Real-Time Obstacle and Collision Avoidance System for Fixed Wing Unmanned Aerial Systems

By

Julien F. Esposito

Submitted to the graduate degree program in Aerospace Engineering and the Graduate Faculty of the University of Kansas in partial fulfillment of the requirements for the degree of Ph.D. in Aerospace Engineering.

_____

Chairperson Dr. Shahriar Keshmiri

_____

Dr. Mark Ewing

_____

Dr. Ray Taghavi

_____

Dr. David Downing

_____

Dr. Chris Allen

Date Defended: June 6th 2013

The Dissertation Committee for Julien F. Esposito

certifies that this is the approved version of the following dissertation:

Real-Time Obstacle and Collision Avoidance System for Fixed Wing Unmanned Aerial Systems

_____

Chairperson Dr. Shahriar Keshmiri

_____

Dr. Mark Ewing

_____

Dr. Ray Taghavi

_____

Dr. David Downing

_____

Dr. Chris Allen

Date approved: June 11th 2013

Abstract

The motivation for the research presented in this dissertation is to provide a two-fold solution to the problem of non-cooperative reactive mid-air threat avoidance for fixed-wing unmanned aerial systems. The first phase is an offline UAS trajectory planning designed for an altitude-specific mission. The second phase leans on the results produced during the first phase to provide intelligent, real-time, reactive mid-air threat avoidance logic. That real-time operating logic provides a given fixed-wing UAS with local threat awareness so it can get a feel for the danger represented by a potential threat before using results produced during the first phase to require aircraft rerouting.

The first original contribution of this research is the Advanced Mapping and Waypoint Generator (AMWG), a piece of software which processes publicly available elevation data in order to only retain the information necessary for a given altitude-specific flight mission. The AMWG is what makes systematic offline trajectory possible.

The AMWG first creates altitude groups in order to discard elevations points which are not relevant to a specific mission because of the altitude flown at. Those groups referred to as altitude layers can in turn be reused if the original layer becomes unsafe for the altitude range in use, and the other layers are used for altitude re-scheduling in order to update the current altitude layer to a safer layer. Each layer is bounded by a lower and higher altitude, within which terrain contours are considered constant according to a conservative approach involving the principle of natural erosion.

The AMWG then proceeds to obstacle contours extraction using threshold and edge detection vision algorithms. A simplification of those obstacle contours and their

corresponding free space zones counterparts is performed using a fixed –tolerance Douglas-Peucker algorithm. This simplification allows free space zones to be described by vectors instead of point clouds, which enables UAS point location.

The resulting geometry is then processed through a vertical trapezoidal decomposition where for each vertex defining a contour a vertical line is drawn, and the results of this decomposition is a set of trapezoidal cells. The cells corresponding to obstacle contours are then removed from the original trapezoidal decomposition in order to solely retain the obstacle-free trapezoidal cells. After decomposition, cells sharing part of a common edge are considered from a graph theory perspective so it becomes possible to list all acyclic paths between two cells by applying a depth first search (DFS) algorithm.

The final product of the AWMG is a network of connected free space trapezoidal cells with embedded connectivity information referred to as the Synthetic Terrain Avoidance (STA network). The walls of the trapezoidal cells are then extruded as the AWMG essentially approximates a three-dimensional world by considering it as a stratification of two-dimensional layers, but the real-time phase needs 3D support. Using the graph conceptual view and the depth first search algorithm, all the connected cell sequences joining the departure to the arrival cell can be listed, a capability which is used during aircraft rerouting. By connecting two adjacent cells' centroids to their common midpoint located on the shared edge, the resulting flying legs remain within the two cells. The next step for paths between two cells is to be converted into flyable paths, and the conversion uses main and fallback methods to achieve that. The preferred method is the closed-form Dubins paths method involving the design of sequences of arc circle-straight line-arc circle (CLC) in order to account for the minimum radius turn constrain of the UAS. An additional

geometric transformation is developed and applied to the initial waypoints used in the Dubins method so the flying leg directions are respected which is not possible by using the Dubins method alone. When consecutive waypoints are too close from one another, a condition called the Dubins condition cannot be respected, and the UAS trajectory design switches to the numerical integration of a system of ordinary differential equations accounting for the minimum turning constraint. Using the Dubins method and the ODE method makes it possible for the AWMG to design flyable offline trajectories accounting for the lateral dynamic of the fixed-wing UAS.

The second original contribution of this research is the development and demonstration of the Double Dispersion reduction RRT (DDRRT), an algorithm which employs two new developed logic schemes respectively referred to as Punctual Dispersion Reduction (PDR), and Spatial Dispersion Reduction exploration (SDR). The DDRRT is employed during the real-time in-flight phase where it initially assumes a perfect terrain and no unpredictable threat, consequently following a 100% adaptive goal biasing toward the next waypoint in its list. When a threat such as an unpredicted obstacle is detected, the (PDR) acknowledges the fact that the DDRRT tree branches have met an obstacle and the its goal-biasing toward the next waypoint is decreased. If the PDR keeps decreasing, the DDRRT develops awareness of its surrounding obstacles by relaxing its PDR and switching to SDR which has the effect of increasing the dispersion of its branches, but keeping their extension bounded by the cell containing the last good position of the UAS, C_{safe}. If a number of branches reach a limit proportional to the C_{safe} and its relative area, then the STA network is queried for alternative rerouting. The two phases provide real-time reactive mid – air threat avoidance scenarios with the ability for a UAS to develop local and realistic threat

awareness before considering intelligent rerouting. Either the local exploration of the DDRRT is successful before reaching a maximum number of points, or the STA Network is required to find another route.

Acknowledgements

Perhaps the most enjoyable part of a PhD dissertation writing is the acknowledgement part!

I want to express my gratitude, in no particular order, to the following people:

Dr. Keshmiri for his constant guidance and support during my research at KU. From him I have learned the importance of developing depth of knowledge before breadth. My father, Edoardo Esposito, whose unconditional love and financial support helped me greatly during difficult times. Scott Rowe for his friendship and sharing his inappropriate but yet much appreciated "New England" sense of humor. George Pubanz (TriQuint) for helping me to write this document despite his busy schedule. Catherine Knight for welcoming me again to her home and letting me finish my document write-up.

My former office mates Himanshu Dande, Vivek Ram, and Sarah Kulhanek for bringing hours and hours of much appreciated "lols" in our office despite our busy schedules and various commitments toward our students. Also special thanks to Alec Bowman and Vivek Ram for helping me to rehearse for my oral defense.

Table of Contents

Abstract ____ iii

Acknowledgements ____ vii

Table of Contents ____ viii

List of Figures ____ xi

List of Tables ____ xv

List of Algorithms ____ xvi

I - Introduction ____ 17

II - Literature Review Covering Synthetic and Real Time Path Planning for UASs ____ 23

II - 1 - Optimization Formulations ____ 26

II - 2 - Genetic Algorithms ____ 29

II - 3 - Potential Functions ____ 32

II - 3 - 1 - Potential Functions - Analytical Expression ____ 32

II - 3 - 2 - Potential Functions - 3D Grid Formulation ____ 40

II - 4 - Geometric Approaches ____ 48

II - 5 - Graph Constructions ____ 53

II - 5 - 1 - Common Concepts to 2.5 D methods and Roadmaps ____ 53

II - 5 - 2 - 2.5 D Space Decomposition Methods ____ 54

The Visibility Graph Decomposition ____ 55

The Trapezoidal Decomposition ____ 56

II - 5 - 3 - Probabilistic Roadmaps ____ 57

II - 6 - Rapidly Exploring Random Trees ____ 64

III - Analysis of Existing Technologies and Identification of a Direction of Research _____ 68

III – 1 – RRT Real-Time Space Exploration and Path Finding _____ 70

III – 2 - Assumptions Used When Assessing an Aircraft Performances _____ 72

III – 2 – 1 – Assumptions Employed for Lateral Maneuver Decisions _____ 72

III – 2 – 2 – Decoupled Longitudinal Maneuvers – The Climb_____ 74

III - 2 - Aiding the Real-Time Threat Avoidance by Preparing a Synthetic Terrain Avoidance Map _____ 76

IV – Preliminary Results_____ 79

IV - 1 - Offline USGS Map Transformation to Aid In-Flight Threat Avoidance_____ 82

IV - 1 - 1 - The Trapezoidal Decomposition Algorithm Employed by the AMWG _____ 88

IV - 2 - Dubins Curves for Lateral Maneuvers Following Waypoints _____ 102

IV - 2 - 2 - Dubins Curves' Equations _____104

IV - 2 - 3 - Dubins Curves With Added Logic for Perfect Flying Leg Tracking _____108

IV - 3 - The Double Dispersion-reduced RRT _____ 116

V – Results _____ 120

V – 1 – A brief overview of the Existing RRT Variations and Why They Are Not Well Suited for Real-Time Realistic Terrain Avoidance _____ 120

V – 1 – 1 – RRT Variation One – The Goal – Biased RRT _____126

V – 1 – 2 – The Multiple RRT _____129

V – 1 – 3 – Conclusion Regarding the Existing RRT Algorithms and Why another Type of RRT Must be Developed for Real-Time Reactive Threat Avoidance_____130

V – 2 – The DDRRT_____ 133

V – 2 – 1 – The DDRRT Spatial Dispersion Reduction _____133

V – 2 – 2 – The DDRRT Punctual Dispersion Reduction 140

V – 2 – 3 – The SDR and the PDR used Altogether in the DDRRT 148

V – 2 – 4 – A Practical Example of Real-Time Threat Avoidance Using the DDRRT 153

In Figure 48, the DDRRT encounters an obstacle, as previously explained. This is shown by a red segment deployed to mark the corresponding collision. The next figure shows the progression of the PDR of the DDRRT. 154

Figure 51 shows the failure for the joint efforts of PDR and SDR to find an alternative route between cell 3 and cell 1 while remaining in one of the two cells. As a result, the DDRRT requests the STA network to provide it with an alternate route, which is shown next. 156

V – 2 – 5 – The DDRRT Algorithmic Strategy When the Current Altitude Layer Cannot Provide Planar Rerouting 160

V – 3 – 6 – Adding Logic to Threat Detection In Order to Create an STA Network When No Elevation Data is Available 166

VI – Closure 168

VI – 1 – Summary 168

VI – 2 – Conclusion 171

VI – 3 – Recommendations and Future Work 173

Appendix A – The DDRRT SDR Algorithm 176

References 177

List of Figures

Figure 1 - Hierarchical View of Available Technique for Path Planning of Vehicles and Robots 25

Figure 2 - Effects of Varying W_g and W_o on the Overall Guidance Driven by the Cost Function 36

Figure 3 - Local Minimum Introduced in Cost Function 38

Figure 4 - Effect of Creating Artificial Local Minima Through a "Wall" of Obstacles 39

Figure 5 - Numerical Laplacian Produced After One Averaging Process for the "Passino" Problem 46

Figure 6 - Attractive and Repulsive Forces Shown as Inward Outward Pointing Vectors 47

Figure 7 - Generalized Diagram Showing the Two Point Mass Objects and the Closest Point of Approach 49

Figure 8 - Miss distance vector with UAV B cooperative and UAV A uncooperative 52

Figure 9 - Planar Trapezoidal Cell Decomposition 57

Figure 10 - Schematic for the Process of Connecting K = 5 Nearest Neighbors In the Query Phase of the PRM Construction 59

Figure 11 - Duality Between Delaunay Triangulation and Voronoi Diagram 60

Figure 12 - The Dangers of Trajectory Smoothing 62

Figure 13 - RRT Spread for $U = \|x_{rand} - x_{near}\|$ 66

Figure 14 - Derivation of the Turning Radius Formula for a Coordinated Turn 73

Figure 15 - Diagram of a Climb Maneuver 74

Figure 16 - A 10x10 m DEM Map of the Anderson Canyon, Arizona Made of 1143x1388 Data Points 77

Figure 17 - Two Altitudes Used as Criteria for Elevation Point Selection from the USGS 10mx10m Map of the Grand Canyon 80

Figure 18 - From USGS File to Working Synthetic Terrain Avoidance Map 85

Figure 19 - Trapezoidal Cell Adjacency Computation Algorithm 89

Figure 20 - Trapezoidal Decomposition Applied to a 2D Map 90

Figure 21 - Undirected Connectivity Graph Generated from Trapezoidal Decomposition Connectivity Property 91

Figure 22 - Four Different Paths to Connect Two Points 94

Figure 23 - Computation of the Path Between Two Cells 95

Figure 24 - Four Paths in White and Their Simplification in Black 97

Figure 25 - Path Ranking for Different Criteria 100

Figure 26 - Four Different Dubins Curves 103

Figure 27 - Example of a Dubins Curve Joining Two Points Q_s and Q_g By Following a Right Turn, Straight Line, Right Turn 106

Figure 28 - Transformation of a Triplet of Waypoints So a Dubins Path Respects the Flying Leg Directions 107

Figure 29 - Static and Dynamic (Dubins) Trajectories. Uncorrected Dubins (Yellow), Corrected Dubins (Pink) 109

Figure 30 - Trajectory Following at Constant Speed for Different Maximum Angles 111

Figure 31 - Trajectory [2 3 9 1 8] at Different Airspeeds, Same Maximum Bank Angle 113

Figure 32 - Heading Rate Discontinuities Introduced by Heading Changes 114

Figure 33 - Example Showing Point Sampling in Natural Coordinates and Mapping to the Configuration Space 117

Figure 34 - Comparison between Simulation-Time-Bounded RRT Expansion and Free RRT Expansion 123

Figure 35 - Goal - Biased RRT - Ten Biases 127

Figure 36 - A Common Trait to All Previously Developed RRT Algorithms - The Tendency to Alternatively Grow on Two Opposite Sides of an Obstacle 131

Figure 37 - The First Dispersion Reduction Level of the DDRRT Showing Green Volumes as Primary Areas to Target the DDRRT Real-Time Exploration 134

Figure 38 - The DDRRT Using the Spatial Dispersion Reduction Only and No Alternative Routes 136

Figure 39 - DDRRT Using Spatial Dispersion Reduction Only for Alternative Route Decisions 138

Figure 40 - The Waypoints Designed for the Punctual Dispersion Reduction of the DDRRT .. 140

Figure 41 - DDRRT Limited to PDR Only and Following a Succession of Waypoints With Zero Dispersion 142

Figure 42 - DDRRT in PDR Mode Only Skipping Only WP3 among the Corrupted Sequence 144

Figure 43 - PDR-Limited DDRRT Showing Its Tree Extension after Discarding the Original Waypoint Sequence In Order to Converge Directly to the Goal 145

Figure 44 - Results of 100 Runs of PDR-Limited DDRRT 147

Figure 45 - PDR-Limited DDRRT Using the STA Network for Rerouting - Showing Two Alternative Routes 149

Figure 46 - DDRRT SDR Proportional Area Coverage for Surrounding Awareness 152

Figure 47 - The UAS Follows Two Flying Legs Designed During Offline Trajectory Planning 153

Figure 48 - The UAS Encounters a Red Obstacle Between Cell 3 and Cell 1. 154

Figure 49 - The DDRRT In Decreasing PDR Mode..... 154

Figure 50 - The PDR Goal Biasing Has Decreased to Fall Below 50%. 155

Figure 51 - The DDRRT Is In SDR Mode Only, Developing 371 Branches Out of the 2000 Possible. 156

Figure 52 - The DDRRT Has Spent 371 Branches Trying to Breach Safely Into Cell 1, ... 156

Figure 53 - Two STA Networks Used for Inter-Layer Path Queries..... 161

Figure 54 - Polygon Clipping Test Performed Between Two Cells of Two different STA Networks 162

Figure 55 - Longitudinal Maneuver Cone 164

Figure 56 - A Detection Sheet 167

List of Tables

Table 1 - Positive and Negative Points for the Six Methods Reviewed for Collision and Obstacle Avoidance 68

Table 2 - Details of Point Sequences for the Different Dubins Configurations 105

List of Algorithms

Algorithm 1 - Guidance Algorithm Used in the Potential-Field-Based Example 35

Algorithm 2 - Probabilistic Roadmap Algorithm 58

Algorithm 3 - RRT Algorithm 64

Algorithm 4 – The Path Simplification Algorithm 96

Algorithm 5 – Algorithm to Generate a Feasible Lateral Maneuver Between Two State Vectors (x,y,φ)..... 158

I - Introduction

Many Aerospace departments from various universities across the U.S. are conducting flight tests involving Unmanned Aerials System (UAS) flights. As a result of the potential for UAS to perform missions where human presence is unwanted, research in autopilots has reached a level of maturity such that those same universities can afford autopilots without involving a defense-sized budget. Lifting surface-wise, UAS typically fall into two categories: rotary wing UAS such as helicopters and quad-rotors, and fixed-wing UAS such as acrobatic airplanes.

The history of UAS research at the University of Kansas began in 1999 but it was in 2007 that a Cloudcap Piccolo II autopilot (Ref. [1]) was mounted on a 33% scale Yak-54 UAS (Ref. [2]), a fixed - wing acrobatic airplane. The autopilot's objective was to smoothly interpolate position-velocity waypoints defining a reference trajectory so that the aircraft could follow that path as closely as possible. Flight tests using Cloudcap's product were however declared unsafe when many communication-losses between the ground station and the aircraft were observed, one of which resulted in the aircraft crashing. A study conducted in Ref. [3] suggested that those communication losses were caused by the high update rate of the autopilot used, creating a bottleneck between the theoretical data link transfer rate and the actual one. In 2008 the weControl wePilot 2000 (Ref. [4]) replaced the Piccolo II autopilot, and more than 65 consecutive autonomous flights have since been performed successfully. Such success provided the flight test research team with more insight on what made the wePilot 2000 such a valuable tool: beside a great appropriate hardware/sensor integration and H-infinity based controller, a 20 Hz update rate of the guidance and control logic was used.

