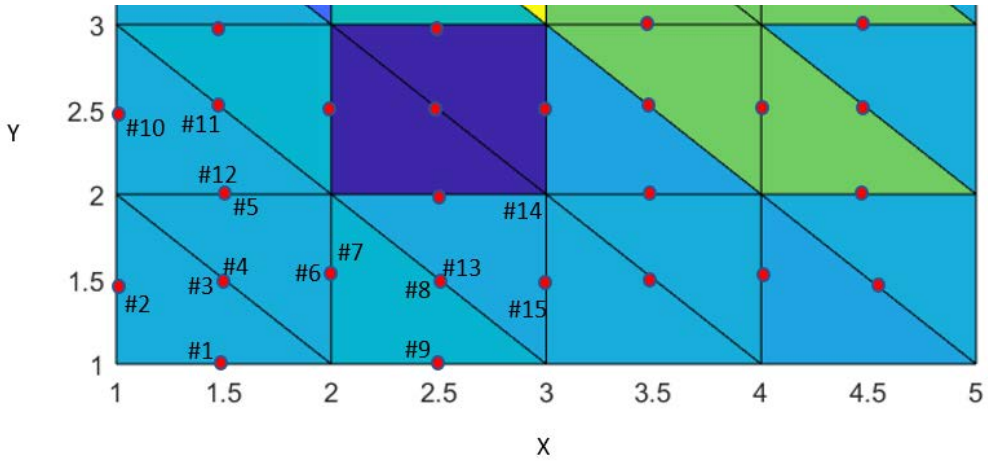


Figure III.7 Map Showing an Example of the Unique Node Numbering

For the example shown in Figure III.7, if the algorithm was propagating from the lower left of the map and it arrived at node 3 from node 1, then upon the next iteration, the algorithm would refer to itself as being at node 4. Identifying it as node 4 allows the algorithm to recognize that the neighbors of the newly opened point/node are nodes 5 and 6. When the algorithm reaches a point, it refers to itself as the point number on the far side triangle so it can identify its neighbors on the far triangle.

2.5) Expanding a Node

Due to the way the A* algorithm expands the lowest cost nodes first and the fact that traversing between two nodes cannot yield a negative cost (the path cannot cost negative time), it is known that the path from point 1 to 3 is cheaper than the path from point 1 to point 2 to point 3, no matter the orientation of the triangle. Thus, there is no need to calculate the path from 3 to 2, since it will not be the cheapest path to point 2.

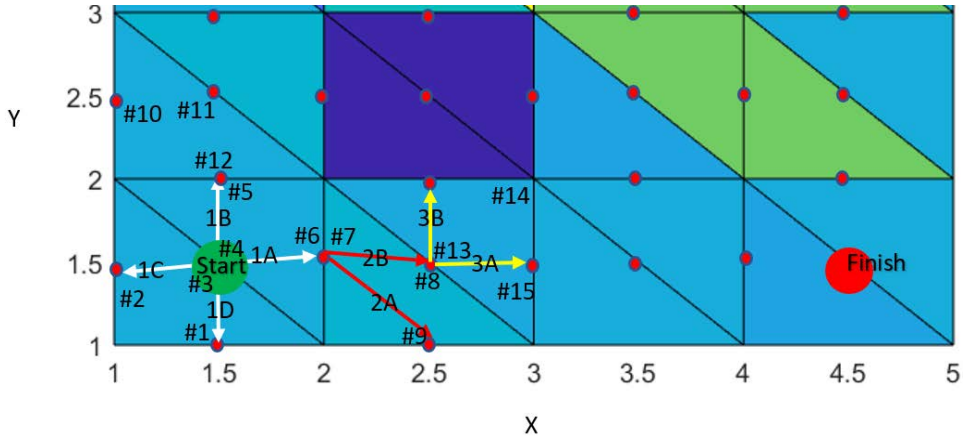


Figure III.8 A Node Expansion Example

An example can be seen in Figure III.8 where each arrow signifies the path for which the cost is being calculated during each expansion. The second expansion (shown in red) does not evaluate the cost to go to nodes 4 or 5 from node 6. Since the cost to go to nodes 5 and 6 were already considered by iteration 1, their costs do not need to be calculated on expansion two since the cost to go to node 5 from node 4 on the first iteration will always be less than the cost to go to point 6 then point 5, due to geometry.

Each node has four neighboring nodes due to the discretization, unless the node is on the edge of the map. With the exception of the starting node and nodes on the edge of the map, each expansion consistently calculates the cost of two nodes.

3) Node Weighting

Weighting of the nodes is needed to account for the characteristics of the 3D environment and constraints.

Each node is weighted appropriately to account for aircraft performance across the terrain between nodes, aircraft performance limitations, and keep out zones. The base cost to travel from one node to another is the Euclidean distance between the two nodes being

evaluated. Each node saves the cost to go to its two neighboring nodes. The base cost is then multiplied by an aircraft performance weight to yield the aircraft performance cost. The aircraft performance weight accounts for the aircraft speed between the two nodes. The cost to travel up a slope will be more than the cost of traveling straight and level. Similarly, the cost to travel down a slope will cost less than the cost to travel flat and level. The increased cost of climbing at a specified angle and distance is not negated by the benefit received by descending a slope of the same characteristics. In other words, the net penalty for climbing and descending over a specified distance is more than if the distance was flown straight and level. The penalty increases as the terrain gradient increases. Similarly, the terrain penalty reduces for decreases in terrain gradient. The base cost multiplied by the aircraft performance weight yields the performance cost of the segment.

Given aircraft data which shows aircraft velocity versus climb rate, the aircraft climb angle and thus the weights of the paths can be appropriately accounted for. Table III.1 shows an example of the aircraft performance weights applied to the paths in this research. The weights penalize climb angles and reward descent angles; however, the weights are not equal in magnitude so the descent will not negate the cost of a climb for the same distance.

The aircraft climb or decent angle in this research is synonymous to the angle generated by the terrain, since it is assumed the aircraft will fly relatively parallel to the surface of each segment of the contour. The use of climb and decent angles are used in this research since it is easier to visualize constraining the angle of the surface rather than the rates.

Table III.1 Example Aircraft Performance Weights

Climb Angle	*Path Weight*
-20	0.80
-10	0.90
0	1
10	1.20
20	1.40
30	1.60

Terrain gradients which exceed the performance capability of the aircraft will be assigned a weight of infinity to ensure the A* algorithm will never choose the path. The rate of the penalty is not continuous, since it jumps to infinity once the maximum gradient is exceeded, but this closely mimics aircraft stall characteristics. Assigning a cost of infinity eliminates consideration of a path which exceeds the performance capability of the aircraft being chosen.

The heuristic cost, the Euclidean distance between the node in question and the goal state node, is added to the total cost. The nodes which lie within keep out zones will have a weight of infinity associated with them to ensure the cost is high enough so that the A* algorithm does not choose to use the node. Initially, closing the nodes which exceeded the aircrafts performance was tried, but this caused the secondary path from the node to be closed, no matter if it exceeded the performance or not.

The final algorithm calculates the total cost to travel between all nodes while considering all of the constraints. The total costs are saved to a matrix for quick reference when searching for the lowest cost path.

4) Pre-calculation Matrices

The ability of this method to quickly calculate a solution would not be possible without creating a matrix which stores the precomputed costs to travel between all nodes in a matrix for quick reference. Some of the data saved associated with each node contains the cost to travel to its neighboring nodes. The algorithm can quickly reference different parts of the matrix to connect nodes in a manner which will result in the lowest cost path.

It is reasonable to assume that the need to pre-calculate the contributions of the terrain and constraints of the environment will still afford relevancy of this algorithm because it takes time to move a piece of equipment, such as an aerial vehicle, into an area where it will be employed. The time spent moving the vehicle to a new operating area can be used to download the terrain and constraint information and pre-calculate costs for the region. Loading detailed maps for specific regions of operation is commonly done on manned aircraft today.