As sophisticated as they may be, autopilots only address one problem: the guidance and control (GNC) problem. Any decent autopilot allows flight test engineers to upload a list of waypoints, which can consequently be used for guidance while the controller generates inputs for the aircraft's deflection surfaces and the throttle setting. However, for an increased level of autonomy such that the flown aircraft can avoid potential obstacles or collisions, autopilots cannot be used alone. Some companies in charge of designing and manufacturing defense-grade autopilots do however partially address this avoidance problem. Such is the case of the Athena autopilot line by Rockwell-Collins: the Athena 211e can be integrated with TCAS II type transceivers, allowing flight GNC and collaborative collision avoidance capability. But such collision avoidance capability is simply the result of a good integration of TCAS II transceivers with high-performance autopilots. Asides from military aircraft, there are currently no commercial solutions available for general aviation (GA) combining a GNC module with a non-collaborative collision/obstacle avoidance logic. Such unavailability is one of the many reasons for the research presented here: *finding a solution to the problem of obstacle and collision avoidance for fixed-wing UAS/GA aircraft when potential hazards are not signaled ahead of time to the flying aircraft*.

The problem of designing a reactive obstacle and collision avoidance system is much more difficult than using collaborative collision avoidance or pre-mapped terrains to avoid threats because it is a double problem:

- There is the need to find an adequate sensor combination to accurately and rapidly detect obstacles and encounters while meeting budget and payload requirements.

- The algorithm(s) employed to avoid the obstacle(s) and/or collision(s) must account for various scenarios and be computationally fast enough in order to be used in real-time, while accounting for the aircraft's dynamics and reachability of a set of waypoints.

Due to the complexity of non-cooperative collision and obstacle avoidance technology, initial efforts have focused upon cooperative techniques.

The first research efforts in the field of cooperative collision avoidance were initiated in 1955, when a crash between a DC-7 and a Lockheed Super Constellation occurred over the US Grand Canyon. This incident is what led Dr. John S. Morrell from the former Bendix Corporation to approach the problem of collision avoidance under its time component rather than under its distance aspect. It is that same concept of time of closest approach (τ), which is at the core of the current cooperative collision avoidance technology. Unfortunately, the research involved in what would later become the Traffic Collision Avoidance System (TCAS) technology was not developing fast enough to prevent the tragically frequent accidents which kept occurring until the early 1990s. Major examples of aircraft collision include the 1978 collision between a Boeing 727 and a Cessna 172 over San Diego, the 1986 mid-air collision between a DC-9 and a Piper Archer over Cerritos, California, or the 1996 collision between a Boeing 747 and an Ilyushin 76 near New Delhi, India, which cost the life of 349 passengers. The consequence of these tragedies on research management is that the TCAS technology changed from being an FAA program to a U.S. Congress-required feature for all carrier aircraft operating within U.S. airspace. More recently, the collision between the RQ-7 Shadow US army drone and the USAF C-130 cargo airplane in August 2011 resulted in human losses and is an example of the need to further collision detection and avoidance research for UAS.

The Federal Aviation Administration (FAA) in the U.S. and the Joint Aviation Authorities (JAA) in Europe are two important aeronautical organizations in charge of establishing safety standards for aircraft as well as managing research programs. Safety standards managed by the FAA for UAS are subject to constant evolution and a designated group, the Radio Technical Commission for Aeronautics (RTCA), is in charge of updating those standards. The Committee in charge of updating the Minimum Aviation Performance Standards for Unmanned Aircraft System (MASPS) publishes each year a document reflecting those changes. A sub category of standards is defined for the class of micro UAS (of which the 33% scale Yak-54 belongs to): the "WK5673 - Standard Guide for Mini-UAV Airworthiness", which summarizes the different criteria any Micro UAS (MUAS) must meet in order to be declared airworthy. In parallel to the safety standards edited by RTCA, FAA has established compliance rules for any aircraft circulating within the National Air Space, stating that "only those UAS that have the capability of pilot intervention, or pilot-on-the-loop, shall be allowed in the National Air Space outside of Restricted, Prohibited, or Warning areas" (Ref. [5]). In addition to human supervision, UAS must comply with the right of way rules, such as those applied for general aviation and presented in Ref. [6].

Clearly, establishing standards for UAS airworthiness is a daunting task involving the work of many specialists. That is the reason for which the RTCA alone accounts for 400 organizations spread all across the world. In an effort to simplify UAS classification based on FAA standards of airworthiness, a first step to evaluate UAS is to consider them under their autonomy component only. Doing so gives an estimate on how good a system is at performing decisions for given scenarios, including collision and obstacle avoidance.

An interesting resource treating of autonomy levels for UAS is Ref. [7], where the author states that in the context of UAS, intelligence and autonomy are often treated as synonyms while they should not be. According to Bruce T. Clough, Technical Area Leader at the Air Force Research Laboratory, intelligence is the capacity to acquire knowledge in order to "do something", whereas autonomy is the ability to perform decisions based on preprogrammed knowledge. Perhaps the most interesting aspect of Ref. [7] may be the point made by Clough when referring to the previous work of the Los Alamos National Laboratory (LANL) and as well as the Cambridge Draper Laboratory (Draper): regardless of the metrics used by each laboratory, a domestic toaster and a UAS can be classified with the same autonomy levels. The reason for the existence of Ref. [7] is to show that metrics used to create autonomy levels must be selected carefully, as the Mobility, Acquisition, and Protection (MAP) metrics from LANL offer poor resolution to evaluate different autonomy levels, and those selected by the Draper Laboratory are in some cases irrelevant, task planning not being a prerequisite for autonomy.

Such a lack of pertinence in previous attempts at classifying UAS autonomy levels are the reason for the development of the Autonomous Control Level (ACL) chart, which covers 10 levels of autonomy under the metrics of:

- Perception/Situational Awareness
- Analysis/Decision Making
- Communication/Cooperation

This classification has also been adopted by the Department of Defense in Ref. [8].

The ultimate goal of the present research is to design a program which operates as a realistic real-time visual simulation framework for fixed-wing UAS flight scenarios involving both non-

cooperative and cooperative obstacle and collision avoidance. In order to achieve collision avoidance, obstacle avoidance is targeted first as it is simpler to implement.

Obstacle avoidance can only be achieved after previously gained knowledge of an area. Indeed, paths satisfying both goal reach (rendezvous) and obstacle avoidance require previous knowledge of the flown space, even locally. In an effort to determine the available technologies fitting best UAS non-collaborative reckoning and path planning, six techniques are reevaluated according to two metrics:

- execution time
- relevance to the problem of obstacle avoidance for UAS

The execution time is a direct indicator of the reaction time of a UAS for a given scenario. The relevance metric is an arbitrary assessment of the involved assumptions for a given technique, and how sound those assumptions are in a realistic case of UAS obstacle/collision avoidance.

After concluding on the most suitable collision/obstacle avoidance methods for fixed-wing UAS, the problems of matching planned paths with actual aircraft maneuverability are identified and listed as areas of research.

The original proposed contribution to collision/obstacle avoidance for fixed-wing micro UAS will be a software suite allowing simulations of 33% scale Yak-54 flight tests where obstacles and moving intruders are to be avoided. The resulting software simulator will use synthetic cartography information as well as simulated sensed obstacles as a framework to validate in-development collision/obstacle avoidance strategies.

II - Literature Review Covering Synthetic and Real Time Path Planning for UASs

With the exponential growth of UAS flights across the world, non-cooperative collision avoidance is a very active research area. This is in contrast with the somewhat slower-paced collaborative collision avoidance. The fact that collaborative collision avoidance devices such as TCAS-II and ADS-B equip general aviation and military aircraft certainly contributes to its rigidity: dealing with the lives of hundreds of passengers cannot be treated lightly, and any new progress must be evaluated thoroughly in order to make sure it does not compromise existing safety. Collaborative collision avoidance is mostly used for Manned Aerial Systems (MAS). On the other hand, non cooperative collision avoidance is inherently a much more challenging research topic. For UAS not equipped with cooperative threat avoidance systems, the level of autonomy must be increased to promote individual-avoidance, so those same UAS can rely solely on themselves.

UAS safety in the event of potential non-cooperative threat requires answering two questions: what is the best sensor combination for a given UAS, and what is the best way to use the sensed information? While the first question is not the object of this document, the current section aims at summarizing the options available for non-collaborative collision avoidance researchers in the Aerospace field.

Collision and obstacle avoidance are two desirable features in many scientific projects involving UAS. The two are usually distinguished in scientific literature: obstacle avoidance aims at avoiding static obstacles such as terrain elevations, whereas collision avoidance aims at escaping a moving potential threat such as another aircraft, moving object, or birds. In biology, an

example of obstacle avoidance is finding a path between molecules so another molecule can be inserted without entering in contact with the others. In robotics, obstacle avoidance can be used to solve a path planning problem so a robotic arm can manipulate an object without colliding with the walls of an assembly line. And in Aerospace in particular, obstacle and collision avoidance can serve two purposes:

- **Offline path planning:** given a coarse map, find a collision-free route from a starting point to an end point or goal, while accounting for some constraints inherent to the vehicle and the terrain geometry
- **Real time reactive path planning:** while flying a previously defined path, perform local route reactive re-planning, so if unpredicted obstacles or collisions threat need to be avoided, a conflict resolution maneuver can be issued in a timely manner.

Many techniques are available to researchers trying to implements the two above features. An attempt at categorizing these techniques results in the following list:

- Optimization Formulations
- Genetic Algorithms
- Potential Functions
- Geometric Approaches
- Graph Constructions
- Rapidly-exploring Random Trees (RRT)s

These techniques are organized in a more visual fashion in the next picture.

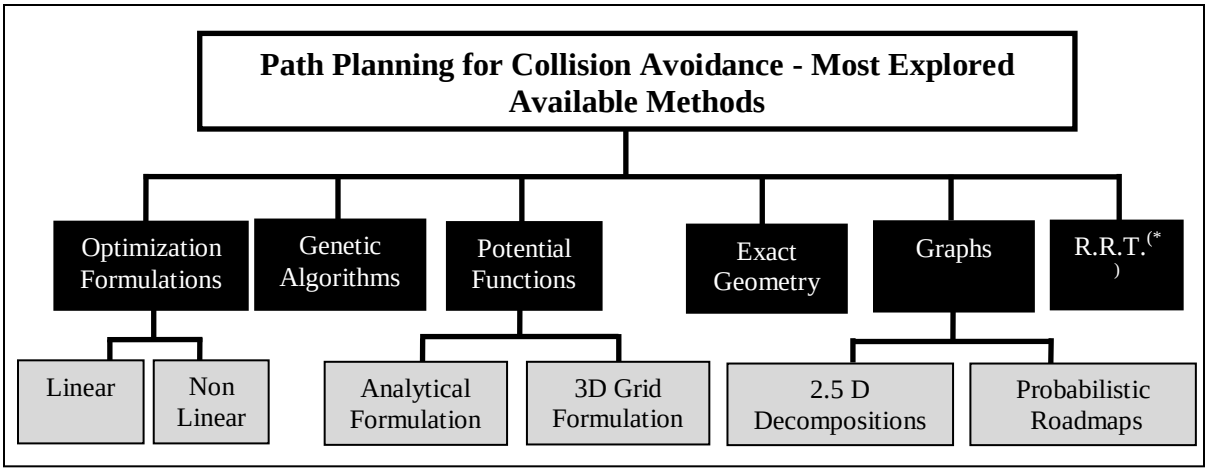


Figure 1 - Hierarchical View of Available Technique for Path Planning of Vehicles and Robots

II - 1 - Optimization Formulations

Optimization is the process of finding the best solution to a problem by formulating that same problem in terms of constraints and a cost function. Optimization can be used as a technique to solve a collision avoidance problem. The first phase of solving the problem is to formulate it. For an aircraft, such formulation must express the goal of finding admissible inputs (control surfaces and throttle settings for a realistic problem) such that the aircraft reaches a goal, while accounting for kinematic, inertial, aerodynamic, and obstacles constraints. Intuitively it appears that the more the constraints, the greater the problem complexity. Depending whether those constraints are linear or nonlinear, the optimization is said to be formulated as a Linear Programming (LP) problem or Nonlinear Programming (NLP) problem. Often time, LP approach involves the use of additional slack variables that are of integer type, and the LP problem becomes a Mixed Integer Linear Programming (MILP) problem Ref. [9, 10, 11].

In Ref. [9], the computation time is discussed for the problem of fixed point-mass UAS avoiding obstacles by comparing the NLP and MILP approaches. Since NLP and MILP approaches are fundamentally different in the way constrains are formulated, authors of Ref. [9] convert linear constraints to non linear constraints by taking the natural logarithm of the MILP kinematic constraints. The conversion of linear constraints to nonlinear constrains enables the comparison of the time taken to solve the same collision avoidance problem with respectively a linear and nonlinear solver. In the MILP formulation, collision avoidance scenarios do not seem to refer explicitly to the geometry involved in obstacles (the discrete obstacle volumes approximation is not known), but since obstacle constraints are formed as parallelepipedal protection zones, collision avoidance is handled by checking Euclidian norms between a vehicle's position with

respected to the different protection zones, and not by checking for contact between the point-mass UAS with the obstacles. The article shows that MILP computation time is always smaller than NLP time, while both optimization methods serve the same goal of reaching a position in the minimum amount of time while avoiding static obstacles. It is reported that for the same threat avoidance problem, MILP and NLP minimum solution computation times are 1.091 s and 1.422, respectively. The MILP and NLP maximum solution times are 18.20 s and 506.18 s. In others words, the NLP formulation yields computation times ranging from 23.3 to 2781 percent greater than the linear formulation. Reference [12] gives an example where a point-mass vehicle evolving in a 2D environment moves through a field of fixed obstacles, and where its dynamics are updated with a time step of 2 seconds. The UAS has limits on maximum speed, turn rate, and control forces. Although the assumption of a 2D UAS is justified by the fact that air space is structured into layers, the point mass simplification per se is the result of using a linear problem formulation. This is a point brought by the article itself, noting that banked turns are by nature nonlinear constraints. The result of solving this collision avoidance problem is a UAS that can avoid all obstacles, but the computation time to converge to a solution is of 49 seconds. The time to solve the simple problem of Ref. [12] and the previous comparison of MILP and NLP running times, gives an indication of what the computation time could be for a nonrealistic problem formulation. Also, the MILP approach of Ref. [12] clearly shows that the collision avoidance problem must be formulated so the LP solver CPLEX can treat it. This is done by formulating the problem as a set of AMPL commands. AMPL is a mathematical language using a set of common instructions to allow a user to use different mathematic solvers using different instructions in a same uniform framework. Based on Refs. [9, 10, 11] the perspectives of using

optimization formulations to develop threat avoidance for UAS are those of using complex tools in a complex fashion to solve simplified problems in suboptimal time.

II - 2 - Genetic Algorithms

Genetic Algorithms (GA) are a set of techniques trying to mimic nature and solve problems by applying the process of natural evolution/selection. Those techniques belong to the superset of Evolutionary Algorithms, an area of research in Artificial Intelligence. The main idea of genetic algorithms is to encode candidate solutions to a problem and then apply a variety of algorithms related to that process of natural evolution. Rather than trying to produce a comprehensive coverage of such techniques, research presented in Ref. [[13] is used to illustrate the use of GA as tool allowing design and optimization of collision-free paths for UAS by imitation of natural selection.

In Ref. [13], authors apply GA to a population of N randomly generated paths, each joining n waypoints. Initially, all paths are connecting a start position to a goal position. These paths are then compared to each other in what authors call tournaments, where based on a fitness function, only part of the initial pool is retained as parents to generate new generations of paths. The new generation is the result of crossovers between the two previously selected parents, and the crossover between the two parents generates new paths.

An interesting aspect of the natural selection proposed in Ref. [13] is the fitness function used. This function produces a scalar number reflecting the quality of generated paths according to four parameters. Hence the term of fitness employed in Ref. [13] is actually a performance index summarizing and ranking the paths generated over a given generation. The ranking is established by multiplying a linear combination of the weighted path lengths *s*, path segments vertical climb rates V_c, and path segments descent rates V_d, with the number of path segments that intersect the

terrain i. This is equivalent to account for the UAS' dynamics by approximating them using maneuver indicators. The full expression of this fitness function is

$f = (s + k_c V_c - k_d V_d)(1 + p.i)$ where

- k_c is the climb factor
- k_d the descent factor
- p *t*he penalty for a terr*a*in path intersection with the terrain.

As a cost is attributed to each parameter, this fitness function is by definition a cost function. Interestingly, paths generated by the genetic algorithms used bear penalties proportional to the number of times they cross obstacles, but those same paths are allowed to collide with obstacles, as opposed to other path planning approaches. In addition, it seems that during the generation process, path flyability is not considered under its lateral component, but treated separately in the design of a reactive planner, which means that genetic algorithms treat offline path planning under the aircraft's vertical component only, whereas lateral maneuvers are treated during the real-time (reactive) collision avoidance. Authors conclude that GA algorithms provide good results for offline planning, but state that the results (feasible paths) depend greatly on the fitness function, making genetic algorithms a heuristic method in essence. The article also mentions the random behavior resulting from the initial selection of the waypoints constituting each initial path. This randomness indicates that the number of collision tests involved is random as well, a trait shared by the Rapidly-exploring Random Trees. The authors run simulations on a fictitious city with many buildings, so geometric consideration are expected, since collision with obstacles are clearly tested during the GA-driven design of flyable paths. No clear mention is made to correlate geometry complexity with running time, so even though the fitness function accounts

for collision detected with the terrain, evaluating a given set of parameters for the fitness function without accounting for collision detection tests involved only gives a partial answer about the method's performances. Hence, with experiments presented in Ref. [13], GA seem to provide an answer to the problem of synthetic path planning for UAS, but are highly heuristic in nature, so the aptitude of designing a "good" fitness function depends upon the design of a ‘good’ fitness function. Such design problem is inherent to the heuristics used: GA work by performing path selections based on a function which rewards paths of shorter length while highly penalizing collisions. This function is thus not a "good" candidate when paths are designed to avoid large obstacles. In fact, for large obstacles, the additional path length required for avoidance may make it more advantageous to fly through the obstructions despite the penalty than actually avoiding them. This consequently implies that designing fitness functions is almost a "per flight mission topology" task, meaning that virtually any new flying area should be assigned an individual fitness function. In addition to the choice of a fitness function, authors mention that different fitness function tunings result in their GA algorithm finding an obstacle-free route in 4 seconds on average. This is an important number, but perhaps it is more interesting to relate it with the complexity of the city the micro UAS are navigating.

II - 3 - Potential Functions

Potential Functions have their origin in Potential Field Theory which finds its roots in Ref. [14]: "Real-time obstacle avoidance for manipulators and fast mobile robots", the first article on the topic published by Oussama Khatib, professor of Computer Science at Stanford University. It was in 1986 that Khatib developed the idea of solving an obstacle avoidance problem by using two types of artificial functions:

- a function $F_O(\bar{x})$ which models the fact that obstacles should repulse a vehicle having knowledge of them (the so-called repulsive forces)
- a function $F_g(\bar{x})$ which models the attraction of a vehicle toward a goal position (the so-called attractive force)

The relationship between forces and potential is a simple consequence of using the Lagrangian formulation, stating that if a force is conservative, it derives from a scalar potential field U, such that $F = -\nabla U$. Over the past 25 years, the potential field theory was investigated further under two aspects: analytical and numerical.

II - 3 - 1 - Potential Functions - Analytical Expression

The reason for the Potential Field Theory success comes from its relative conceptual simplicity, and it was first natural to the scientific community to try to formulate analytically appropriate repulsive and attractive functions. A concrete example using potential function formulation can be found in Ref. [15]. There, Passino created a two-dimensional geometric environment where a robot with an obstacle detection radius R detects obstacles modeled as points and whose repulsive actions are described as Gaussian functions $G_{o,i}(\bar{x})$ decaying with distance. This

means that as the robot evolves among obstacles, its knowledge of the world is only a subset of all the possible location that can be occupied in the world, and its position vector $\bar{x}(t)$ only takes values in a disc of radius R. This also means that for a given robot position vector $\bar{x}(t)$ and a given point obstacle position $\bar{o}_i$, $d(\bar{x},\bar{o}_i)$ is the Euclidian distance separating the obstacle o_i centered at $\bar{o}_i$ from the current robot position $\bar{x}(t)$. The analytical expression of these Gaussian functions is

$$G_{o,i}(\bar{x},t) = e^{-cd^2[\bar{x}(t),\bar{o}_i]} \tag{2.3.1}$$

where c is a scalar used to tune the repulsive effect of the obstacles on the robot.

Equation (2.3.1) is general enough so that only obstacle centers change from one $G_{o,i}$ to another. Obviously each individual $G_{o,i}$ is maximum when the robot position $\bar{x}(t)$ coincides with a given obstacle center $\bar{o}_i$. For a robot evolving in a two-dimensional world with n obstacles and a position $\bar{x}(t)$, it is important for the robot to determine which of the n threats are the most dangerous ones. This is done by taking the maximum value of the assemblage of the n $G_{o,i}$ evaluated at $\bar{x}(t)$. Hence the most immediate threat for the robot placed at position $\bar{x}(t)$ is

$$F_O[\bar{x}(t)] = \max_i \{G_{o,i}[\bar{x}(t)]\} \tag{2.3.2}$$

Solving equation (2.3.2) consists in finding the maximum of a multi-variable function, an analytical problem which does not always have a solution. However, if the two-dimensional world in which the robot evolves is discretized, finding a solution to equation (2.3.2) simplifies in finding the maximum of a collection of scalar values. F_O represents the maximum threat for

the robot given a circular area bounded by the disc of detection of radius R centered at $\bar{x}(t)_c$, the robot's current position. Equation (2.3.2) alone does not account for the attraction of the robot to the goal. In order to model this second fictitious force, author of Ref. [15] uses the square distance to the goal, such that the attraction to the goal is minimizing F_g given by

$$F_g\left[\bar{x}(t)\right] = d^2\left\{\bar{x}(t), \bar{g}\right\} \tag{2.3.3}$$

In equation (2.3.3), the distance to the goal appears as a quadratic term. Making the distance function as a quadratic function has the effect of emphasizing the value taken by the distance function to minimize. The threat modeled by equation (2.3.2) and the positive attraction represented by equation (2.3.3) must both be minimized among the sampled points in $D\left[\bar{x}(t)_c, R\right]$, but they play antagonistic roles. Thus, weights for each objective (obstacle avoidance, goal attraction) must be applied. If w_o is the weight for obstacle avoidance and w_g is the weight used for goal attraction, equations (2.3.2) and (2.3.3) can be combined in a cost function J such that, for any $\bar{x}(t) \in D\left[\bar{x}(t)_c, R\right]$,

$$\begin{aligned} J\left[\bar{x}(t)\right] &= \min\left(w_g \cdot F_g\left[\bar{x}(t)\right] + w_o \cdot F_O\left[\bar{x}(t)\right]\right) \\ J\left[\bar{x}(t)\right] &= \min\left\langle w_g \cdot d^2\left\{\bar{x}(t), \bar{g}\right\} + w_o \cdot \max_i\left\{G_{o,i}\left[\bar{x}(t)\right]\right\}\right\rangle \end{aligned} \tag{2.3.4}$$

As a result of its time-varying nature, equation (2.3.4) needs to be updated for any new position of the robot as a function of time.

In his book, Passino uses 6 obstacles. Using equation (2.3.4), the guidance to the goal follows the algorithm presented next.

```
for N simulation steps
1)    x ←robot's current position
2)    update discrete circle points of circle_sensors(x, r_sensors)
3)    apply cost functions for all discrete circle points
      find the steering angle for which the cost function local
            minimum was found
4)    update the robot's dynamic using that steering angle
end
```

Algorithm 1 - Guidance Algorithm Used in the Potential-Field-Based Example

The guidance described by Algorithm 1generates very different results depending on the values of w_o and w_g. Next is a picture reflecting such fact.

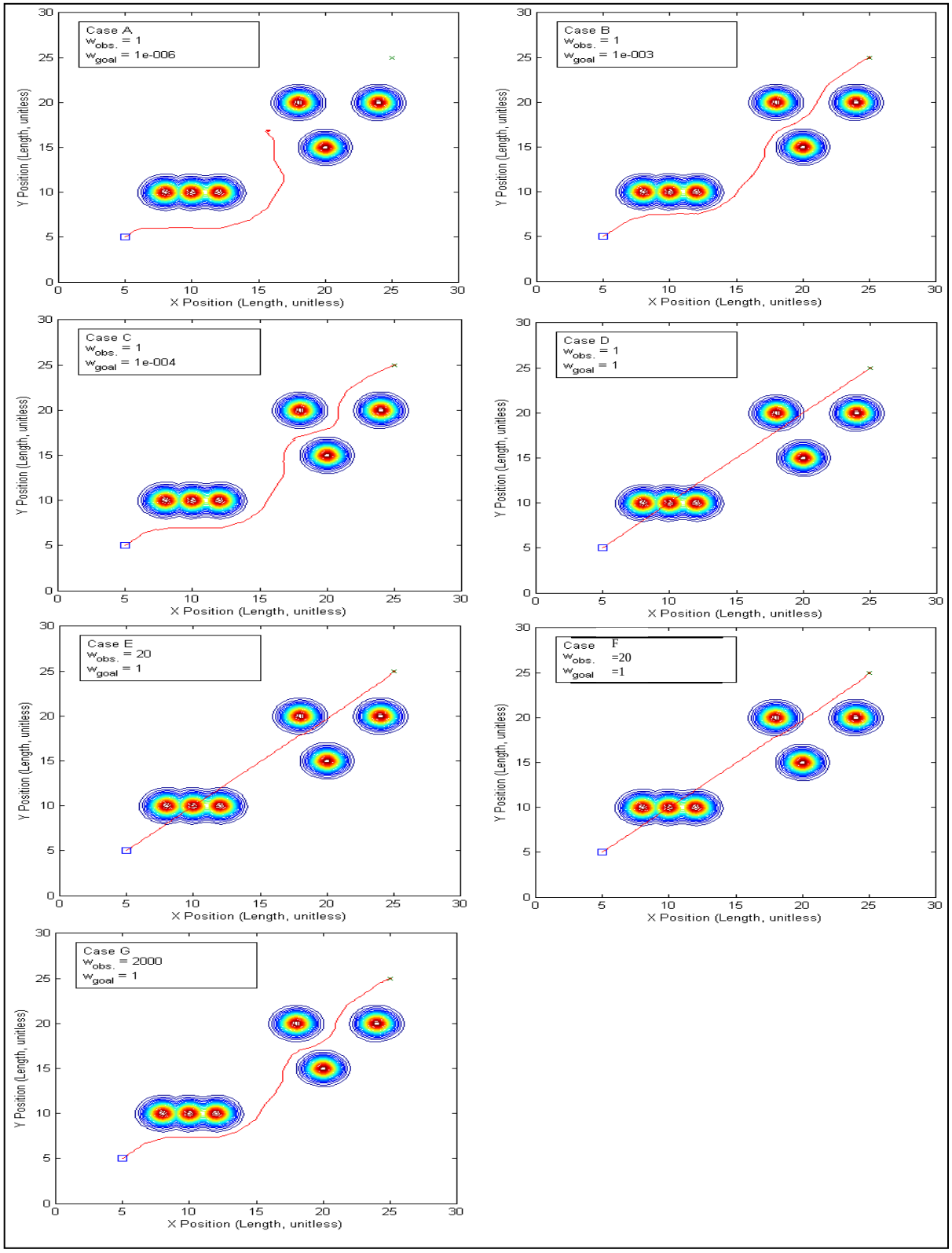


Figure 2 - Effects of Varying W_g and W_o on the Overall Guidance Driven by the Cost Function

Figure 2 shows the weights' highly heuristic impact on the cost function described by equation (2.3.4) and used in the guidance described by Algorithm 1. For cases A to G, it can be observed that only cases B, C, F, and G allow the goal to be reached while avoiding all obstacles. For those cases, the dominance of w_o over w_g is such that $1\times10^3 \le w_o / w_g \le 1\times10^4$, but in case of case A, $w_o / w_g = 1\times10^6$, obstacle avoidance is so much favored compared to goal reachability that the goal is not reached. This shows the heuristic nature of weight decision for equation (2.3.4). The process of tuning a cost function is a common practice in engineering, such as is the case when designing Linear Quadratic Regulator LQR controllers for aircraft. However, stratifying different heuristic-based layers of decision for control and obstacle/collision avoidance targeting a specific UAS makes any reconfiguration of those heuristics a multi-varied complex task. In addition, the potential-function-modeled cost function approach used by Passino shows its deficiency in the case of a fast-moving fixed-wing UAS, as the required time to find an adequate set of weights may take too long for that same UAS to avoid a potential threat.

Another problem with analytical design of multi-varied potential functions is the one known as the "local minima" problem. From multi-variable analysis, the only class of functions which guarantee a single minimum are harmonic functions, which satisfy the Laplace equation. This is a desirable feature when formulating equations such as(2.3.4): the minimum is the goal to reach, and there cannot be more than one minimum, otherwise the consequence is the formation of local minima. To better illustrate the problem, one can observe the result of reconfiguring obstacle positions and using the same original weights as those used in the example of Ref. [15]. Illustration of the problem is given in the next figure.

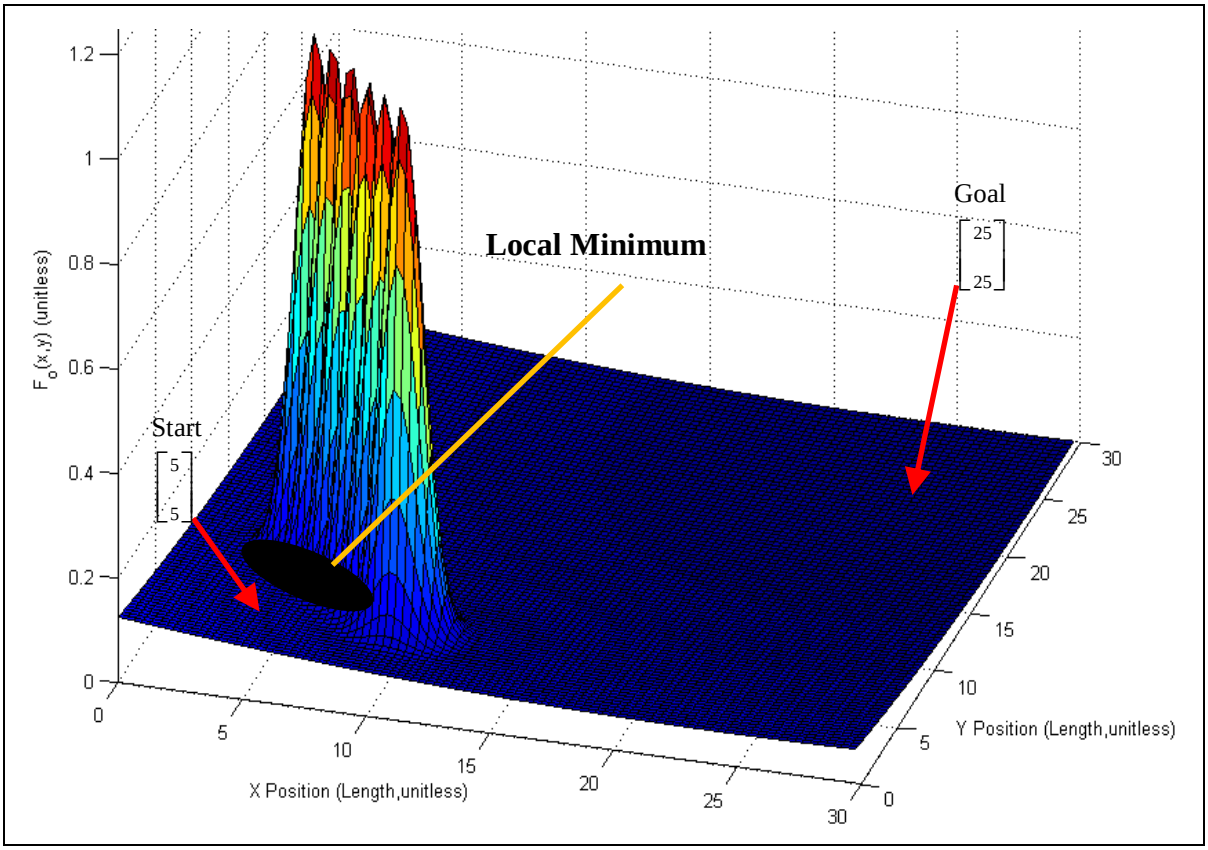


Figure 3 - Local Minimum Introduced in Cost Function

Figure 3 shows the effect of the wall of obstacles creating a local minima just before the wall. The goal function creates a surface descending from the initial position to the goal position, whereas the hills of the obstacle function add local minima to the descending surface. Consequently, in this particular case, the potential field driven guidance gets "trapped" by local minima: around the hills, there is no more unique position for the best avoidance of obstacles. This is best shown by the corresponding trajectory picture.

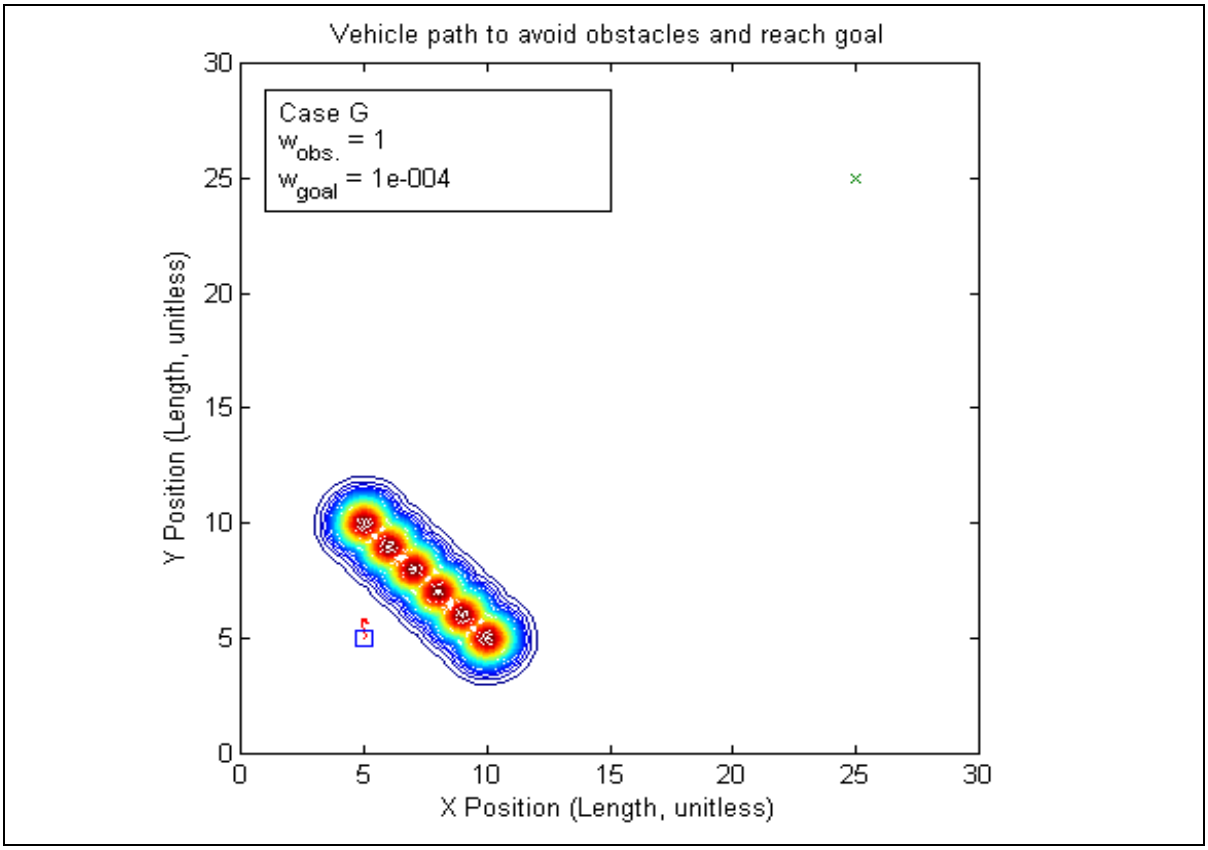


Figure 4 - Effect of Creating Artificial Local Minima Through a "Wall" of Obstacles

As can be seen in Figure 4 the "wall" of obstacles (very close one another) has the effect of blocking the guidance described in Algorithm 1 such that the robot stagnates close its initial position.