4.1) Pre-calculation Matrix

The pre-computation consists of calculating the total cost to traverse between all nodes, which incorporates the cost of all constraints (terrain distance, aircraft performance across terrain, and keep out zones) and the heuristic cost. Each node saves information about itself, its neighbors, and their costs. The terrain and constraints make up the majority of the calculation percentage. Once an end state is known, the heuristic cost for each node can be made which is the last of the calculations necessary for all the calculations on the map. Notice that all calculations are agnostic to the start location on the map. Figure III.9 describes the data matrix necessary for completion of the algorithm.

		# of nodes (N)							
1	Point #	1	2	3	4	5	6	...	N
2	Pt#1 in tri	2	1	1	5	4	4	...	...
3	Pt#2 in tri	3	3	2	6	6	5	...	...
4	$ Pt#1 in tri	0.05	0.08	0.03	0.10	0.00	0.04	...	...
5	$ Pt#2 in tri	0.07	0.09	0.01	0.07	0.03	0.10	...	...
6	Pt# of neighboring tri	6073	6078	6078	8400	8377	8377	...	...
7	Pt# of neighboring tri	6119	6119	6073	8393	8393	8400	...	...
8	Neigh Tri of Pt#1	2025	2026	2026	2800	2793	2793	...	...
9	Neigh Tri of Pt#2	2040	2040	2025	2798	2798	2800	...	...
10	Tri of current point	1	1	1	2	2	2	...	...
11	Heuristic$ Current Pt# in tri	3.68	3.67	3.71	1.05	1.10	1.05	...	...
12	Heuristic$ diff to Pt#1 in tri	-0.01	0.01	-0.02	0.05	-0.05	0.00	...	...
13	Heuristic$ diff to Pt#2 in tri	0.02	0.04	-0.04	0.00	-0.05	0.05	...	...
14	Cost-to-go	Inf	Inf	Inf	Inf	Inf	Inf	...	...
15	Parent	0	0	0	0	0	0	...	...
16	Open	0	0	0	0	0	0	...	...
17	Closed	0	0	0	0	0	0	...	...
18	Goal	0	0	0	0	0	0	...	...

Figure III.9 Pre-calculated Matrix Column Format (One Column per Node)

The pre-calculation matrix contains a vector of stored data for each node on the map. Nodes are referred to as points in the code. Rows 2 and 3 indicate the other two points which lie on the same triangle that the referenced node is on. Rows 4 and 5 contain the total cost to travel to the neighboring points. Rows 6 and 7 contain the point numbers which the points in row 2 and 3 are also known as when the algorithm is at that location and propagating away from the current triangle and point number. Once the lowest cost is selected, (say point 1 is the lowest cost) then the point which point 1 is also known as will be opened. This allows the algorithm to know which triangle, and therefore which nodes, to expand next. Rows 8 and 9 contain the triangle numbers of the two neighboring triangles. Row 10 contains the triangle number of current point being evaluated. Rows 11-13 contain

the heuristic costs of each point. This cost is already included in the total cost in rows 4 and 5 but was kept in the matrix for possible future work such as the ability to easily change the end location or work with a moving target. Row 14 contains the cost-to-go which is the lowest cost to go to the associated point. Row 15 contains the parent of the current node which yields the lowest cost-to-go. Row 16 indicates if the node is open and the cost is being considered upon each iteration. Row 17 indicated if the node is closed. A node is closed when its neighbors and the node itself have been evaluated. This reduces the number of nodes the algorithm needs to consider during iterations. Row 18 indicates if the node is the end goal, so the algorithm knows when to stop the search.

4.2) Waypoint Node Matrix

Since flight through certain nodes which lie directly on the constraint surface would result in penetration of the constraint surface, a feasible waypoint for the aircraft to fly can must be created above the node. An aircraft will not be able to fly along the surface and through a node recessed between two surfaces on the map; therefore, feasible waypoints must be created above the recessed nodes. This will allow the initial A* solution to utilize the waypoint nodes above the recessed nodes to generate a flyable path.

The nodes which do not lie in a recessed location with respect to its two neighboring surfaces will retain their original waypoint location. Nodes which lie in a recessed location will be assigned a waypoint location which is above the location of the constraint surface node. By utilizing the combined minimum pitch/turn radius of the aircraft and the path at each node with the steepest gradient, one can calculate the height above the recessed node which to place the waypoint node to allow for a feasible flight over original node location. The waypoint nodes are numbered the same as their nodes on the constraint surface.

5) Heuristic Search Algorithm (A*)

Since the constraints of the problem are handled by the node creation process and the weighting, the search algorithm can now quickly calculate a path by simply connecting the nodes which yield the lowest cost. An A* heuristic search algorithm was used to evaluate the nodes in the space in order to determine the cheapest path solution without requiring a search of the entire space. The heuristic used in this algorithm was the Euclidean distance. A heuristic can be multiplied by a weight factor to magnify its effect, but a heuristic weight of 1 is used in this research to ensure the effect of the heuristic is present, but mild – meaning that it does not seem to overpower the cost of the terrain and yield virtually straight line paths.

The A* algorithm first starts with an array of nodes which store all the costs to traverse to their neighboring nodes. From the starting position, the A* algorithm picks the neighboring node which has the lowest cost to go.

The unique discretization method allows for each iteration of the algorithm to expand a node and only open and consider the cost of two new nodes at a time. This is different from other discretization methods which may cause the number of nodes considered upon each node expansion to vary. A varying number of nodes which require opening after a node expansion would result in an inconsistent number of data elements saved per node, and would require additional checks in the algorithm; thus, complicating the algorithm and increasing the computation time. The consistent opening of two nodes for each node expansion allows for the algorithm to run more efficiently.

The algorithm's node expansion process in this research is relatively simple due to the discretization and numbering technique. Each node expansion (besides the first and last

iteration, where the starting point or ending point may not lie at the exact mid-side node of the triangle) is only required to calculate the cost of two paths. For each expansion, the algorithm only calculates the cost of the two nodes on the triangle it just connected to. This is because the other two of its four neighboring nodes it can connect to are on the triangle it just came from. The two nodes on the triangle that the algorithm propagated from are node that it just came from and the node it calculated the cost for on a previous iteration; therefore, it does not recalculate their cost.

Figure III.10 shows an example of the expansion process described above, where expansion 1 is in white and expansion 2 is in red. Expansion 1 considers four points because it is the starting node. Each consecutive expansion only calculates the cost of its two nodes on its current triangle. Expansion 2 does not consider the cost to go to points 5 or 6 because it already calculated the lowest cost to get to those nodes during the previous iteration.