II - 3 - 2 - Potential Functions - 3D Grid Formulation

As mentioned previously, harmonic functions guarantee a unique minimum if used in the context of multi-varied potential functions. An idea developed by the authors of Ref. [16] is to derive the tri-dimensional discrete Laplacian equation, and use it to define some values in a tri-dimensional sampled grid of points. The derived Laplacian serves as a way to iteratively determine grid point values based on the initial boundary conditions imposed by the presence of obstacles. Then a negative gradient is applied to the Laplacian, and finally a steepest descent code is used to find the collision-free path. The initial boundary conditions are set numerically to:

- (**0**) for obstacles
- (+**1x10**$^{-300}$**)** for grid points around obstacles
- (**-1**) for the goal region of goal point

In the case of a scalar function U of real-valued variables x, y, and z, the Laplace equation can be written

$$\nabla^2 U = 0$$

The general expression of $\nabla^2 U$ however does involve cross-derivatives which are not necessarily identically null. Hence, in the more general case, the expression of $\nabla^2 U$ is given by

$$\nabla^2 U = \nabla\left(\nabla U\right) = \nabla\left(\frac{\partial U}{\partial x} + \frac{\partial U}{\partial y} + \frac{\partial U}{\partial z}\right)$$

$$\because \nabla^2 U = \frac{\partial}{\partial x}\left(\frac{\partial U}{\partial x} + \frac{\partial U}{\partial y} + \frac{\partial U}{\partial z}\right) + \frac{\partial}{\partial y}\left(\frac{\partial U}{\partial x} + \frac{\partial U}{\partial y} + \frac{\partial U}{\partial z}\right) + \frac{\partial}{\partial z}\left(\frac{\partial U}{\partial x} + \frac{\partial U}{\partial y} + \frac{\partial U}{\partial z}\right)$$

$$\because \nabla^2 U = \frac{\partial^2 U}{\partial^2 x} + \frac{\partial^2 U}{\partial^2 y} + \frac{\partial^2 U}{\partial^2 z} + \left(\frac{\partial^2 U}{\partial x \partial y} + \frac{\partial^2 U}{\partial y \partial x}\right) + \left(\frac{\partial^2 U}{\partial x \partial z} + \frac{\partial^2 U}{\partial z \partial x}\right) + \left(\frac{\partial^2 U}{\partial y \partial z} + \frac{\partial^2 U}{\partial z \partial y}\right)$$

The general expression of $\nabla^2 U$ is thus

$$\nabla^2 U = \frac{\partial^2 U}{\partial^2 x} + \frac{\partial^2 U}{\partial^2 y} + \frac{\partial^2 U}{\partial^2 z} + \left(\frac{\partial^2 U}{\partial x \partial y} + \frac{\partial^2 U}{\partial y \partial x}\right) + \left(\frac{\partial^2 U}{\partial x \partial z} + \frac{\partial^2 U}{\partial z \partial x}\right) + \left(\frac{\partial^2 U}{\partial y \partial z} + \frac{\partial^2 U}{\partial z \partial y}\right) \qquad (2.3.5)$$

Applying the definition of the Laplace equation to equation (2.3.5) yields

$$\frac{\partial^2 U}{\partial^2 x} + \frac{\partial^2 U}{\partial^2 y} + \frac{\partial^2 U}{\partial^2 z} = \left(\frac{\partial^2 U}{\partial x \partial y} + \frac{\partial^2 U}{\partial y \partial x}\right) + \left(\frac{\partial^2 U}{\partial x \partial z} + \frac{\partial^2 U}{\partial z \partial x}\right) + \left(\frac{\partial^2 U}{\partial y \partial z} + \frac{\partial^2 U}{\partial z \partial y}\right) \qquad (2.3.6)$$

The grid the authors of [16] use is initially only set according to the values defining the boundary conditions (walls and obstacles). The rest of the unassigned values are set by iteratively applying the definition of the Laplace equation in the discrete case. In order to convert equation (2.3.6) to the discrete case, finite differences are used. The following equations are expressions of the first derivatives in the discrete version.

$$\frac{\partial U}{\partial x} \underset{fwd.\ diff.}{\rightarrow} \frac{U\left(x[i+1], y[i], z[i]\right) - U\left(x[i], y[i], z[i]\right)}{x[i+1] - x[i]}$$

$$\frac{\partial U}{\partial y} \underset{fwd.\ diff.}{\rightarrow} \frac{U\left(x[i], y[i+1], z[i]\right) - U\left(x[i], y[i], z[i]\right)}{y[i+1] - y[i]}$$

$$\frac{\partial U}{\partial z} \underset{fwd.\ diff.}{\rightarrow} \frac{U\left(x[i], y[i], z[i+1]\right) - U\left(x[i], y[i], z[i]\right)}{z[i+1] - y[i]}$$

Second derivatives involve the same process with the addition of backward differences, that is, in the case of $\frac{\partial U_{art}}{\partial x}$ (U_{art} standing for "artificial potential")

$$\frac{\partial}{\partial x}\left(\left(\frac{\partial U_{art}}{\partial x}\right)_i\right)_i = \frac{\partial}{\partial x}\left(\frac{U_{art}\left(x[i+1],y[i],z[i]\right)-U_{art}\left(x[i],y[i],z[i]\right)}{x[i+1]-x[i]}\right)_i$$

$$\therefore \frac{\partial}{\partial x}\left(\left(\frac{\partial U_{art}}{\partial x}\right)_i\right)_i = \frac{U_{art}\left(x[i+1],y[i],z[i]\right)-U_{art}\left(x[i],y[i],z[i]\right)}{\left(x[i+1]-x[i]\right)\left(x[i]-x[i-1]\right)} -$$

$$- \frac{U_{art}\left(x[i],y[i],z[i]\right)-U_{art}\left(x[i-1],y[i],z[i]\right)}{\left(x[i+1]-x[i]\right)\left(x[i]-x[i-1]\right)}$$

The latter equation involves backward and forward finite differences, but those differences are taken along the same gradient, so assuming a uniform grid in the x direction, $x[i+1]-x[i]=$ $x[i]-x[i-1]$. This allows the previous equation to be rewritten as

$$\frac{\partial}{\partial x}\left(\left(\frac{\partial U_{art}}{\partial x}\right)_i\right)_i = \frac{U_{art}\left(x[i+1],y[i],z[i]\right)-U_{art}\left(x[i],y[i],z[i]\right)}{\left(x[i+1]-x[i]\right)^2}$$

$$-\frac{U_{art}\left(x[i],y[i],z[i]\right)-U_{art}\left(x[i-1],y[i],z[i]\right)}{\left(x[i+1]-x[i]\right)^2} =$$

$$\frac{U_{art}\left(x[i+1],y[i],z[i]\right)-U_{art}\left(x[i],y[i],z[i]\right)-U_{art}\left(x[i],y[i],z[i]\right)+U_{art}\left(x[i-1],y[i],z[i]\right)}{\left(x[i+1]-x[i]\right)^2}$$

Therefore, because finite differences are always computed along a same gradient for a given direction, and because the grid used for those differences is uniformly sampled,

$$\frac{\partial}{\partial x}\left(\left(\frac{\partial U_{art}}{\partial x}\right)_i\right)_i = \left(\nabla U_{art}\big|_x\right)\Big|_x =$$

$$\Delta U_{art}\big|x \underset{\substack{\text{uniformly}\\ \text{sampled}\\ \text{grid, finite}\\ \text{differences}\\ \text{along same}\\ \text{gradient}}}{\equiv} \frac{U_{art}\left(x[i+1],y[i],z[i]\right)-U_{art}\left(x[i],y[i],z[i]\right)-U_{art}\left(x[i],y[i],z[i]\right)+U_{art}\left(x[i-1],y[i],z[i]\right)}{\left(x[i+1]-x[i]\right)^2}$$

(2.3.7)

Also, by definition of numerical differentiation on a uniform grid, the mixed-derivatives vanish.

Originally unassigned grid point values are then assigned iteratively by using the tri-dimensional discrete Laplacian equation whose derivation is given next. By reusing the concept introduced by Khatib, repulsive and attractive forces are fictitious and set artificially, hence the subscript "art" for the potential they derive from.

$$\nabla U_{art} = \frac{U_{art}\left(x[i+1], y[i], z[i]\right) - U_{art}\left(x[i], y[i], z[i]\right)}{x[i+1] - x[i]} + \frac{U_{art}\left(x[i], y[i+1], z[i]\right) - U_{art}\left(x[i], y[i], z[i]\right)}{y[i+1] - y[i]} + \frac{U_{art}\left(x[i], y[i], z[i+1]\right) - U_{art}\left(x[i], y[i], z[i]\right)}{z[i+1] - y[i]}$$

and,

$$\nabla\left\{\frac{U_{art}\left(x[i+1], y[i], z[i]\right) - U_{art}\left(x[i], y[i], z[i]\right)}{x[i+1] - x[i]}\right\} = \frac{U_{art}\left(x[i+1], y[i], z[i]\right) - U_{art}\left(x[i], y[i], z[i]\right)}{x[i+1] - x[i]} - \frac{U_{art}\left(x[i], y[i], z[i]\right) - U_{art}\left(x[i-1], y[i], z[i]\right)}{x[i] - x[i-1]}$$

Likewise, using the same assumptions as those used to derive equation (2.3.7),

$$\nabla\left\{\frac{U_{art}\left(x[i], y[i+1], z[i]\right) - U_{art}\left(x[i], y[i], z[i]\right)}{y[i+1] - y[i]}\right\} = \frac{U_{art}\left(x[i], y[i+1], z[i]\right) - U_{art}\left(x[i], y[i], z[i]\right)}{y[i+1] - y[i]} - \frac{U_{art}\left(x[i], y[i], z[i]\right) - U_{art}\left(x[i], y[i-1], z[i]\right)}{y[i] - y[i-1]}$$

And

$$\nabla\left\{\frac{U_{art}\left(x[i],y[i],z[i+1]\right)-U_{art}\left(x[i],y[i],z[i]\right)}{z[i+1]-z[i]}\right\}=$$

$$\frac{U_{art}\left(x[i],y[i],z[i+1]\right)-U_{art}\left(x[i],y[i],z[i]\right)}{z[i+1]-z[i]}-$$

$$\frac{U_{art}\left(x[i],y[i],z[i]\right)-U_{art}\left(x[i],y[i],z[i-1]\right)}{z[i]-z[i-1]}$$

Consequently,

$$\begin{aligned}\nabla^2 U_{art} \quad = \quad & \frac{U_{art}\left(x[i+1],y[i],z[i]\right)-U_{art}\left(x[i],y[i],z[i]\right)}{x[i+1]-x[i]} \\ & -\frac{U_{art}\left(x[i],y[i],z[i]\right)-U_{art}\left(x[i-1],y[i],z[i]\right)}{x[i]-x[i-1]} \\ & +\frac{U_{art}\left(x[i],y[i+1],z[i]\right)-U_{art}\left(x[i],y[i],z[i]\right)}{y[i+1]-y[i]} \\ & -\frac{U_{art}\left(x[i],y[i],z[i]\right)-U_{art}\left(x[i],y[i-1],z[i]\right)}{y[i]-y[i-1]} \\ & +\frac{U_{art}\left(x[i],y[i],z[i+1]\right)-U_{art}\left(x[i],y[i],z[i]\right)}{z[i+1]-z[i]} \\ & -\frac{U_{art}\left(x[i],y[i],z[i]\right)-U_{art}\left(x[i],y[i],z[i-1]\right)}{z[i]-z[i-1]}\end{aligned} \tag{2.3.8}$$

In equation (2.3.8), the red-boxed terms cancel out because of the assumption that the tridimensional grid is made of unit cubes and finites differences follow a same gradient along the x, y, or z directions, that is,

$$x[i+1]-x[i]=x[i]-x[i-1]=y[i+1]-y[i]=y[i]-y[i-1]=z[i+1]-z[i]=z[i]-z[i-1]=1,$$

and all the red-boxed terms cancel one another.

By definition of the Laplace equation, $\nabla^2 U_{art} = 0$, so equation (2.3.8) can be rewritten as

$$
\begin{aligned}
&+U_{art}\left(x[i+1], y[i], z[i]\right) + U_{art}\left(x[i-1], y[i], z[i]\right) + \\
&U_{art}\left(x[i], y[i+1], z[i]\right) + U_{art}\left(x[i], y[i-1], z[i]\right) + \\
&U_{art}\left(x[i], y[i], z[i+1]\right) + U_{art}\left(x[i], y[i], z[i-1]\right) - \\
&6U_{art}\left(x[i], y[i], z[i]\right) = 0
\end{aligned}
$$

Which itself can be rewritten as

$$
\begin{aligned}
U_{art}\left(x[i], y[i], z[i]\right) \quad = \quad & \frac{1}{6}\Big\{U_{art}\left(x[i+1], y[i], z[i]\right) + U_{art}\left(x[i-1], y[i], z[i]\right) \\
& +U_{art}\left(x[i], y[i+1], z[i]\right) + U_{art}\left(x[i], y[i-1], z[i]\right) \\
& +U_{art}\left(x[i], y[i], z[i+1]\right) + U_{art}\left(x[i], y[i], z[i-1]\right)\Big\}
\end{aligned}
\qquad (2.3.9)
$$

Equation (2.3.9) simply states that the value corresponding to a given position (cell) of a tridimensional grid is the average of its six immediate surrounding ones. It is equation (2.3.9) which is iteratively used to find the remaining originally uninitialized values of the initial grid based on the initial conditions. This process of averaging is then repeated N times until the solution is considered to be stable (little numerical fluctuations). Applying this averaging principle to the example presented in Ref. [15] produces the next figure.

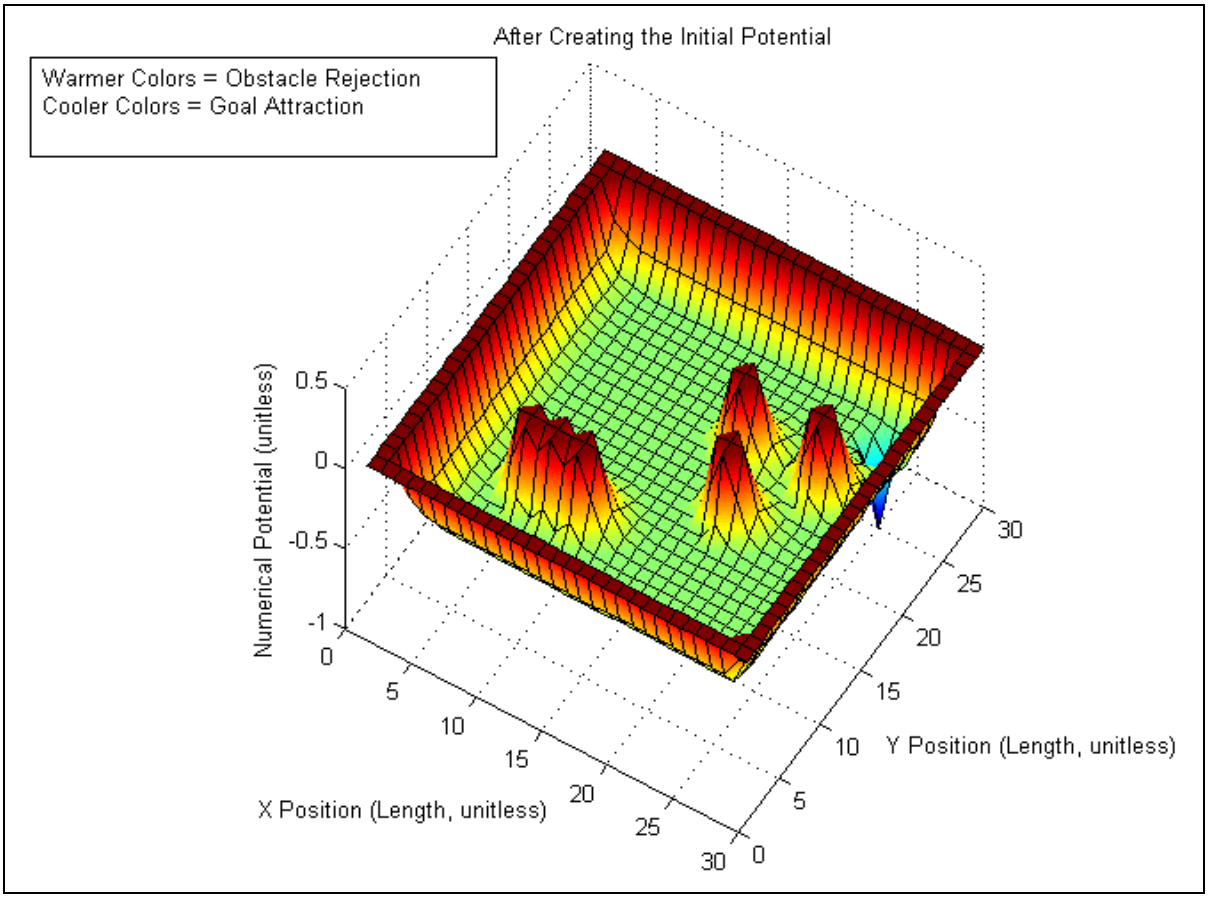


Figure 5 - Numerical Laplacian Produced After One Averaging Process for the "Passino" Problem

Figure 5 shows a topology similar to the one depicted in Figure 2, and is a harmonic function found numerically. The significance of Figure 5 is that an imposed numerical solution to the Laplace equation yields a numerical scalar potential whose topology when graphed is identical in trend to the result provided using analytical formulation and avoiding the local minima problem at the same time. It consequently proves that imposing the Laplace equation will produce guidance results similar to those of the previous example while never blocking the vehicle. The repulsive and attractive effects are embedded within the boundary conditions, and carried on over the iterative Laplacian construction. The function found numerically is harmonic because it

satisfies the Laplace equation. Based on the previously defined Lagrangian formalism definition, a negative gradient to the Laplacian of Figure 5 yields the attractive effect of the goal and the repulsive effects of the obstacles. This is best viewed in the next picture.

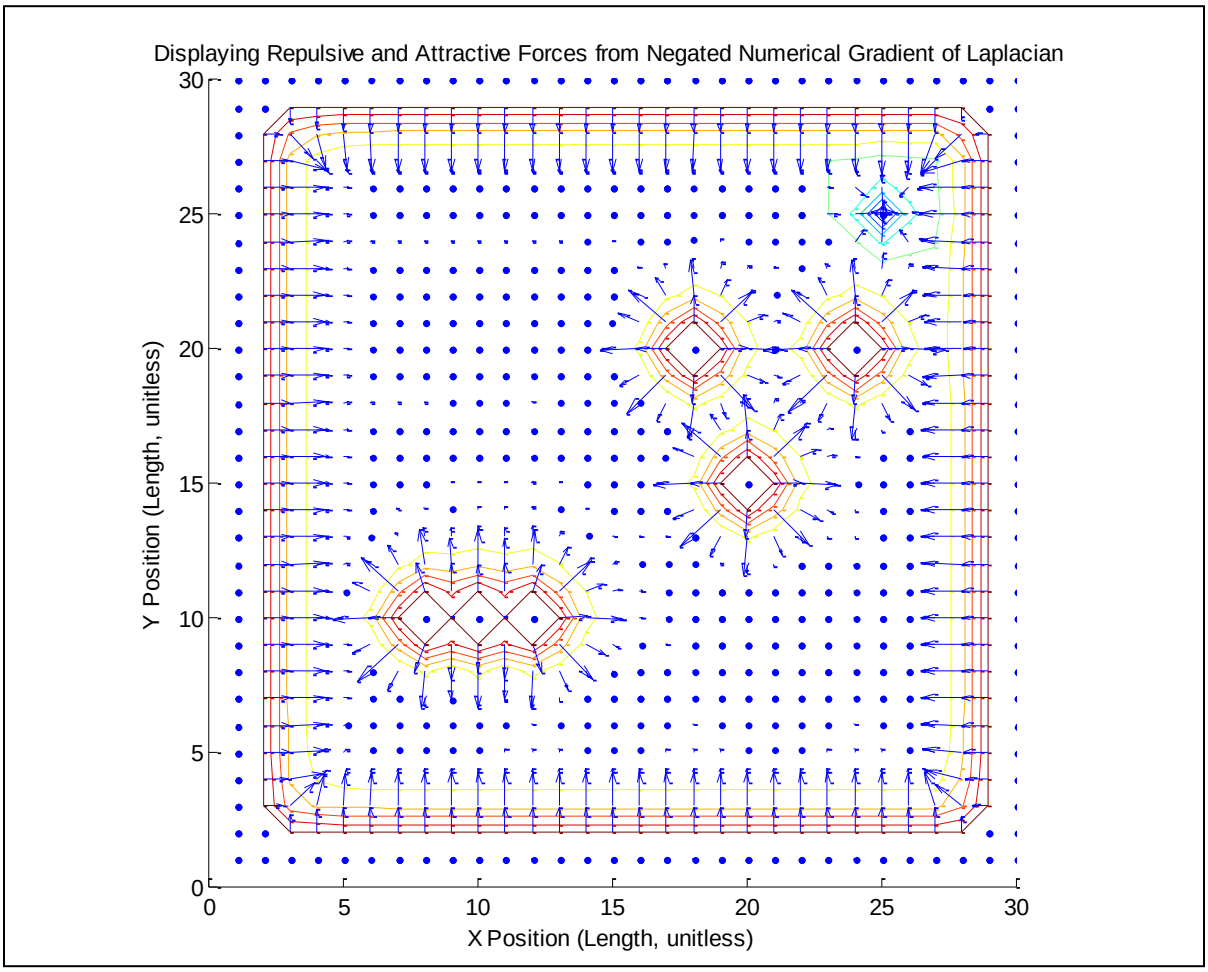


Figure 6 - Attractive and Repulsive Forces Shown as Inward Outward Pointing Vectors

Once arrived at the step described by Figure 6, a gradient descent is ran to find the resulting collision free path. As opposed to the analytical formulation, the heuristic of this method only lays in the number of iterations set for the Laplacian solution to be considered stable. Authors of Ref. [16] do mention that they use a multigrid process to accelerate the time needed to find a converging solution. This multigrid approach is described in Ref. [17].

II - 4 - Geometric Approaches

By simple geometric consideration, what is meant is that given a scenario involving potential collisions between a vehicle and other obstacles (such as mountains or even dynamic obstacles like other aircraft), basic vector calculus is the foundation of what is used to trigger avoidance maneuvers. This is the approach introduced by Dr. John S. Morrell of former Bendix Corporation when he developed a collaborative collision avoidance logic based on the time of closest approach τ.

In Refs. [18, 19] authors developed a method to share the airspace in case of conflict. The method is restricted to a coplanar encounter between two aircraft flying at constant speed. Assumption is made that only one of the two aircraft is cooperating to avoid collision while the other aircraft maintain its heading. Authors developed a three step approach, summarized below:

- Estimate a point of closest approach (CPA)
- Estimate a time of closest approach, τ
- Based on the time constant τ, issue a maneuver for avoidance

The first step allows to estimate a distance, and the second step allows to find a time constant, τ, which if positive leads the cooperative aircraft to consider an avoidance maneuver. If $\tau > 0$, the avoiding aircraft must estimate another quantity, the miss distance vector $\overrightarrow{r_m}$. Depending on whether the magnitude of that vector is greater or equal to zero, an escaping maneuver may be issued.

The point of closest approach is a concept that needs perfect understanding in order to better grasp the logic used to issue a potential collision avoidance maneuver. In order to define CPA, the following scenario is considered:

Aircraft A and B are initially located at GPS positions $P_{A,0}$ and $P_{B,0}$ and flying at velocities $\underline{V_A}$ and $\underline{V_B}$. Because of such assumptions, at any instant t, the position of each UAS can be described as:

$$\begin{cases} P_A(t) &= P_{A,0} + t\underline{V_A} \\ P_B(t) &= P_{B,0} + t\underline{V_B} \end{cases} \tag{2.4.1}$$

The next figure helps to visualize the simplified scenario.

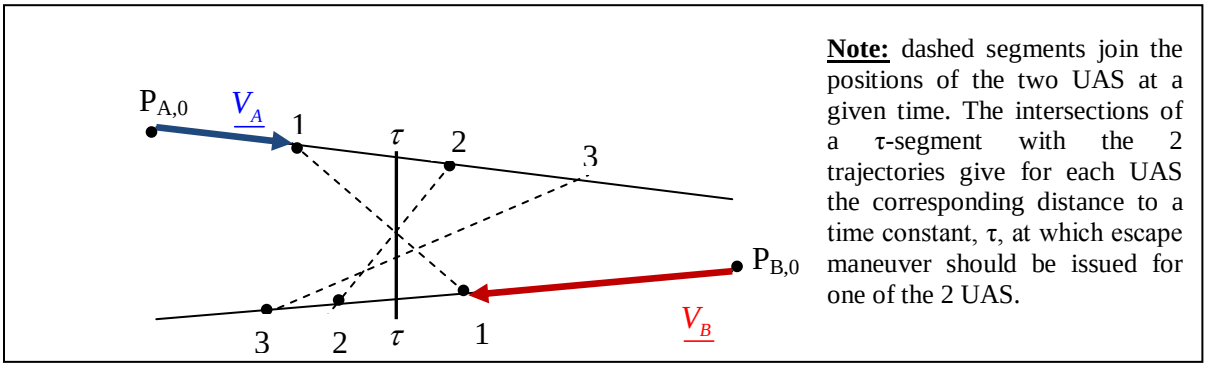


Figure 7 - Generalized Diagram Showing the Two Point Mass Objects and the Closest Point of Approach

The Closest Point of Approach (CPA) refers to the positions at which two dynamically moving objects (in the present case UAS A and B) reach their closest possible distance. If the two moving objects are considered to be point-masses, moving at fixed speed, toward fixed locations, then the two points are moving along two lines in space. A common misconception is to think of the closest possible distance as the closest distance between two lines. **The closest distance between two lines and the closest possible distance between two dynamically moving points are two different concepts**. The closest possible distance between two moving points is the

shortest distance between two line segments **at the same time t**. Because CPA is inherent to the behavior of two dynamic objects, a diagram showing the calculated location of a CPA can somehow look erroneous, as it appears to be an attempt at representing the shortest distance between two line segments.

In Figure 7, the CPA can be placed on each intersection of the τ line with the UAS trajectories, depending whether the coordinate system is fixed on UAS A or UAS B. If the reference frame is placed on the UAS B, then the CPA lies on the trajectory of UAS B. If UAS B is chosen to be the reference frame, then the CPA would be the projection of $P_A(t)$ onto the trajectory of UAS B, at time t= τ . Since τ is the time at which UAS A and B would reach the closest distance, it is referred to as the **time to closest approach**.

Based on equation(2.4.1), the distance vector $\vec{r}$ separating each UAV can be expressed, at any time t, as

$$\begin{aligned} \underline{r} &= P_A(t) - P_B(t) \\ \underline{r} &= \underbrace{\left(P_{A,0} - P_{B,0}\right)}_{-\underline{r_0}} + t\underbrace{\left(\underline{V_A} - \underline{V_B}\right)}_{-\underline{c}} \end{aligned}$$

Based on the above derivation, two new vectorial quantities can be defined:

- $\underline{r_0}$, the distance vector between the two UAS at an arbitrary initial time of observation
- $\underline{c}$, velocity vector of the non cooperative UAS B relative to the cooperative UAS A

Using the above definitions, the distance between the two UAS A and B, at any point in time, can be recast as:

$$\underline{r} = -\left(\underline{r}_0 + \underline{c} \cdot t\right) \tag{2.4.2}$$

Based on the above expression, it possible to obtain the derivative of distance between the two UAS, and solve it for the case where it equals zero, so the corresponding time at which the two UAS would be in CPA, τ, can be obtained.

The distance between the two UAS, raised at the power two, can then be expressed as:

$$d^2\left(A,B\right)=\left\|\underline{r}\right\|^2=\left(\underline{c}.\underline{c}\right)t^2+(2\underline{r}_0.\underline{c})t+\underline{r}_0.\underline{r}_0$$

And by taking the derivative of $\left\|\underline{r}\right\|^2$,

$$\frac{\partial\left[d^2\left(A,B\right)\right]}{\partial t}=2\left(\underline{c}.\underline{c}\right)t+(2\underline{r}_0.\underline{c}) \tag{2.4.3}$$

Solving for (2.4.3)=0 yields

$$2\left(\underline{c}.\underline{c}\right)t+(2\underline{r}_0.\underline{c})=0\Leftrightarrow\begin{cases} t & = & \tau \\ \tau & = & -\dfrac{\vec{r}_0.\underline{c}}{\underline{c}.\underline{c}} \end{cases}$$

$$\tau \quad = \quad -\frac{\underline{r}_0.\underline{c}}{\underline{c}.\underline{c}} \tag{2.4.4}$$

Equation (2.4.4) gives the time at which UAS A and B would be in CPA configuration. While τ is greater than zero, collision between the two UASs is not imminent: there is some time ahead before a collision potentially occurs. However, if τ becomes negative, then other parameters such as the miss distance vector must be investigated, since an escape maneuver may be needed. Based on equation (2.4.4) it is also possible to express the miss distance vector $\underline{r_m}$, whose norm is the distance for the cooperative UAS A between its current position and the its CPA configuration. This is best captured by the next picture.

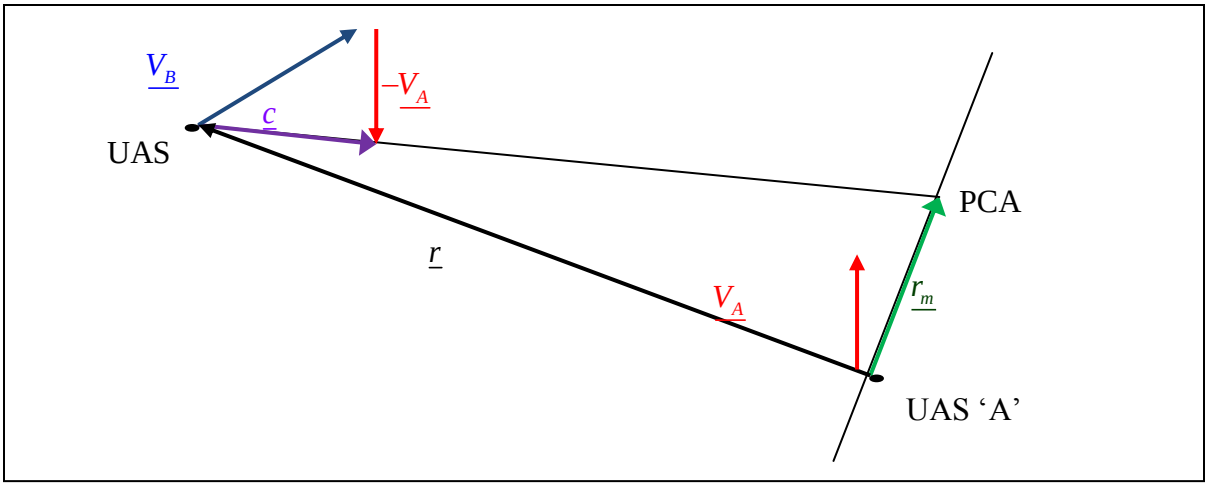


Figure 8 - Miss distance vector with UAV B cooperative and UAV A uncooperative

Based on Figure 8,

$$\underline{r_m} = \underline{r} + \tau \underline{c} \tag{2.4.5}$$

As explained in Ref. [18], each UAS defines a radius of safety, $r_{safe} = 5$ nautical miles (nm). However, in the case of the 33% scale Yak 54, that same safety radius would be within the 0.25 to 0.5 nm range, and would vary as a function of speed and the airspace flown. If $r_{safe} > r_m$, evasion maneuvers are issued.

The geometric approach presented here is very similar to the core of the logic of TCAS II modems as stated in Ref. [20]. The vehicles are assumed to have knowledge of theirs and other's inertial positions, and fly at constant speed. In terms of computation, only vector calculations are involved, making the approach very simple and fast. The escape maneuvers are not discussed here, but can be found in Ref. [18], and are a set of heuristics.

II - 5 - Graph Constructions

This section regroups two techniques which are usually not classified under the same category in the scientific literature (Ref. [21]): 2.5 D space decompositions and probabilistic roadmaps. The reason behind being presented here in the same section is because although their respective algorithms are different, the two families use the same concepts.

2.5 D techniques are two-dimensional space decompositions based on specific algorithms. They are qualified as 2.5 dimensional because their algorithms operate on two-dimensional cases. In order to extend their use to 3D, 2D workspace extrusions must be used. Some of those 2.5 dimensional techniques use points as primitives, which once connected form roadmap-like networks, comparable to commercial roadmaps. These roadmaps are results of specific algorithms such as the "Visibility Graph", and are not to be confused with the generic "Probabilistic Roadmap" algorithm, whose logic is different from any of the available 2.5 D space decompositions. Both 2.5 D techniques and roadmap methods share the same fundamental concepts which are recapitulated in the following section.

II - 5 - 1 - Common Concepts to 2.5 D methods and Roadmaps

In the robotic terminology such as presented in Ref. [22], a tridimensional space where vehicles can evolve is referred to as a workspace W . Typically, a vehicle evolving in a workspace W needs to change its configuration, by moving its wheels in the case of a car, or its control surfaces for an aircraft. The subset of W where the robot can freely change configuration without colliding with obstacles is called the configuration space, referred to as Q in Ref. [22] but as C in Ref. [23]. In the rest of the discussion, the notation of C is used to refer to the configuration space.

The author of Ref. [23] explains in chapter 4.2 that C (also referred to as C-space) is a special state space. This is not to be confused with the state space representation used in Control Engineering. In the C-space, the robot can occupy certain positions, but transition between those positions is not accounted for. In Ref. [23], the state space X_{space} is a superset of C where configurations are results of inputs to a vehicle. Typically, 2.5 D decompositions of W use C as the framework for their respective algorithms, although some refinements can account for the continuum between two positions.

Most of 2.5 D and Roadmaps variants use the mathematical concept of a graph developed in Ref. [24]. 2.5 D techniques can associate points or bounded regions called geometric cells. Those elements are referred to as vertices V. An association or connection between two vertices is referred to as an edge E. Therefore, the mathematical structure used to develop paths among obstacles is a graph of vertices and edges, G(V,E). By definition of a graph, there can be more than one edge between two vertices: vertex A can be connected to C directly, or through edge AB and edge BC. This is an important feature in the sense that paths found using 2.5 D techniques or Probabilistic Roadmaps are not necessary optimal, and algorithms such as Dijkstra in Ref. [25], or "A Star" presented in Ref. [26] may be used when solving the shortest path problem for a given G(V,E).