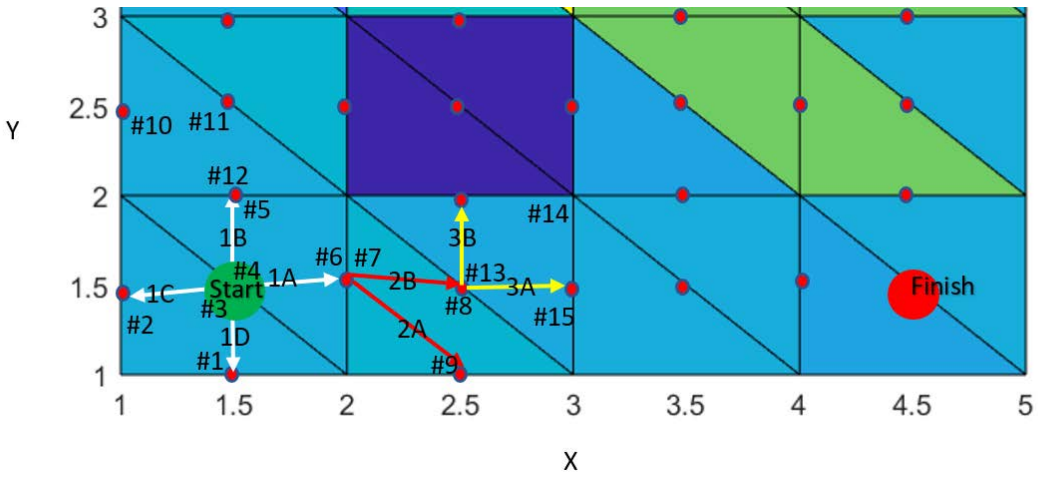


Figure III.10 An A* Node Expansion Example (Same as Figure III.8)

More information about the A* algorithm and heuristic path planning can be found in [4]. The general process of the A* algorithm is the following: Starting from the start position, the algorithm finds all the starting position's neighboring nodes and puts them on the open list. The algorithm expands the lowest cost node which is on the open list. Expansion of a node is the process by which the algorithm considers all the neighbors of the said node, which can also be viewed visually as a branch expanding out to all neighbors of a point. The newly discovered neighboring nodes that are not on the closed list are put on the open list. Upon the next iteration, the algorithm expands the lowest cost node off of the open list. After a node has been expanded, it is listed as the parent node of the newly found neighboring nodes and is put on the closed list to prevent the cost of node from being calculated again. The process of selecting nodes off the open list and expanding them repeats until the goal node is reached. During each expansion iteration, each node stores the node number of its parent node (the node which the algorithm propagated from to get to it). Once the goal node is reached, the parent of each node, starting from the goal node, can be used to identify the lowest cost path which connects the start and finish node.

6) Establishing the Initial Path

The initial path will consist of connecting the waypoints which yield the lowest cost path across the surface, as identified by the A* algorithm solution. The lowest cost path can be found by tracing back the parents of each consecutive node, starting with the ending node. Each node will point back to the parent node which yielded the lowest cost to go. The set of waypoints within the waypoint matrix contain feasible waypoints for each corresponding node. Together these waypoints create a degenerate Dubins path, since the

turn between each waypoint may not necessarily be a maximum rate turn. This solution is sufficient for most UAS planners.

7) Defining the Connected Simplex Corridor (CSC)

The triangles which the initial path passes through can be connected to form a CSC. Figure III.11 shows a discretized surface with an initial path through the nodes which identify the CSC. The CSC identifies a path of connected triangles (aka simplexes) which represents an unconstrained "corridor" in which the vehicle can operate as long as it stays on the surface constraint.

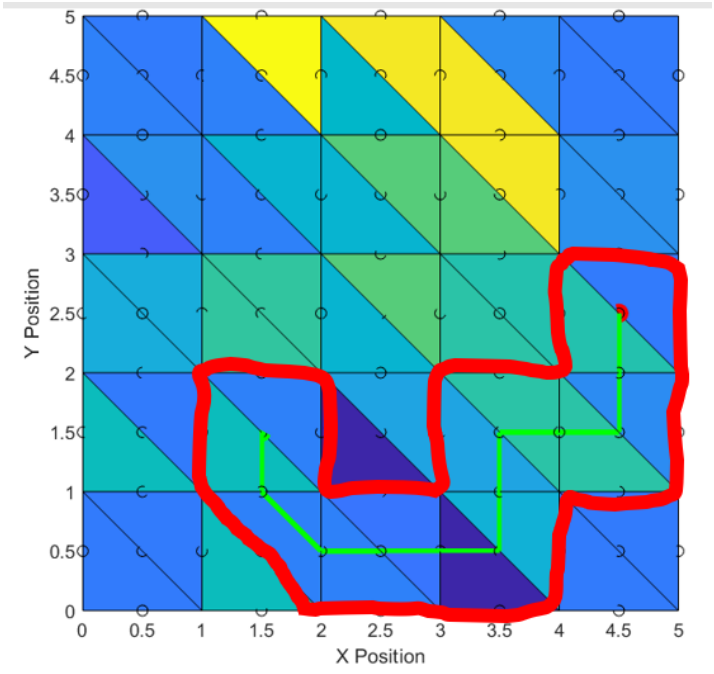


Figure III.11 An Initial Path Identifying A CSC

8) Optimal Path Through the Corridor

The initial feasible waypoint attained by the A* solution and the identification of the CSC is likely sufficient for most UAS planners. However, the initial feasible A* solution can be further improved within the CSC to yield an improved path. An optimal path through the corridor can be determined by calculating a Dubins path over the surfaces within the CSC. Figure III.12 shows an urban environment with a Dubins path that maintains a minimum clearance distance through the CSC.

The path waypoints may be provided which identify different points along the path. Since most UAS utilize waypoint navigation or are easily capable of such, waypoints which identify the start and stop points of each turn along the Dubins path are appropriate.

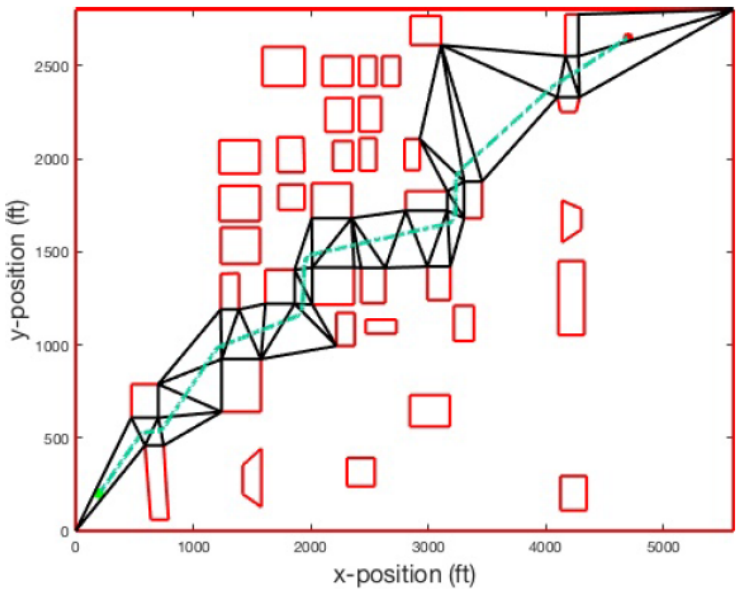


Figure III.12 A Connected Simplex Corridor (CSC) [25].

Summary

This chapter provided an overview of the methodology used in this research. The discretization process breaks down the large and computationally expensive characteristics of the map into subsets. Constraints from the terrain, vehicle performance capability, and other sources (keep out zones, etc.) were considered when determining the weighting of each node. A heuristic algorithm was used to define the optimal path across the nodes and identify a Connected Simplex Corridor (CSC). The path can then be refined within the CSC, where it is free of environmental constraints, to yield a near-optimal path.

Chapter IV will present simulations of different scenarios. Parameters such as the discretization level, heuristic weighting, starting and ending positions will be varied.

IV. Simulations

Chapter Overview

The purpose of this chapter is to display some of the results attained by utilizing the algorithm proposed in the methodology. Simulations which compare performance against the Dijkstra solution, solutions through mountainous regions, and solutions around keep out zones were explored.

The coloring of the surface has a color gradient which is based on the elevation of the surface, as shown in Figure IV.1. The MATLAB peaks command with a .1 weighting on the z component was used to create the surface used for the simulations and shown in Figure IV.1. A finer discretization was used in this section than may have been shown in previous illustrations in order to simulate a larger environment and more path options.

After the algorithm is run, the nodes on the surface are colored so that the final path, open nodes, and closed nodes can be distinguished. The number of open and closed nodes indicates the level of effort the algorithm spent finding the path. The nodes of dark blue color indicate closed nodes, while nodes of light blue color indicate the open nodes and appear on the edges of the closed node regions. Red nodes indicate a keep out zone.