II - 5 - 2 - 2.5 D Space Decomposition Methods

Many 2.5 D space decomposition methods are described in [22]. In [22], authors divide those 2.5 D methods into cell-based methods and roadmap methods. Perhaps the most popular decomposition algorithms for those two sub categories are respectively the "trapezoidal" decomposition and the "visibility graph". From a conceptual point of view, the main difference

between cell-decompositions and roadmap decompositions lies in the notion of coverage: cells define (cover) entire zones, as opposed to sampling points of the C-space for roadmaps.

The Visibility Graph Decomposition

The visibility graph decomposition algorithm detailed in [22] first appeared in 1979 with the publication of Ref. [27]: “An Algorithm for Planning Collision-Free Paths Among Polyhedral Obstacles”. The main idea of this algorithm is to decompose any polyhedral obstacle into a set of edges with distinct vertices. Once the decomposition is finished, a path-finding algorithm is applied to create a path from G(V,E) between the start and goal locations. The algorithm essentially sweeps a line from a starting point v, to all obstacles' vertices v_i. If the segment $[vv_i]$ does not intersect any obstacles' edges, v_i is visible from v. The result of the algorithm is a list of obstacle vertices visible to v for which it is safe to travel "nearby" without colliding with obstacles. The concept of nearby must be developed further: the decomposition itself works on the obstacles, but does not account for the vehicle's projected area/volume. Hence in the visibility graph technique, obstacles must be enlarged virtually so calculations on the graph account for the volume/area taken by the mobile agent. The side effect of the decomposition is that it is also safe to travel between two vertices v_i. Although the visibility graph algorithm is relatively simple, the authors of Ref. [27] clearly mention that "*adapting it to a 3D environment has a significant effect on the execution time of the algorithm"*. Growing obstacles even in two-dimensions can result in having two or more obstacles intersecting virtually, and additional geometric computations must be performed in order to merge the different obstacles (initially not intersecting) into a single one. In a tridimensional case this is equivalent to computing the convex Hull of the intersections' result. Further, another problem arises: the number of rays between two objects in 3D may be countless as stated in Ref. [28]. Authors of [28] then extend

the concept of a visibility ray to which of a visibility plane. In essence, such approach is equivalent to extruding a 2D world along an axis, and therefore is the reason why the visibility graph, even when adapted to work in three dimensions, actually is a 2.5 D algorithm.

The Trapezoidal Decomposition

The trapezoidal decomposition is a cell-based decomposition, which means that as opposed to the Visibility Graph, the resulting graph of the decomposition is a set of vertices which are areas in 2D, volumes of extrusion in 3D, and which share connectivity properties. Cell decompositions account for coverage: They define areas or volumes where it is safe to change configuration for a robot, provided that the robot can fit within the cells. The decomposition itself is straightforward: a sweep line traverses a 2D land of obstacles and collects any potential intersecting obstacles' vertices. Each time at least one vertex is collected a vertical line along the parallel to the y-axis is added. After the sweeping line has swept all vertices, the result of the algorithm is a collection of trapezoids. When those trapezoids contain a portion of an obstacle, they cannot be traversed. Once the decomposition is performed, cells are connected based on the geometric edges they may share. The result is a graph G(V,E) where the vertices are cells and edges are actual geometric edges (or planes for the extruded tridimensional cases). An example of trapezoidal cell decomposition is given in Figure 10.

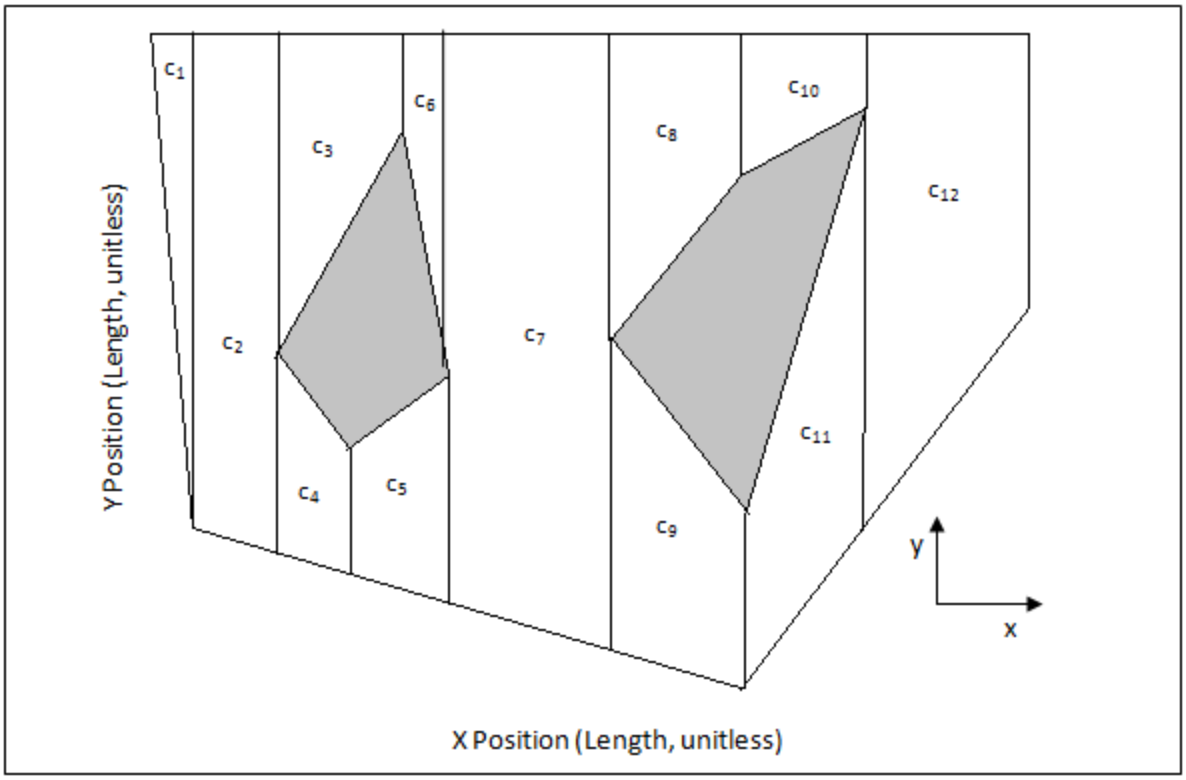


Figure 9 - Planar Trapezoidal Cell Decomposition

II - 5 - 3 - Probabilistic Roadmaps

Actual roadmaps are just areas of interest for people to travel to connected by roads. By abstracting this concept of roadmap in order to extend connections not just between areas but to a set of entities known to be safe, it is consequently possible to relate them together based on their connectivity properties. Connection between two entities is usually possible if those two entities belong to a convex set. The decomposition phase is known as the learning phase, while the connection phase is known as the query phase. These two phases are detailed in the next algorithm.

```
GENERATE_PRM( N, K )
1)      while N samples Q have not been picked
2)              q_rand ← RANDOM_SAMPLE( );
3)              if OBSTACLE_FREE( q_rand )
4)                      G.ADD_SAMPLE( q_rand );
5)              endIf
6)      endWhile

7)      For all q in G(V,E)
8)
        Nq ← FIND_A_NUMBER_OF_NEAREST_NEIGHBOR(
V, K )
9)              for all q' in N_q
10)                     if OBSTACLE_FREE_SEGMENT( q, q' )
11)                             MAKE_EDGE( q, q');
12)                     endIf
```

I - Learning

II - Query

Algorithm 2 - Probabilistic Roadmap Algorithm

Lines 1 through 6 of Algorithm 2 constitute the most general learning phase of a probabilistic roadmap: the samples that may belong to the roadmap are selected according to a uniform distribution (line 4). Line 7 through 14 correspond to the query phase. In the first phase, random samples are selected to construct a network of free locations for the robot, which is an attempt at mapping obstacle locations. The resulting network is a set of interconnected paths where the robot can be moved holonomically. This means that those selected positions are places where the robot can fit when maintained at rest, but not necessarily safe locations if the robot is to move from one location to another using non-holonomic equations of motions.

Those samples or positions can be viewed as vertices (nodes) of what could become a network of vertices. Such network, where edges connecting the nodes are constructed is called a graph, which explains why the resulting structures of phases I and II are referred to formally as G(V,E), the graph structure connecting vertices V with edges E. Phase II, where the network is actually

built from previous samples and new ones can be detailed further, especially line 8. In order to develop the concept of nearest neighbors, the Figure 10 is used.

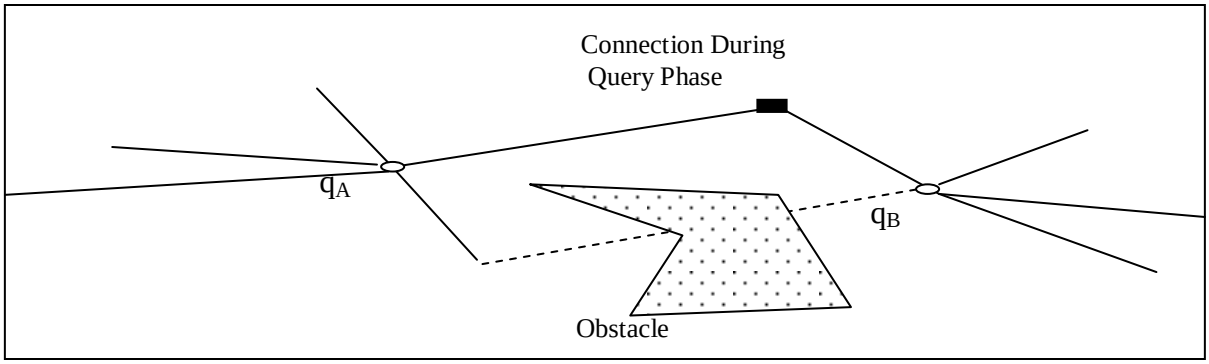


Figure 10 - Schematic for the Process of Connecting K = 5 Nearest Neighbors In the Query Phase of the PRM Construction

Figure 10 illustrates Algorithm 2 for the query phase and is a proof that the vertices picked during phase I can be connected in a suboptimal fashion. Indeed, based on Figure 10 it is possible to observe that vertices q_A and q_B could be connected through a straight line. However, q_B not being among the K=5 nearest neighbors of q_A, is not directly connected to q_A. Had K been set to be equal or greater than 6, a straight path from q_A to q_B, free of obstacles, could have been found. So clearly increasing K involves a better mapping, but also slows down the computation time. Also, if samples q_A, q_B and their K nearest neighbors are to be joined together, the resulting closed polygons are to be related with Voronoi diagrams. Voronoi regions V(x) are convex boundaries surrounding a point x such that any other point inside V(x) is closer to x than to any other point outside of V(x). In the case of q_A and assuming its K nearest neighbors are connected together, the parallel between nearest neighbors and Voronoi regions is clear: Voronoi diagrams for path planning allow defining optimized danger areas. A UAS not being inside a Voronoi cell is either already in another Voronoi cell, and has either crashed or managed to preserve a safe distance with the obstacle cluster in the previous cell. The alternative is that the

UAS approaching the next Voronoi cell should aim for the one with the greatest volume, because although that latter might contain many obstacles of large size, its surface also account for free space. Using K nearest neighbors in a PRM is the dual of using Voronoi diagrams for path planning: Voronoi diagrams are attempts at mapping danger zones optimally, K-sample selections are attempts at choosing optimized (but not optimal) free areas. Hence K-Sample selections approximate Delaunay triangulations, the dual construction of Voronoi cells optimally surrounding danger zones.

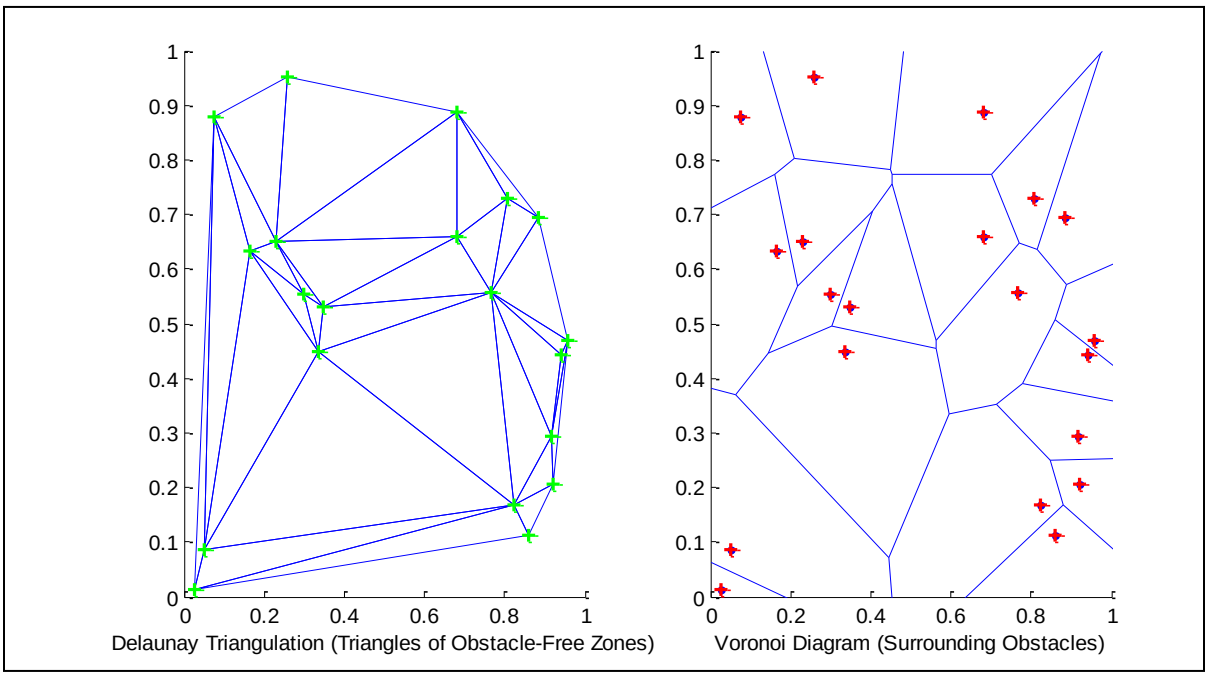


Figure 11 - Duality Between Delaunay Triangulation and Voronoi Diagram

Also, the result of a probabilistic roadmap being a graph, if elements connecting a starting point and a goal point exist, many paths can be found to connect the two extrema. This implies the need to develop a search graph in order to find the shortest path, such as Dijkstra [25] or its extension, "A Star" presented in Ref. [26]. Like for 2.5 D techniques, the resulting sample points lie exclusively in the configuration space. In Ref. [29], the authors run a PRM algorithm first to

get a coarse mapping of the environment where the UAS is supposed to fly through, and in a second

phase smooth out the found trajectory. However later on in the article the UAS used is found to be a helicopter, notable for being able to hover over places unlike fixed-wing UAS. However, in the case of fixed-wing UAS, path smoothing can actually introduce a previously non-existing collision as shown in the next picture.

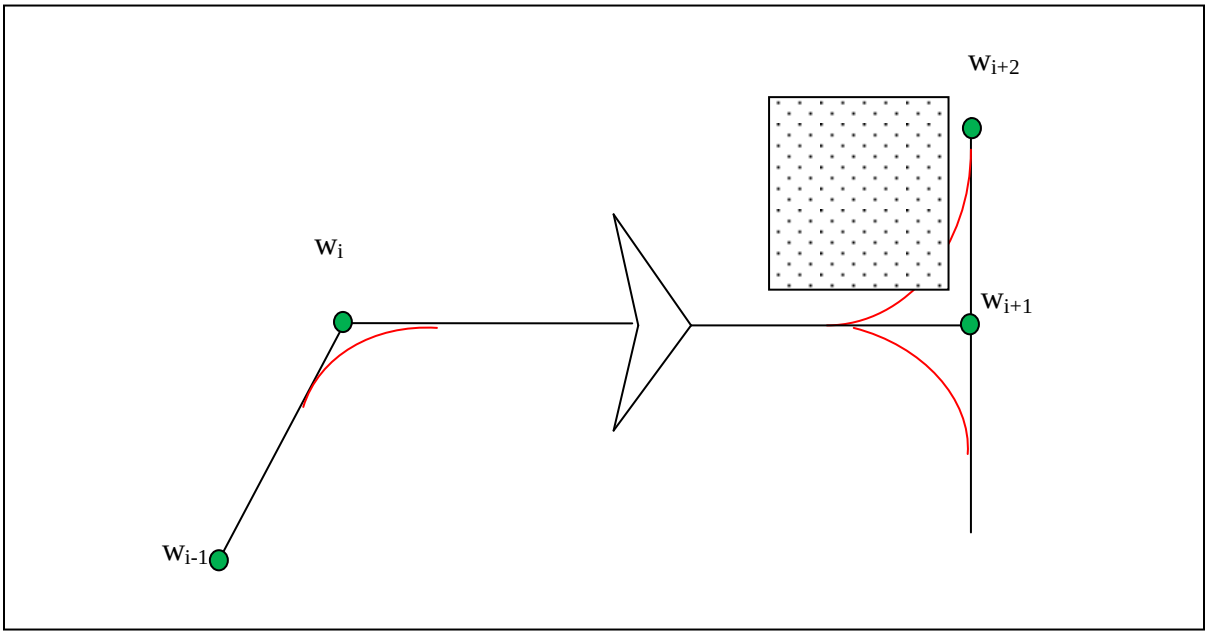


Figure 12 - The Dangers of Trajectory Smoothing

Figure 12 is the result of trajectory smoothing applied to the flight legs delimited by the waypoints in green. As it can be observed, the primary flight legs can sometimes represent unrealistic trajectories. Such is the casewith the connected segments [w_i w_{i+1}] and [w_{i+1} w_{i+2}] forming a right angle. Althought the right angle is impossible to achieve for fixed-wing UAS, its design has the benefit of getting around obstacles such as the dashed square. On the other hand, the red curves smooth the initial trajectory, but traverse the dashed square, and thus cannot be used for obstacle avoidance. This shows that flyable trajectories must be designed during real-time waypoint design, and not after. Also, those waypoints cannot be designed based

on pure geometric considerations only, but rather account for the dynamics of the vehicle in flight, directly or through inspection of parameters reflecting the capability of the UAS.

II - 6 - Rapidly Exploring Random Trees

Two major differences separate Rapidly-exploring Random Trees (RRTs) from Probabilistic Roadmaps:

- RRTs spread by using a differential model of a vehicle. This is part of RRTs' framework: spreads are results of inputs.

- RRTs develop as trees and not as graphs. Consequently, once a goal is met, joining an initial point to that goal does not involve path finding algorithms such as A star. It is just a process of back-tracking child to parent nodes.

The concept of RRTs first appeared in Ref. [30] and is reproduced in Algorithm 3 for the sake of discussion.

```
GENERATE_RRT( x_init, K, Δt )
    T.init( x_init );
    for k = 1 to K do
        x_rand ← RANDOM_STATE( );
        x_near ← NEAREST_NEIGHBOR( x_rand, T );
        u ← SELECT_INPUT( x_rand, x_near );
        x_new ← NEW_STATE( x_near, u, Δt );
        T.add_vertex( x_new );
        T.add_edge( x_near, x_new, u );
    endFor
    Return T
```

Algorithm 3 - RRT Algorithm

Algorithm 3 is the original algorithm of RRTs. It is important to note that nowhere from line 1 through 10 it is attempted to reach any specific destination. The algorithm uses the vertex x_{init} as the root of the tree, and spreads from that sample. The tree extension if performed K times with integration of the vehicle's differential equations using a time step of Δt . Each time, a vertex x_{rand} is taken from the world space W using a random variable. This random variable is actually a function, and is chosen to be a uniform distribution of points over W . As the tree T expands, the vertex x_{rand} and all the previous vertices of T are used to form Euclidian distances, and the vertex x of the tree whose distance $d(x,x_{rand})$ is the shortest is designed as x_{near}, the nearest neighbor of x_{rand} based on the currently available vertices of T. In other words, line 4 has the effect of trying to minimize the effort undertaken by the vehicle to move from x_{near} to x_{rand}. Once x_{near} is found, the custom part of the algorithm is to find the input required for the vehicle to move from x_{near} to x_{rand}. This part (line 5) is qualified of custom because it involves finding a set of suitable inputs in order to drive the vehicle from one position to another. Often times a vehicle's dynamic behavior is described by a system of differential equations, so line 6 yields x_{new}, is the result of integrating the vehicle's equation with x_{near} as initial condition over a sample time Δt . When representing RRTs for general purposes, u is often taken as the distance between x_{near} and x_{rand}, so in such cases x_{new} is x_{rand}, which yields RRTs such as the one depicted in Figure 13.

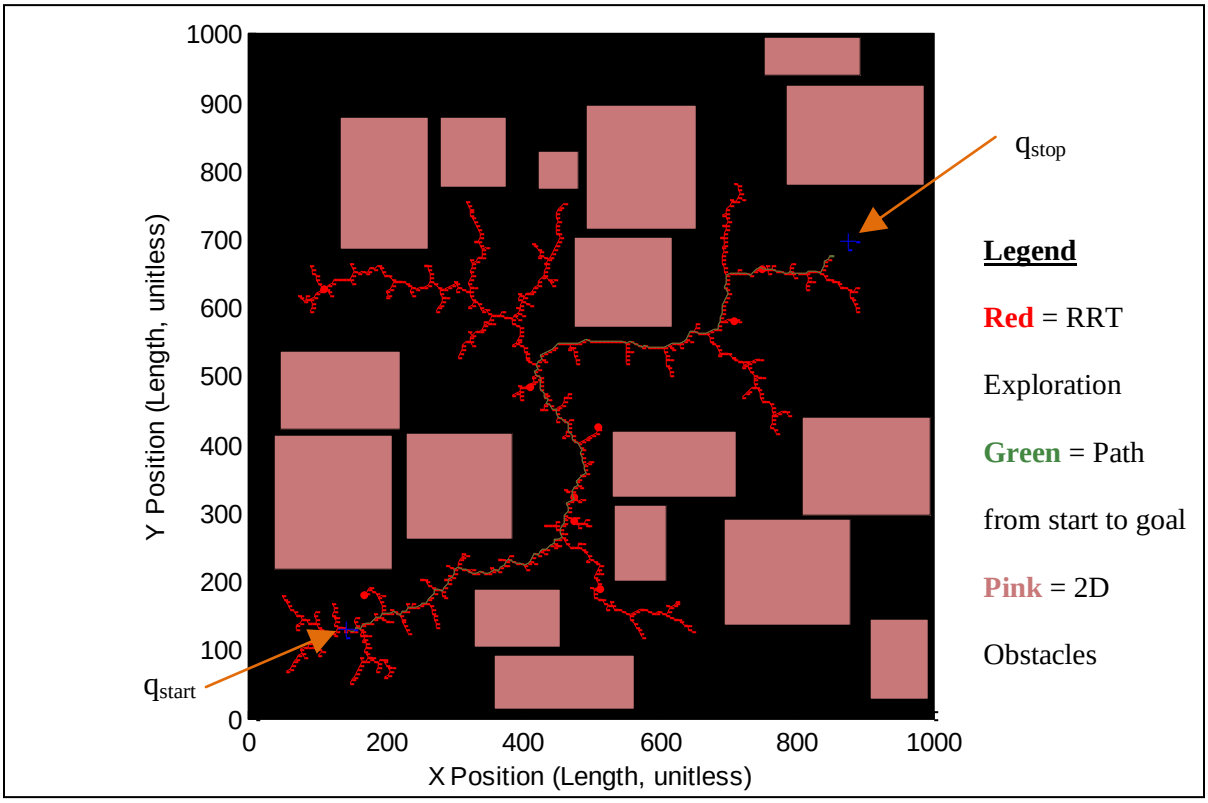


Figure 13 - RRT Spread for U = $||x_{rand} - x_{near}||$

In more realistic cases, when the vehicle is described as a state space with multiple variables, x_{new} and x_{rand} may differ. The resulting x_{new} is then added to the tree T, and if no obstacle is in the vehicle's way between x_{new} and x_{near} , and edge is created between x_{new} and x_{near} , while the corresponding input us which drives x_{near} to x_{new} is recorded as well. This procedure is then repeated k - 1 times.

Practically however, RRTs are used to reach a destination, and Algorithm 3 must be modified in order to account for goal reachability. This means that the algorithm must run until K samples have been drawn OR the goal has been reached (with some tolerance). Also, line 4 alone is the reason why the algorithm is qualified of rapid. At first when the tree does not have many nodes

the spread extends rapidly. But as the number of vertices increases inside the tree T, there are many more candidates to find a minimal distance. RRTs are probabilistically complete: given enough random samples to spread anywhere, they will asymptotically reach a desired area. Given a very high number of random samples to grow to, the tree would at some point have enough readily available vertices so to make the distance between any newly drawn random sample infinitesimally small. Random samples are typically generated by a random variable, a misnamed function in charge of drawing samples. The uniform discrete distribution is often used to generate samples. Uniform distributions have shown in practice good performances in allowing any position of a workspace to be selected with "as many chances as the others".

III - Analysis of Existing Technologies and Identification of a Direction of Research

In the literature review part of this document, six different methods for avoiding obstacles and collision were reviewed. The next table summarizes the principal positive and negative points of each.

Method	Main Attractive Features	Main Drawbacks
Optimization Formulations	- All the effort is put in formulating the optimization problem, not solving it	- slow, even for overly simplified linear problem
Genetic Algorithms	- Relatively fast	- Highly heuristic by nature
Potential Functions	- Discretized Laplacian method provides good results in terms of computation time	- Involves a second gradient descent process - Not necessarily accounting for the space occupied by the vehicle
Geometric Approaches	- Simple concept - Computationally fast (very simple vector calculus)	- Assume constant speed for the UAS and its encounter
Graph Constructions	- Visibility Graph and Trapezoidal Decomposition are simple algorithms and are efficient approaches for 2.5 D problems - Probabilistic Roadmaps are easy to implement	- Do not account for input to the vehicle, so only the configuration space is considered - Requires a path finding algorithm such as A star once the decomposition/mapping is complete
RRTs	- Fast - Rapidly explore previously unexplored areas - account for vehicle input	- Suboptimal - Effort must be directed toward reducing the area of exploration so not to waste computation time

Table 1 - Positive and Negative Points for the Six Methods Reviewed for Collision and Obstacle Avoidance

Out of the six methods reviewed in Table 1, Optimization Formulations are discarded because of being too slow even for over simplified simulations (point-mass vehicles) for the purpose of real-time collision and obstacles avoidance. Genetic Algorithms depend greatly of the goodness of fit, which is expected. However, provisions must be made from a performance perspective, because studied literature does not clearly show a relationship between obstacle map complexity (i.e. the virtual city) and the running time of the algorithms (tournaments and crossing-overs). Potential functions satisfying the numerical version of the Laplace partial differential equation s seem to offer very decent computation times as low as 0.6 seconds for a 128x128x64 grid. For the TCAS II - based approach discussed in Geometric Approaches, the only scenario studied is coplanar collision avoidance with encounter flying at constant speed in a same plane. As airspace is a typically divided into layers, this approach seems reasonable (and is actually used) for general aviation. In the case of non moving obstacles, the calculations to determine the risk of an encounter still hold true. This simply means that all calculations are carried with a zero encounter velocity. The main drawback with this method is the assumption of constant speed for both UAS. Graph approaches are divided into exact 2.5 D techniques and Probabilistic Roadmaps. Visibility graph and trapezoidal decomposition provide a practical approach to the problem of sectional aeronautical map design: importing a general purpose map from a public server such as Google and manually defining coarse no-fly zones that can be translated into convex polygons for further space decomposition is an interesting idea.

Argument is made here that RRTs are the best method to use for real-time obstacle and collision avoidance for fixed-wing UASs modeled as 6 DoF nonlinear models and flying through terrain instantiated as either synthetic or sensor-constructed. The main reason for such a statement lays in the framework of RRTs: the state space domain of admissible solutions (trajectories) for a

given set of inputs. Provided sensible inputs are given to an approximation of a fixed-wing UAS model, it is possible to determine whether a maneuver is possible or not. This means that that an improvement to currently available work in RRTs is to develop a method in order to drive an airplane from one position to another without treating its dynamic behavior as a simple point mass.

III – 1 – RRT Real-Time Space Exploration and Path Finding

As previously explained, a Rapidly-exploring Random Tree is a flexible algorithm which allows states to be selected based on inputs. This allows driving the tree expansion according to specific vehicle's model. However, such flexibility comes with a price: the dimension of the input vector has a direct impact on the RRT workspace coverage. Assuming a vehicle is modeled as a system of equations relating inputs to the state space vector of observed variables, in order for the exploration to be probabilistically complete, the model has to be completely controllable [31]. In the case of linearized aircraft, controllability is usually possible. However, while it is possible for an aircraft to reach a given position using a GNC block, the drawback of such a solution is added complexity. Assuming a real-time exploration of synthetic or real terrain, the exploration must be performed quickly. Based on experience with update rates of commercial autopilots, adding a guidance and control logic to a RRT is not synonymous with significant overhead. On one hand, the role of a RRT is to act as a real – time planner: it is not supposed to replace an autopilot. In fact, a RRT explores the space, which involves a selection of many alternatives routes. It is simply not possible for an autopilot to follow such a behavior, as this would lead to the crash of the aircraft. On the other hand, using one GNC logic block for workspace exploration and another one for actual guidance defeats the original simplicity of the RRT framework.

This is the reason why space exploration must be performed using a RRT-based algorithm and according to some performance parameters, while avoiding a control theory approach.

III – 2 - Assumptions Used When Assessing an Aircraft Performances

As explained in the previous section, while driving a RRT workspace exploration using a GNC logic block is possible, such approach defeats the simplicity of the algorithm. The objective is to bias the workspace exploration as intelligently as possible. This means not wasting computer resources when possible. But biasing a RRT exploration too much is equivalent to breaking the algorithm's initial purpose: being fast and probabilistic-complete. On the other hand, exploring a wide range of possible configurations can involve a great complexity of the guidance logic. Hence, a simpler method of determining whether a point in space is reachable by a UAS while being collision-free is to resort to indicators which give a reasonable estimate of whether that same point is reachable or not. Those indicators are restricted to the coordinated turn and steady climb, and detailed in the following section.

III – 2 – 1 – Assumptions Employed for Lateral Maneuver Decisions

The next figure shows the different forces and angles involved during a coordinated turn. The equations derived from that can be found in Ref. [32], but are reproduced here for reference sake.

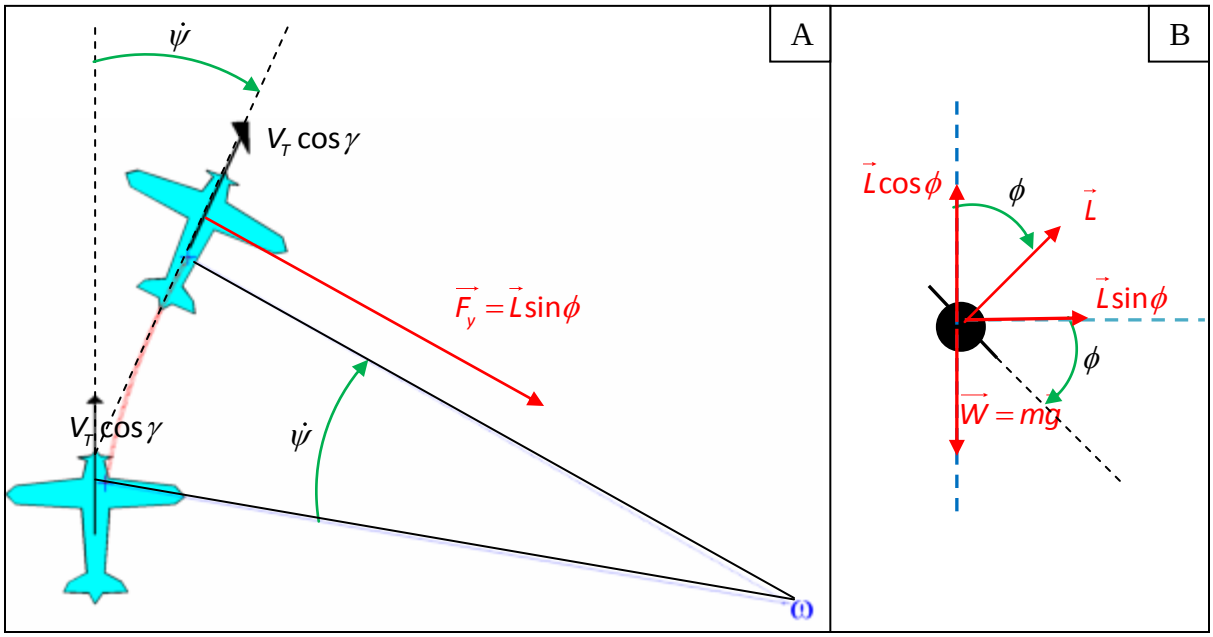


Figure 14 - Derivation of the Turning Radius Formula for a Coordinated Turn

Figure 14 shows a top view (left) and rear view (behind the aircraft, right of the figure) of the angles, velocity, and forces involved during a coordinated turn. In part A of Figure 14, Newton's second law yields $\vec{L}\sin\phi = m\vec{a}_r$, which algebraically translates into

$$L\sin\phi = m\frac{V_T^2\cos^2\gamma}{R_{turn}} \qquad (3.2.1)$$

When considered in the inertial axis, the vertical component of the lift $\vec{L}\cos\phi$ balances the aircraft's weight $\vec{W}$, such that it is possible to write algebraically that

$$L = \frac{mg}{\cos\phi} \qquad (3.2.2)$$

Combining (3.2.1) and (3.2.2) yields the expressing of the turning radius R_{turn} for a coordinated turn. This expression is:

$$R_{turn} = \frac{V_T^2\cos^2\gamma}{g\tan\phi} \qquad (3.2.3)$$

When assuming that the airspeed vector is almost aligned with the horizon (level flight), the $\cos\gamma$ in equation (3.2.3) is very close to the unit, and thus equation (3.2.3) can be rewritten as

$$R_{turn} \simeq \frac{V_T^2}{g\tan\phi} \quad \left(\text{for quasi level flight}\right) \tag{3.2.4}$$

Equation (3.2.4) show that coordinated turns can be considered as circles of constant radius provided that V_T and ϕ are kept constant. In the "results section" of the current chapter, the minimum turning radius is investigated, and because it involves two variables, V_T is kept constant, while the bank angle is taken to its maximum allowable value, in order to manage tight turns.

III – 2 – 2 – Decoupled Longitudinal Maneuvers – The Climb

According to Ref. [33], in order for an aircraft to climb, the airspeed must be held constant while all excess power is used to increase the potential energy. The following figure is a schematic of the different forces expressed in body axes and involved during a climb maneuver.