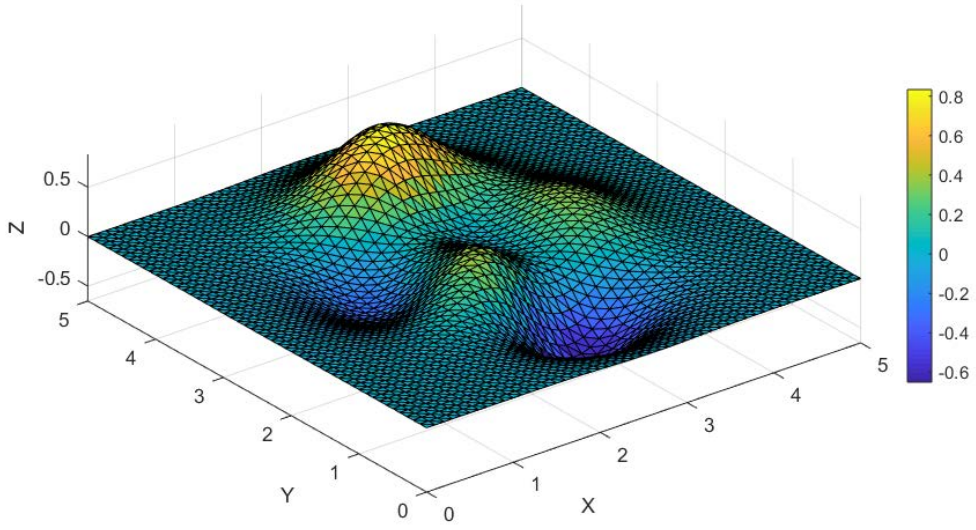


Figure IV.1 A Fine Discretization of the Surface (2,500 points)

Simulation of Dijkstra Solution vs. A* Solution

A solution from the A* algorithm was compared to the Dijkstra solution for the same inputs in order to judge how near-optimal the A* solution is. The comparison results were as expected, the A* algorithm finds a solution in less time, but the solution is sub-optimal. Figure IV.2 shows a comparison between the two solutions, where the A* algorithm finds a solution 15x faster than the Dijkstra algorithm, with only a 7% increase in path length.

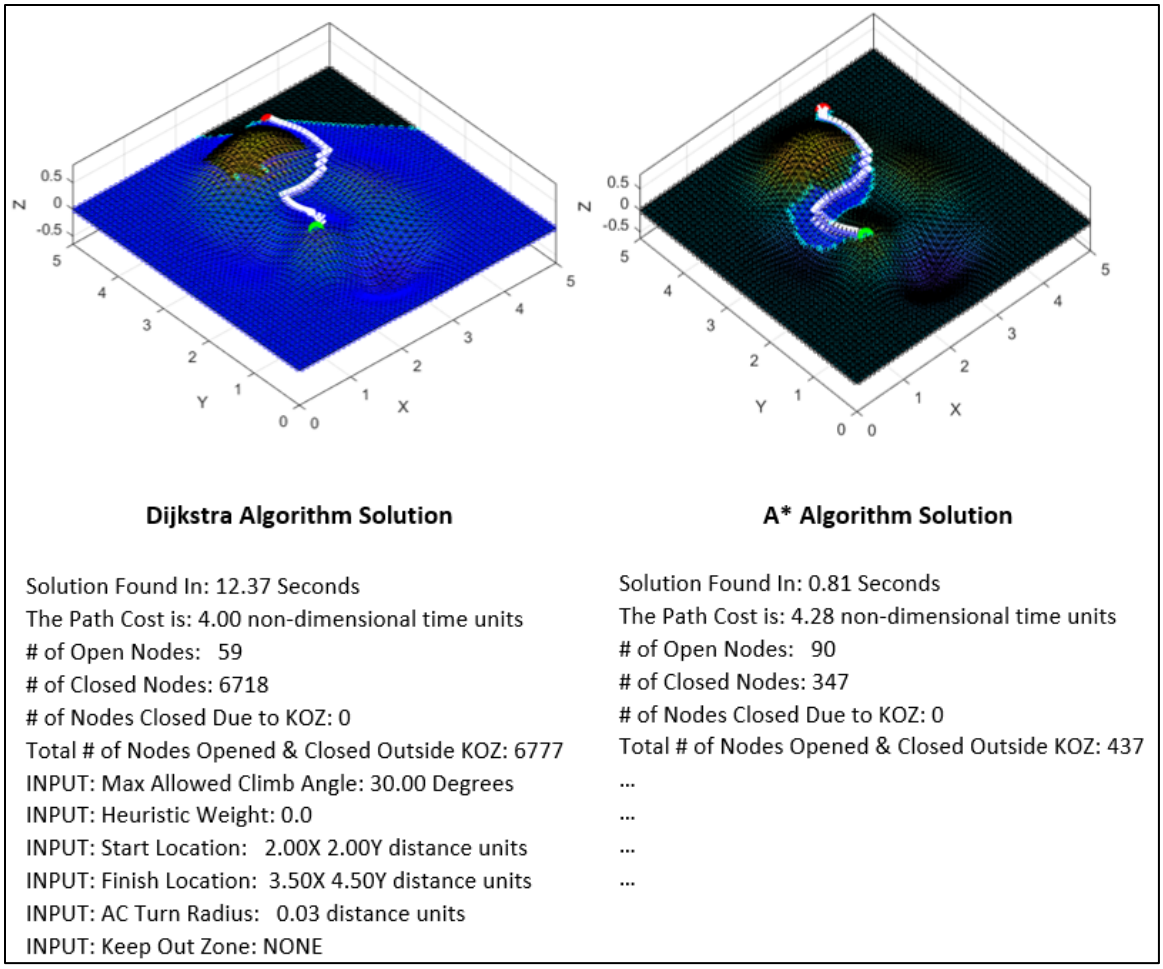


Figure IV.2 Simulation of Dijkstra Solution vs. A* Solution (3D view)

Figure IV.3 shows a 2D view comparison between the two solutions which more clearly shows the small path difference between the solutions. The blue nodes indicate the expanded region of the map. It can be seen that the Dijkstra solution expanded a much larger region than the A* solution. The difference in sub-optimality of the A* solution will vary for different environments and constraints. Thus the difference between the A* and Dijkstra solution will vary, but typically results in a similar result where the Dijkstra solution takes longer to find than the A* solution.

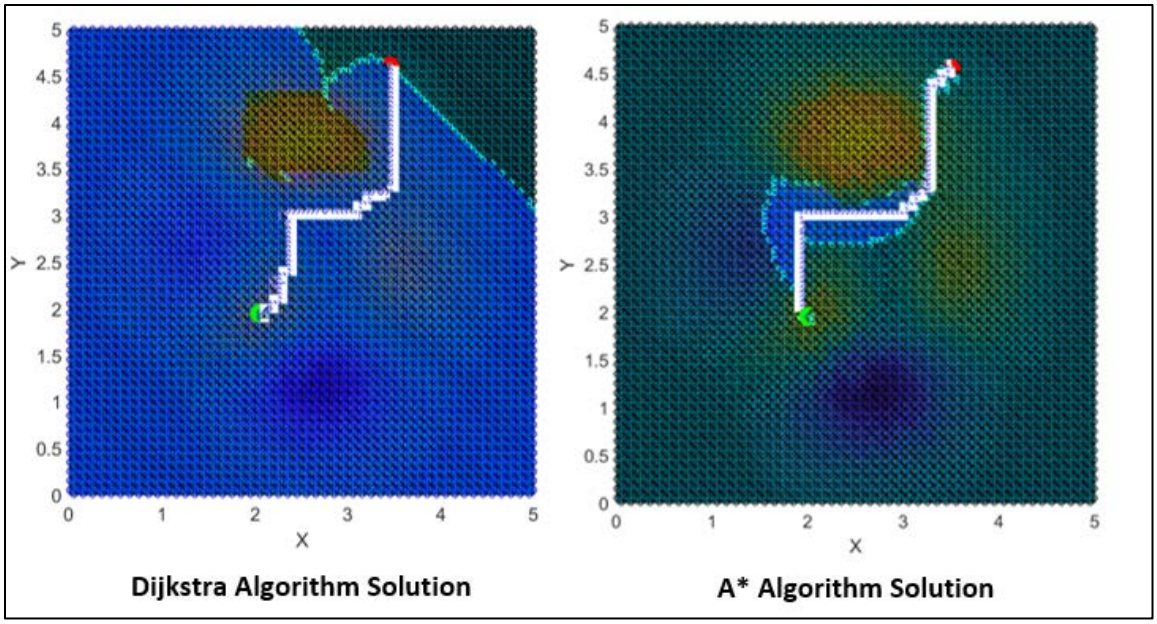


Figure IV.3 Simulation of Dijkstra Solution vs A* Solution (2D view)

Simulation Avoiding Region Which Exceeds Max Climb Angle

The purpose of this simulation is to show that the algorithm is able to incorporate maximum climb angle constraints. Figure IV.4 below shows the different solutions based upon the maximum allowable climb angle.

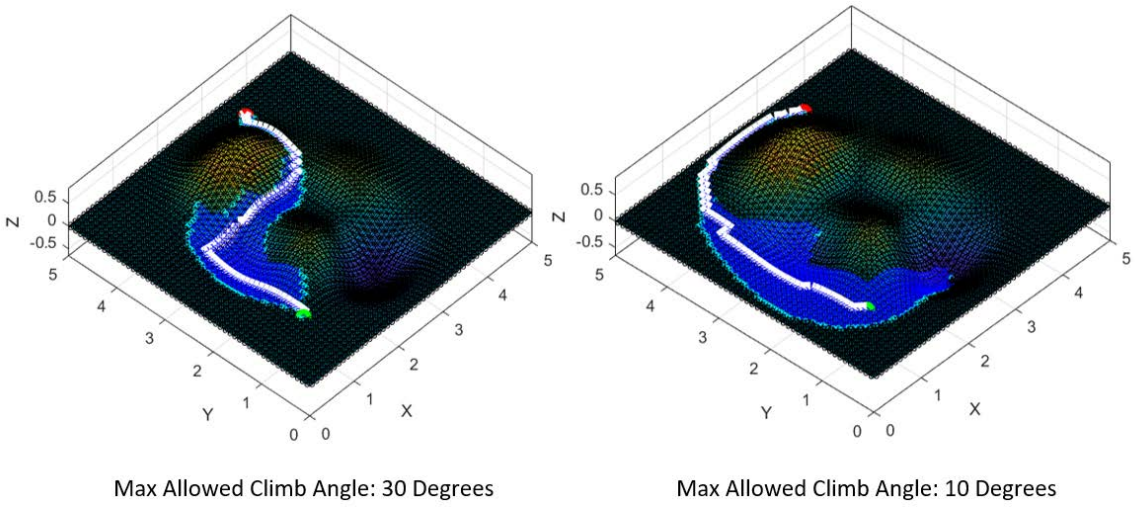


Figure IV.4 Simulation Avoiding Region Which Exceeds Max Climb Angle

When the maximum allowable climb angle is reduced, the algorithm avoids the mountainous region to stay within the performance constraints, as shown by the blue expanded regions of the map.

Simulation of Keep Out Zone Avoidance

The simulation shown in Figure IV.5 is the same setup as Figure IV.4 with the addition of a keep out zone intended to overlay the region of the original solution.

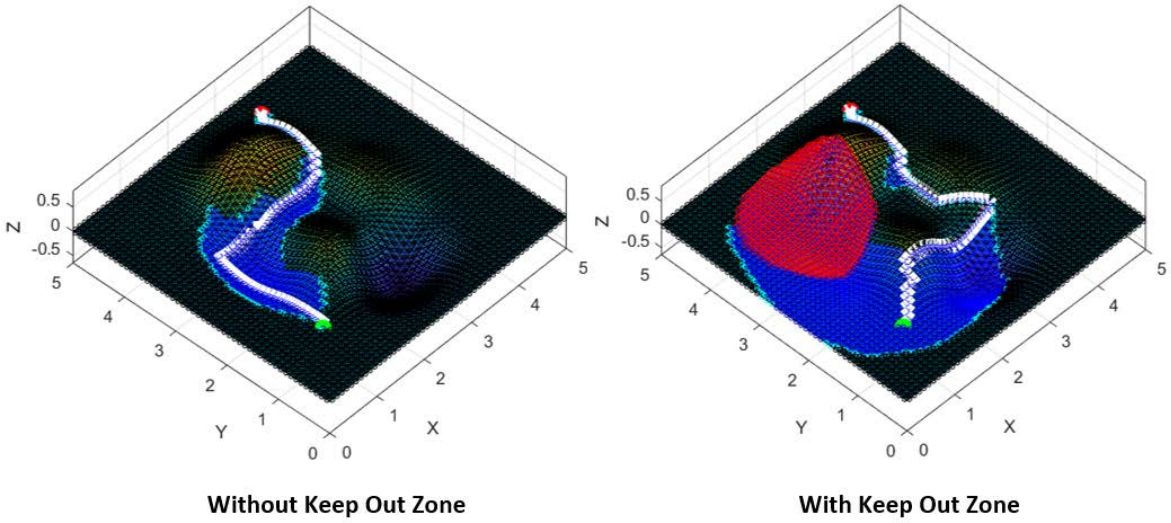


Figure IV.5 Simulation of Keep Out Zone Avoidance

The path successfully avoids the keep out region. The path initially leads to the right because of the lower elevation and thus lower cost in that region.

Summary

The algorithm successfully navigates around infeasible regions, such as keep out zones and those which exceed the maximum climb angle, while quickly finding a near-optimal solution. A solution was found in less than a second for the tested scenario, which is 15x

faster than the Dijkstra solution. Solution times will depend on the start and finish locations and the constraints imposed.

A couple of downfalls of the algorithm are evident in the simulations. The algorithm tends to spend effort exploring large areas of the lower cost regions, such as the valley regions. It is also known, and is evident from the simulations, that the A* algorithm takes more of a straight-line path as it nears the goal. This characteristic exists because the heuristic contribution to the total cost is more sensitive to changes in direction as the algorithm nears the goal since a small change in the leg of a triangle effects a small triangle more significantly than a large triangle.

Run times may differ depending on the computer used. A mid-tier performance laptop was used to run the algorithm which is coded in MATLAB 2018b. The laptop is a Dell Inspiron 15-7569 with an Intel(R) Core (TM) i5-6200U CPU @ 2.30GHz processor, and 8 GB RAM.

V. Analysis and Results

Chapter Overview

This chapter will analyze and discuss the results of the proposed methodology by answering the investigative questions. The simulations show the ability of this methodology to quickly calculate solutions for the size of the environment analyzed. A near-optimal solution is attained in fractions of the amount of time required for the optimal solution.

Investigative Questions Answered

The investigative questions poised in Chapter I are described in detail throughout this chapter and are listed below:

i. How should the discretization technique scale (temporal & spatial) based on terrain?

ii. How should the discretization technique scale (temporal & spatial) based on vehicle performance (caused by size/speed/maneuverability)?

iii. What method is the best for finding an optimal path through the equivalent 2D discretized surface?

iv. What are the tradeoffs of this proposed method as compared to alternative methods in use and proposed in the literature?

1) How Should the Discretization Technique Scale (Temporal & Spatial) Based on Terrain?

The discretization of the map should be scaled based upon the terrain gradient of the region. The terrain gradient will influence the 3D Euclidean distance between two

points. A set of points separated by a specified 2D distance will have a greater 3D distance between them if there exists a difference in the third dimension (elevation). The 3D Euclidean distance between the vertices of the triangles should be at least two times the minimum turn radius of the aircraft to ensure aircraft turn feasibility of any arbitrary connection. This will guarantee a path which is feasible maneuverability path across the selected surface.

During the development of the algorithm, the terrain was discretized using only the XY 2D Euclidean distance between nodes, as opposed to the 3D distance, due to research time limitations. The complexities of discretizing the space based on the 3D distance is described below. This choice is allowable since the XYZ distance is guaranteed to be equal to or greater than the XY distance. Therefore, as long as the XY distance between nodes is at least two times the minimum turn radius in order to satisfy the turn feasibility requirement, then the XYZ distance will satisfy the length requirement.

As trivial as it sounds, discretizing the space based on the 3D distance between triangle vertices proved to be a bit of a challenge; therefore, the 2D distance was used in this research. MATLAB has a built-in function called Delaunay which creates a 3D triangulation mesh; however, the built- in function does not include the ability to ensure a specified distance between nodes, which is a fundamental requirement to ensure feasibility. It was found that writing the code to do this is challenging as it requires a lot of checks- a couple of which were ensuring points are not within an already selected triangle and ensuring the points being evaluated do not have a connection from a previously created triangle between them. Additionally, it was not trivial how to ensure that all the map was

included and no small segments around the edges were left out. Code which uses the 3D distance would likely provide minimally better path options.

High fidelity data about the terrain can always be downgraded to the level appropriate for the map and vehicle. The terrain data must be sized appropriately so that the aircraft can maneuver closely to the terrain. Including small features in terrain which the aircraft cannot closely maneuver around is of no benefit to the map, since a large enough surface which covers the constraint would need to be used. Conversely, terrain data which lacks the fidelity necessary to identify reasonable features of the environment and fails to provide spacing proportional to the performance capabilities is likely to have a significant effect on the optimality of the solution.

Initially, it was thought to be useful to have a discretization which would make the entire surface feasible from all directions; however, it was quickly realized that in order to make the entire surface feasible from all directions, the terrain gradients which exceed the performance capability of the aircraft would have to be decreased. This would lead to essentially raising the surface floor in the deep valley regions in order to make the valley shallower and feasible. For an exaggerated case, this would create a surface which hardly has any large gradients compared to the original surface and would prohibit the development of a path which closely followed the original valley.

Accounting for gradients which were too steep for the vehicle to climb was handled by the weighting of the path and the search algorithm. The cost to traverse the segment of infeasible gradient was assigned a cost of infinity, which would assured that the A* would not choose it as a low cost path to expand.

The temporal influence of the scaling based on terrain was not investigated and is avoided when using a consistent velocity UAS. The temporal influence is deemed negligible if a new solution is able to be calculated quickly.

2) How should the discretization technique scale (temporal & spatial) based on vehicle performance (caused by size/speed/maneuverability)?

The discretization of the map based upon aircraft performance is essential to achieving a feasible initial solution and establishing a feasible corridor. Without considering the aircraft performance in the discretization of the map, this algorithm would be useless.

The discretization of the map will scale based upon the maneuverability of the aircraft by scaling with the turn capability of the aircraft during a maximum climb maneuver. Using the aircrafts' minimum turn radius while performing a maximum climb maneuver will provide a conservative length to scale the discretization. The sides of the triangles which make up the discretized map should be at least twice the length of the minimum turn radius. A triangle consisting of sides at least twice the turn radius will allow the aircraft to enter the triangle at the mid-edge and conduct a turn equal to or less restrictive than the minimum turn radius and achieve a path which lies within the triangle and exits the triangle perpendicular to the neighboring edge. Increases in aircraft maneuverability allows for a finer discretization in regions where the complexity of the terrain deem a finer discretization beneficial.

The aircraft speed will influence the discretization scaling by directly influencing the maneuverability of the aircraft. Since an increase in aircraft speed is associated with an

increase in minimum turn radius and thus a larger minimum turn radius, increases in aircraft speed will increase the coarseness of the discretization.

The aircraft size may influence the discretization if certain clearance distances from obstacles are desired. The maneuverability requirement of sides greater than or equal to twice the turn radius assumes the vehicle as a point mass. To incorporate significant aircraft size into the discretization, one may require the sides of the triangles to be at least twice the minimum turn radius, plus an additional length, such as aircraft wingspan to ensure desired clearances.

The temporal effect on the discretization was not directly investigated. One obvious effect was that if the vehicle's velocity were to change over time for the same operating area, then it would be conservative to have the coarsest discretization allowable to ensure feasibility.

3) What Method Is the Best for Finding an Optimal Path Through the Equivalent 2D Discretized Surface?

The most vastly used method for finding the optimal path on a 2D surface is the A* algorithm, or some variation of it such as the Weighted A*or Anytime A* [4]. The heuristic of the A* algorithm allows the user to prioritize optimality or timeliness. Computing optimal paths to a complex problem where the agent must react quickly under a time constraint can sometimes be impossible, simply due to the number of states which must be considered to ensure the optimal path has been attained. Therefore, often times, the user must be satisfied with the best solution available, subject to the time available.

The best A* algorithm for finding a path across the 2D discretized surface within a specified amount of time is the Anytime A*, since it runs parallel instances of the A*

algorithm with different weights and selects the solution which is calculated the fastest. Once a solution is found, the Anytime A* continues to improve upon the solution as time permits. This assures a solution will be found and improved upon as long as time permits or the solution cannot be improved further. Parallel instances of the algorithm with different weights is necessary at the start because there are times when a heavily weighted A* solution can actually take longer than a lightly weighted solution [11]. This can be seen when the algorithm drives into a dead-end section of the map which does not contain the goal. This type of feature could be created within this methodology by the overlapping of multiple keep out zones.

The method herein uses a traditional A* algorithm with a manually variable heuristic for simplicity. A traditional A* was chosen to prioritize ensuring the feasibility of the methodology of this research before implementing a more complicated algorithm. An Anytime A* would likely have been the best algorithm to use to find the best path given a specified amount of time; however, it is likely that there will be a better way to store the data if an Anytime A* algorithm was to be used.

4) What Are the Tradeoffs of this Proposed Method as Compared to Alternative Methods in Use and Proposed in the Literature?

4.1) Alternative Methods

An alternative method to determine shortest paths is to utilize an optimal control solver. Such solvers are computationally expensive, lack the ability to robustly provide a solution, and in some cases provide solutions of unnecessary fidelity.

Another alternative method is finding paths through a set of nodes which represent 3D space. One method consists of multiple 2D map layers stacked on top of each other which allow for connection points between the layers, creating a 3D region out of multiple 2D planes, as described in [20].

Another method is one in which the 3D space is filled with nodes which allow for connections to their neighboring nodes. This method is not guaranteed to follow the contour of the surface and is more computationally expensive to compute.

The methods in the literature which are similar to the method proposed herein do not mention how they ensure the feasibility of the agent maneuvering across the surface, which is a key feature of the method proposed herein. Kallmann et al. [18] discuss clearance distances of paths between objects, but do not account for such in three-dimensions.

4.2) Tradeoffs

As with all things, there are tradeoffs. The main benefit of the proposed method is its ability to robustly calculate a sub-optimal terrain contour following solution in a timely manner.

A solution from an optimal control solver will typically have more fidelity to the solution than the method proposed herein. However, the robustness and speed at which the proposed method can calculated a solution is valuable and useful for a UAS which can follow waypoint navigation.

A technique which uses 2D layers with connection points between the layers to represent path options in a 3D space has a slight disadvantage when being used for an aerial vehicle because creating a limited the number of connections between 2D planes would

artificially limit the aircraft to transitioning only at the connection points. Additionally, the computational cost would be more than a single layer and provide no more improved of a solution when following the contour is desired.

A technique which uses a 3D spread of nodes throughout the space above the constraint surface has the potential to become computationally expensive due to the large number of nodes, and therefore paths. If this technique was chosen, it would be beneficial to keep the nodes placed close to the surface to achieve a path which follows the contour of the surface, similar to the method proposed herein.

The method proposed herein quickly and robustly calculates a near-optimal feasible solution which is typically more valued in everyday activities than a non-robust time-consuming methodology. However, the method lacks the ability continually improve upon the sub-optimal solution when there exists additional time to find a solution. The anytime algorithms do offer a tremendous advantage to robustness and optimality and should be utilized when possible.

Summary

The discretization technique is integral to reducing the time to solve the problem and the feasibility of the solution. The terrain characteristics, vehicle performance, and additional constraints influence the discretization and weighting of the map. Numerous algorithms exist for solving the shortest path solution, but variations of the weighted A* are most commonly used for its ability to quickly find a sub-optimal path and improve on the solution. The proposed methodology has some tradeoffs but the benefits outweigh the shortfalls for the problem proposed in Chapter I.

VI. Conclusions and Recommendations

Chapter Overview

This chapter presents the conclusions and significance of the methodology proposed. The objective of this research was to develop a methodology which provides a near-optimal path solution for a UAS in a highly constrained environment in near-real-time.

Conclusions of Research

The methodology herein proved the ability to yield near-optimal path solutions in the highly constrained environment quickly. The simulations showed the algorithm calculated a near-optimal path solution across terrain while avoiding regions which exceed the maximum climb angle or penetrate the keep out zones. The path is feasibly flyable due to the discretization method.

Significance of Research

This research is significant due to its ability to offer a method to determine a near-optimum path solution to a complex problem in near-real-time. Most traditional optimization solvers require a feasible initial guess. If the solver does not require an initial guess, providing a one will usually yield a solution faster. This method can be used to provide a near-optimum path solution which can be acted upon by a UAS in near-real-time. Alternatively, if the true optimum path is desired, this method can be used to provide an initial feasible guess to the optimization solver, thereby drastically reducing the time to calculate a solution.

This research contributes to the field by presenting observations about the discretization of the environment which will afford a feasible and timely solution. The unique feasible discretization and mapping method, the pre-calculation of data, and use of an efficient A* algorithm afford an efficient 3D pathfinding technique.

Recommendations for Action

In order to attain improved path solutions and execution of such paths by real flight vehicles across real terrain, two enhancements to this methodology are recommended; First, enabling the node expansions to expand to all four neighbors as opposed to only two neighbors at a time; second, utilizing a Latitude-Longitude-Altitude (LLA) wrapper to convert results to LLA and enable immediate usability of the method by flight vehicles.

Enabling the algorithm to expand to all four of its neighboring nodes upon each expansion, as opposed to only two nodes in the method herein, would provide more possible paths between nodes and improve solutions. While developing the methodology, it was assumed that a linear increase in the weighting of the nodes would allow for the calculation of a third path to be omitted, because it can be shown that for linear weights the path between nodes A-C is always cheaper than the path A-B-C. Imposing a maximum allowable angle constraint is a non-linear characteristic. Enabling expansion to all neighboring nodes would eliminate the requirement that the node weighting be linear in trend. The node expansion process is shown in Figure III.8 and the particular example given is quoted below:

“The second expansion (shown in red) does not evaluate the cost to go to nodes 4 or 5 from node 6. Since the cost to go to nodes 5 and 6 were already considered by iteration 1, their costs do not need to be calculated on expansion two since the cost to go to node 5

from node 4 on the first iteration will always be less than the cost to go to point 6 then point 5, due to geometry"

Allowing for calculation of the path to the third neighboring node can allow for a feasible path (A-B-C) to a node which may have been too steep to go to directly (A-C) if the gradient from A to C is too steep.

Additionally, it may be computationally more efficient to simply calculate and store the cost of the fourth path, which traces back to the parent node of the node being expanded, then it is to omit the calculation through the use of a check.

Second, utilizing LLA for terrain information and aircraft waypoints would be immensely useful. Converting LLA data of an operation area into the XYZ coordinate frame would be useful. This can be done by putting a wrapper in front of the code. The actual LLA information can be saved as additional rows in the pre-calculation matrix to allow for minimal time cost to convert the solution to LLA once a XYZ solution is attained. It is believed that calculations in a code which uses LLA would take longer than a code which receives LLA as an input, converts LLA to XYZ positions, does calculations, then converts the final solution to LLA. However, if one is to consider implementing this algorithm for fast vehicles over long distances, then using a spherical Earth model might be necessary.

Potential for Future Research

1) Non-Uniform Discretization Using the 2D Distance Between Nodes

Varying the discretization based upon the 2D distance between nodes for regions of increased complexity or gradient would reduce the computation time. The minimum

discretization size would be influenced by the minimum speed of the aircraft. The requirements for the connection of non-uniform triangles are outlined in the methodology.

2) Utilize an Anytime A* Algorithm

Utilizing an Anytime A* algorithm will allow for a solution within a specified amount of time. The Anytime A* guarantees a sub-optimal solution for any time (reasonable time) and continually improves the solution until no more time is available [4]. An anytime algorithm would increase the robustness of the proposed methodology by always providing a solution and providing the best solution depending on the time provided.

3) Non-Uniform Discretization Using the 3D Distance Between Nodes

Discretizing the surface based upon the 3D XYZ distance between nodes as opposed to the 2D XY distance would allow for a finer discretization in complex regions and improve the possible solutions. Such a discretization could likely enhance the ability of the algorithm to utilize switchback paths along the side of a steep terrain where the environment map permits such paths as the lowest cost.

4) Create a Matrix Which Allows for a Change in End Goal and Heuristic to be Accounted for Quickly

Since the heuristic used in this algorithm is the Euclidean distance to the end goal, then accounting for a change in the end goal would affect the heuristic of each segment. It is recommended that an additional column in the pre-calculation matrix be added to include the base cost to travel the path without the heuristic. Currently the algorithm does calculate both costs, but later adds the heuristic cost to the segment cost to establish the total cost

across the segment. Store the additional information would reduce the complexity of the calculation needed to account for a change in the end goal.

Summary

The methodology proposed herein was shown to yield timely near-optimal solutions usable for real-world applications in highly constrained environments. This methodology analyzed the requirements of the discretization to afford a timely and feasible solution. The recommended actions and suggested future work provide exciting potential to the methodology proposed in this research.

VII. Bibliography

[1] Joint Concept of Operations for Unmanned Aircraft Systems. US Department of Defense, Washington, D.C., 2011.

[2] James A Winnefeld and Frank Kendall. Unmanned Systems Integrated Roadmap FY2013-2038. Technical report, US Department of Defense, Washington, D.C., 2013.

[3] Air Force Research Laboratory. Air Force Research Laboratory Autonomy Science and Technology Strategy. Technical report, Air Force Research Laboratory, Wright-Patterson AFB, OH, 2013.

[4] D. Ferguson, M. Likhachev, and A. Stentz, "A Guide to Heuristic-based Path Planning," 2005. [Online]. Available: https://www.cs.cmu.edu/~maxim/files/hsplanguide_icaps05ws.pdf

[5] J. A. Rivera, "Design and Flight Test of a Path Planning Algorithm Utilizing Graph Theory for Real Time Applications," Master's Thesis, Air Force Institute of Technology, Wright-Patterson AFB, OH, 2019.

[6] T. M. Liebling, and Lionel Pournin, "Voronoi diagrams and delaunay triangulations:Ubiquitous siamese twins," Documenta Mathematica, pp. 419{432, 2012. [Online]. Available: https://www.math.unibielefeld.de/documenta/vol-ismp/60 liebling-thomas.pdf

[7] P. L. Chew, "Constrained delaunay triangulations," *Association of Computing Machinery*, 1987. [Online]. Available: https://www.cs.jhu.edu/_misha/Spring16/Chew87.pdf

[8] B. Ben-Moshe, O. Hall-Holt, M. J. Katz, and J. S. B. Mitchell, "Computing the visibility graph of points within a polygon," Symposium on Computational Geometry, pp. 27-35, 2004. [Online]. Available: http://www.ams.sunysb.edu/_jsbm/papers/p219-mitchell.pdf

[9] M. A. Vanputte, "Shortest Path Planning on Topographical Maps," Master's Thesis, University of Missouri-Columbia, 1997. [Online]. Available: https://pdfs.semanticscholar.org/8cdd/327e68fdfc102b2cb9763d95b467af477d54.pdf

[10] E. W. Dijkstra, "A note on two problems in connexion with graphs," *Numerische Mathematik*, 1959. [Online]. Available: http://www-m3.ma.tum.de/foswiki/pub/MN0506/WebHome/dijkstra.pdf

[11] C. Wilt and W. Ruml, "When does Weighted A * Fail ?," 2012. [Online]. Available: http://www.cs.unh.edu/~ruml/papers/wted-astar-socs-12.pdf

[12] N. Nilsson, The Quest for Artificial Intelligence. Cambridge, 2009. [Online]. Available: https://ai.stanford.edu/~nilsson/QAI/qai.pdf

[13] R. Dechter and J. Pearl, Rina. “Generalized best-first search strategies and the optimality of A*”. *Journal of the Association of Computing Machinery*, 32(3):505-536, 1985.

[14] K. Daniel, A. Nash, S. Koenig, and A. Felner, “Theta*: Any angle path planning on grids," *Journal of Artificial Intelligence Research*, 2010. [Online]. Available: https://arxiv.org/ftp/arxiv/papers/1401/1401.3843.pdf

[15] R. Geraerts, M. H. Overmars, “The corridor map method : a general framework for real-time high-quality,” pp. 107–119, 2007. http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.80.8355&rep=rep1&type=pdf

[16] R. Geraerts, Planning Short Paths with Clearance using Explicit Corridors, pages 1997-2004, 2010.

[17] M. Kallmann, 2010. “Shortest Paths with Arbitrary Clearance from Navigation Meshes”. Proceedings of the Eurographics SIGGRAPH Symposium on Computer Animation SCA, pp. 159—-168. Available: http://graphics.ucmerced.edu/papers/10-sca-tripath.pdf

[18] M. Kallmann, “Dynamic and robust local clearance triangulations,” *Association of Computing Machinery Transactions on Graphics*, vol. 33, pp. 269-271, 2014. [Online]. Available: http://graphics.ucmerced.edu/papers/14-tog-lct.pdf

[19] M. Kallmann, and M. Kapadia, 2014. “Navigation meshes and real-time dynamic planning for virtual worlds”. ACM SIGGRAPH 2014 Courses, pp. 1–81. Available: http://graphics.ucmerced.edu/papers/14-sig-navplan-s.pdf

[20] M. Kallmann. Flexible and Efficient Navigation Meshes for Virtual Worlds Flexible and Efficient Navigation Meshes. *In Simulating Heterogenous Crowds with Interactive Behaviors.* 2016. https://pdfs.semanticscholar.org/adba/12514a70ca8776ed77d114e682d0e8912215.pdf

[21] W. Van Toll, M. Kallmann, R. Geraerts, et al. “A Comparative Study of Navigation Meshes,” pp. 91–100. https://dl.acm.org/doi/10.1145/2994258.2994262

[22] R. Prazenica et al. “3-D Implicit Terrain Mapping and Path Planning for Autonomous MAV Flight in Urban Environments,” pp. 1–23, 2013.

[23] W. Zhan, W. Wang, N. Chen, and C. Wang, “Efficient UAV Path Planning with Multiconstraints in a 3D Large Battlefield Environment,” vol. 2014, 2014.

[24] P. Muñoz, M. D. R-Moreno, and B. Castaño, “3Dana : Path Planning on 3D surfaces,” *Int. Conf. Innovtive Tech. Appl. Artif. Intel.*, pp. 177–191, 2016. https://link.springer.com/chapter/10.1007/978-3-319-47175-4_12

[25] M. D. Zollars, “Simplex Control Methods for Robust Convergence of Small Unmanned Aircraft Flight Trajectories in the Constrained Urban Environment,” Ph. D. dissertation, Air Force Institute of Technology, Wright-Patterson AFB, OH, 2019.

.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 074-0188

PLEASE DO NOT RETURN YOUR FORM TO THE ABOVE ADDRESS.

1. REPORT DATE *(DD-MM-YYYY)*	**2. REPORT TYPE**	**3. DATES COVERED** *(From – To)*
26-03-2020	Master's Thesis	Sept 2018 – Mar 2020

TITLE AND SUBTITLE

Timely Near-Optimal Path Generation for an Unmanned Aerial System (UAS) in a Highly Constrained Environment

5a. CONTRACT NUMBER

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)

Matissek, Kyle J. Capt, USAF

5d. PROJECT NUMBER

5e. TASK NUMBER

5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAMES(S) AND ADDRESS(S)

Air Force Institute of Technology
Graduate School of Engineering and Management (AFIT/ENY)
2950 Hobson Way, Building 640
WPAFB OH 45433-8865

8. PERFORMING ORGANIZATION REPORT NUMBER

AFIT-ENY-MS-20-M-271

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

Air Force Research Laboratory, Aerospace Systems Directorate (AFRL/RQQA)
Attn: Dr. David Grymin
3110 Skyline Dr
WPAFB OH 45433-7542
Email: David.Grymin.1@us.af.mil

10. SPONSOR/MONITOR'S ACRONYM(S) AFRL/RQQA

11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT

DISTRIBUTION STATEMENT A:
APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

13. SUPPLEMENTARY NOTES

14. ABSTRACT

A current challenge in path planning is the ability to efficiently calculate a near-optimum path solution through a highly-constrained environment in near-real time. In addition, computing performance on a small unmanned aerial vehicle is typically limited due to size and weight restrictions. The proposed method determines a solution quickly by first mapping a highly constrained three-dimensional environment to a two-dimensional weighted node surface in which the weighting accounts for both the terrain gradient and the vehicle's performance. The 2-D surface is then discretized into triangles which are sized based upon the vehicle maneuverability and terrain gradient. The shortest feasible path between the nodes of the two-dimensional triangulated surface is determined using an A* algorithm. An optimal path is then chosen through the unconstrained corridor to yield a quick near-optimal path solution in three-dimensional space. This technique requires prior knowledge of the terrain map and vehicle performance. The cost to traverse each segment of the map is independent of the starting position on the map and can be pre-calculated once the goal position is known. The proposed method allows for a rapid path solution from any start position to a goal position while satisfying all constraints. It was shown that employing the methodology herein resulted in near-optimal solutions in less than a couple seconds for the scenarios tested. The future work section proposes methods for improving the algorithms efficiency even further.

15. SUBJECT TERMS

Unmanned Vehicles, SUAS, Path Planning, Terrain Following

16. SECURITY CLASSIFICATION OF:			**17. LIMITATION OF ABSTRACT**	**18. NUMBER OF PAGES**	**19a. NAME OF RESPONSIBLE PERSON**
a. REPORT	**b. ABSTRACT**	**c. THIS PAGE**			Dr. Richard G. Cobb, AFIT/ENY
U	**U**	**U**	**UU**	94	**19b. TELEPHONE NUMBER** *(Include area code)* (937) 255-6565 x4559 richard.cobb@afit.edu

Standard Form 298 (Rev. 8-98)
Prescribed by ANSI Std. Z39-18