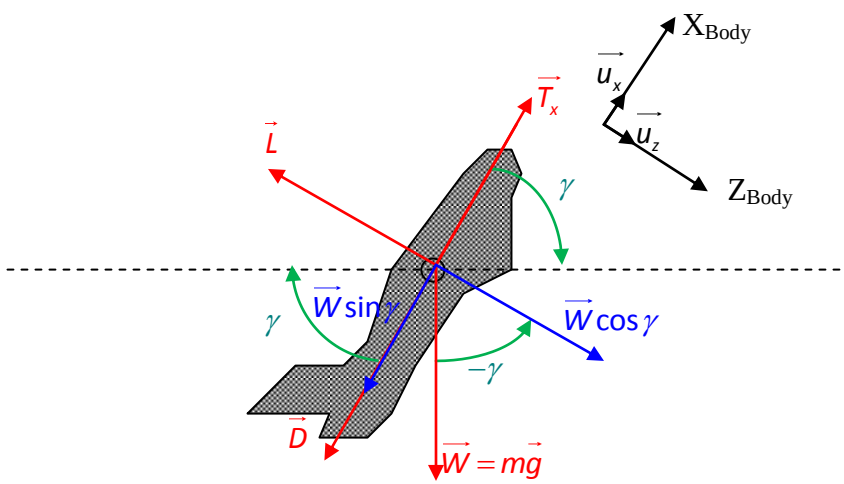


Figure 15 - Diagram of a Climb Maneuver

Assuming a constant airspeed ($dV_T/dt = 0$) and using Figure 15 to apply Newton's second law with respect to the $\vec{u}_x$ unit vector (in body coordinates) yields

$$T_x - W\sin\gamma - D = 0 \tag{3.2.5}$$

Which can be rewritten as

$$\sin\gamma = \frac{(Tx - D)}{W} \tag{3.2.6}$$

Using Figure 15, it can also be observed that

$$\frac{\dot{h}}{V_T} = \sin\gamma \tag{3.2.7}$$

So combining equations (3.2.6) and (3.2.7) gives

$$\dot{h} = V_T \frac{(Tx - D)}{W} \tag{3.2.8}$$

In general, $T_x = T_x(h, V_T)$, $D = D(h, V_T)$, and without specific assumption (e.g. h fixed), equation (3.2.8) cannot be solved analytically, as models of forward thrust and drag models are needed to calculate the rate of climb (ROC). But assuming that those models are available, it is possible to find the forward thrust and drag force for which the quantity $(Tx - D)$ is maximum. The airspeed value corresponding to $(Tx - D)_{\max}$ is $V_T(\gamma_{\max})$.

Hence, for a maximum flight path angle, the decoupled climb maneuver is a straight line, or circle of infinite radius.

Hence, for the lateral maneuvers, circles are interesting smoothing primitives, and for the longitudinal maneuvers, it can be assumed that with knowledge of the flight envelope, maximum climb angle allowing straight ascending lines can be achieved.

The conclusion of Section III - 1 is that RRT are the best family of space exploration algorithm, but need to be further improved in order to account for realistic aircraft maneuvers. The

conclusion of Section III - 2 is that only the coordinated turn for the lateral maneuvers and vertical climb for the longitudinal maneuvers are considered in order to limit the scope of the performance parameters considered. However, because the climb maneuver depends too much of the flight envelop in which the flight mission takes place, it is not explicitly discussed in the rest of the document.

The present section introduced two performance parameters for which an aircraft can be evaluated: the coordinated turn and the climb. The climb maneuver depends greatly on the flight envelope its being performed, and is therefore not discussed any further in the rest of this document. The lateral maneuvers are limited to coordinated turns of minimum turning radius and zero flight path angle.

The next section addresses another topic which is aimed at helping realistic real-time threat avoidance: the use of US General Survey to reduce RRT dispersion.

III - 2 - Aiding the Real-Time Threat Avoidance by Preparing a Synthetic Terrain Avoidance Map

Since 1962 the USGS has been developing Digital Elevation Model (DEM) maps of the world for public use. While the resolution of those maps vary from one area of the world to another, some areas such as Hawaii have been mapped as grids designed with a precision up to 3 arc-second resolution (around 90 meters) and elevation of 30 meters. The following picture is an example of DEM available from the geo COMMUNITY website [34] and viewed using the MICRODEM software available from the United States Naval Academy [35].

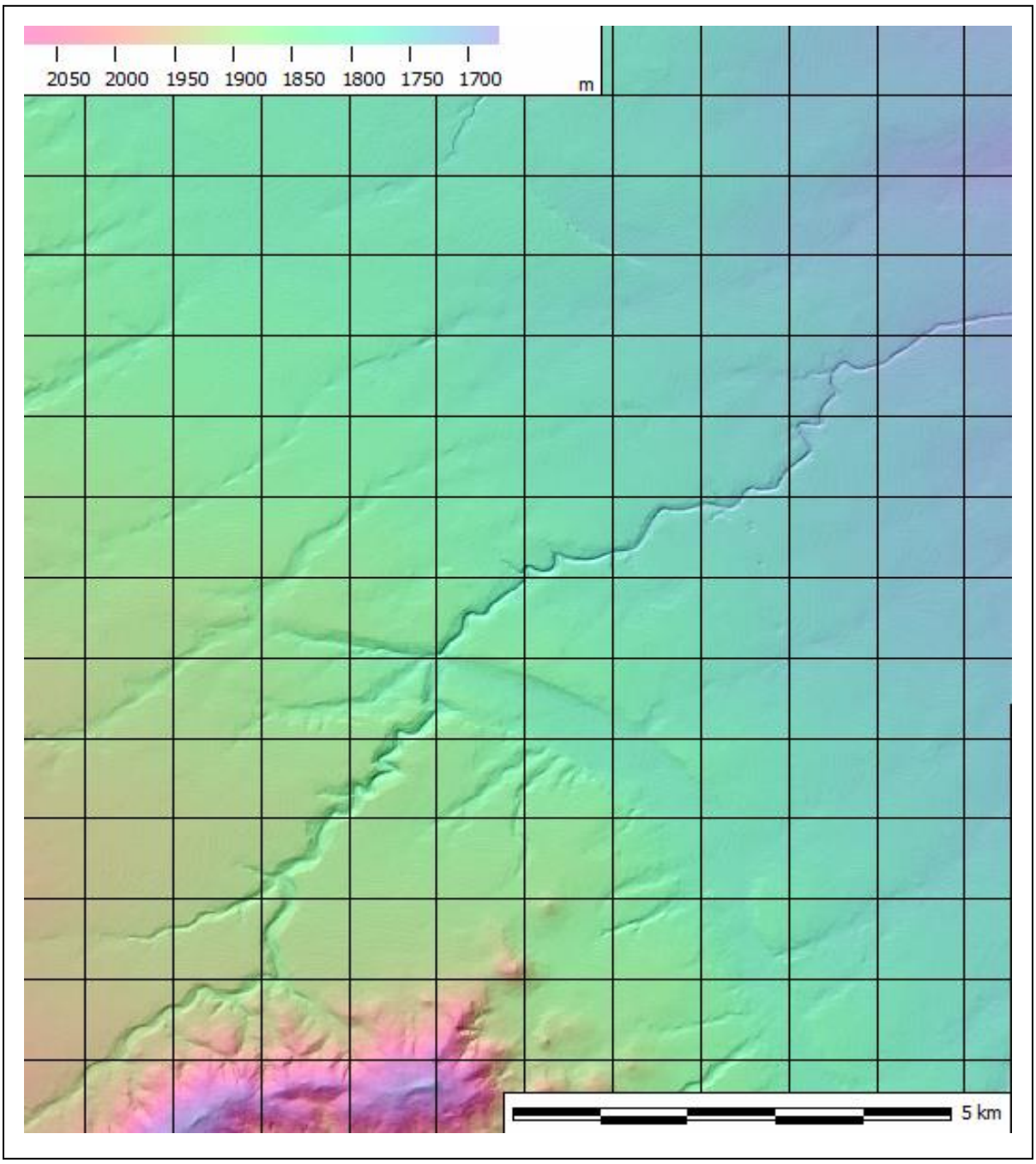


Figure 16 - A 10x10 m DEM Map of the Anderson Canyon, Arizona Made of 1143x1388 Data Points

Unlike standard 2D maps, elevation maps contain necessary data for creating iso-contours. Iso-contours help define flying zones based on elevation. This means that maps such as displayed in Figure 16 can help an end-user define polygonal no-fly zones for missions where altitude is held to a certain value. Those maps can be further enhanced with atmospheric condition information such as wind direction and magnitude. Regions separating no-fly zones can then be connected with a Trapezoidal method.

IV – Preliminary Results

The reviewed literature shows that RRT-based avoidance logics are inefficient since the performed exploration is not targeted. This is normal, as the algorithm is probabilistically complete, and as such does not favor any specific direction during exploration. But it is possible to keep it probabilistically complete while reducing its exploration area.

In this research a new RRT exploration technique is presented which eliminates the exploration of unnecessary areas by performing smart local explorations defined by an offline pseudo 3D coarse search using a trapezoidal decomposition. The reason why the search is qualified of pseudo 3D is because for a given 3D terrain the exploration space is divided and reorganized in a pile of 2D layers. The trapezoidal decomposition decomposes any 2D layer into a obstacle-free Synthetic Terrain Avoidance (STA) network of cells containing connectivity information. One way of applying this 2D technique to a 3D world is by defining an altitude increment that represents perturbed altitudes at any given trim point. The following figure is the result of selecting data points corresponding to a trim altitude of 6595 ft AMSL, and it is assumed that the aircraft may be perturbed between 6130 and 7060 ft AMSL. It is important to mention that these numbers are taken with no base on the 33% scale Yak 54 UAS performance criteria and must be adjusted accordingly in different flight conditions.

Note: The white areas on the map comprise altitudes between 6130 and 7060 feet AMSL. Those areas are thus potential "flying zones" illustrating the background required to proceed to a trapezoidal decomposition of the Grand Canyon map. The red areas are not flyable.

Figure 17 - Two Altitudes Used as Criteria for Elevation Point Selection from the USGS 10mx10m Map of the Grand Canyon

The presented work brings a different perspective to the topic of mid-air reactive fixed-wing UAS threat avoidance, as it treats pre-flight phase and real-time in-flight phases with equal importance.

The pre-flight phase, also referred to as offline phase, uses a dedicated software suite in order to reorganize elevation data and subsequently support the real-time phase. The real-time phase uses the data generated during the offline phase in order to guide a family of rapidly-reactive threat avoidance algorithms.

In part IV - 1, the presented results focus on a method employed to reorganize terrain elevation maps into piles of tessellated 2D layers and create paths between obstacle-free zones Those paths are then transformed into dynamic trajectories using Dubins curves. In part IV - 2, a modified algorithm deriving from the family of Rapidly-exploring Random Trees is grown along the trajectories developed in part IV - 1. This modified version reduces the original RRT dispersion in two ways:

- by focusing the spatial exploration along the dynamic trajectories found in part IV - 2
- For a given trajectory following, limiting RRT growth to the trapezoidal cells being traversed only.

IV - 1 - Offline USGS Map Transformation to Aid In-Flight Threat Avoidance

Non-collaborative threat avoidance targeting small UAS is still a fairly recent research, and as such, cannot afford to shy away from any reliable source of information made available to the public such as USGS elevation maps. But those maps are not readily suited for the task of conducting aircraft safely between two points in the presence of environmental uncertainties like terrains themselves or other aerial intruders.

However, such a limitation can be overcome by writing a computer program in order to re arrange those maps. RRTs allow fast exploration of unknown areas. Therefore, a logical idea is to apply this algorithm to reorganized USGS maps in order for this method to reactively and realistically explore environments using a knowledge of their topology when the latter is made available.

The purpose of the work presented here is to show that clever use can be made of available elevation terrain data in order to improve the real-time reactive exploration of a mission terrain by a modified version of the Rapidly-exploring Random Tree (RRT) algorithm.

By defining flying zones, it is possible to provide a threat avoidance scheme with a "preview" of what the world looks like for a given mission. General aviation can use sectional charts for that, but such a luxury is not available to small UAS which often have to rely on themselves. But USGS maps can provide similar capabilities to UAS albeit through a transformation. Through such transformation, a complex maze of heights becomes a structured layered network of free flying cells, and obstacle-filled non flying zones. Because this network is artificial, and elevation maps are terrain in essence, it bears the name of Synthetic Terrain Avoidance (STA) network.

This transformation has thus the effect of a flight safety - oriented restructuration of USGS elevation maps, which can then be used to improve real-time exploration efficiency during in-flight mission threat avoidance.

STA network must be automated, in order to test various terrain avoidance scenarios. To this end, a software, the Advanced Mapping and Waypoint Generator (AMWG), has been developed. Its only role is to transform elevation data into STA files. But the benefits resulting from STA file productions allow to run simulations which better bridge the gap with reality. An example of that is that STA files intrinsically allow flight altitude scheduling, a generally accepted procedure when designing flight missions. However, because following a constant altitude may not always be possible for aircraft flying in cluttered environment, STA files allow aircraft location on many altitude layers, a feature which is not "out of the box" when using USGS maps directly.

Reorganizing height maps into altitude layers consists for each layer in selecting a lower and uppers bound which defines the membership criterium domain for that layer. The direct consequence of regrouping different altitudes in a same layer is that the different layers are "coarsified" views of the original USGS map.

This coarsification may appear count-intuitive. After all, it results in a level of detail (LOD) loss in USGS original maps. But this LOD loss is necessary when the altitude layers generated serve the purpose of point location queries. Those point location queries must be optimized when requested by real-time randomized algorithms. As those algorithms select a point for exploration, it is more efficient for the algorithm to know about the vicinity of this point, which may be an obstacle - free zone, or a non-flying zone. These zones are extruded tridimensional cells which are extruded from an initial two-dimensional space decomposition. And the simpler the contour of those walls, the easier the point location. This is why the generated STA files involve USGS

map stratification into altitude layers and contour simplification. Converting USGS maps into STAF files involves 5 other steps which are illustrated in the next figure.

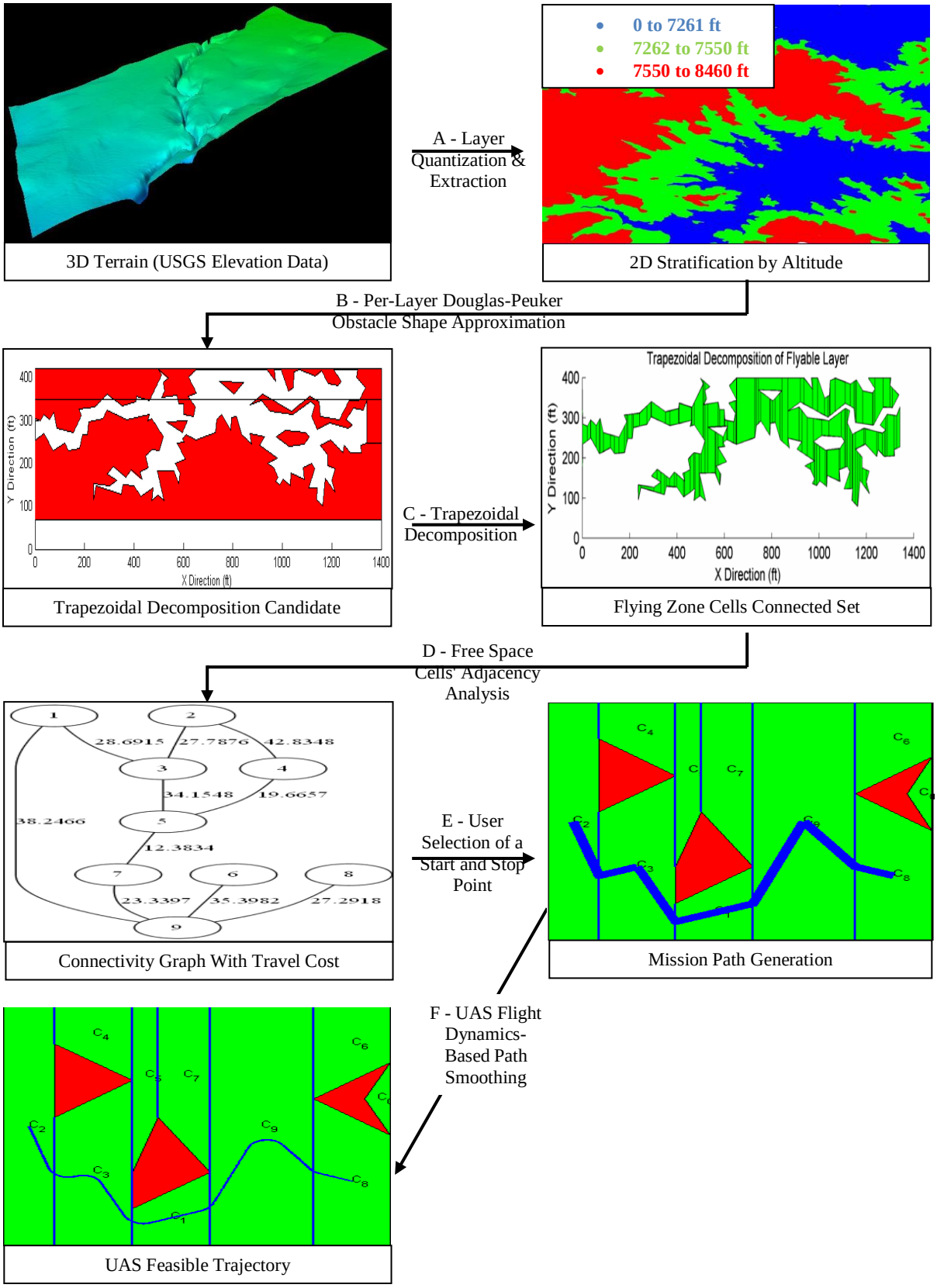


Figure 18 - From USGS File to Working Synthetic Terrain Avoidance Map

In Figure 18, 6 steps are listed from A through F. The first step A is the quantization of an original complex height map into group of altitude layers. The consequence of this is that all altitudes between 0 to 7261 ft are considered to belong to a same group.

After extracting altitude layers from the original height map, step B consists in transforming each altitude layer in a set of simple 2D polygons, but since those polygons have too many vertices for the purpose of space decomposition, their contours are approximated using a Douglas-Peucker algorithm, as described in ref. [36].

At the end of step B, a given layer is separated into free space and obstacles. Because obstacles and free-zones have fewer vertices than in step A, they are good candidates for a trapezoidal decomposition (detailed later in this document), which is the end result of step C.

The trapezoidal decomposition shown at the end of step C works on a vertex basis, by decomposing the space with vertical segments. Because through each vertex of the space passes a line segment, if two line segments are too close, the resulting trapezoidal cell becomes too "thin" and is of little interest for aircraft volume/surface consideration. This is the reason why in step B obstacles and free zones are down-sampled, as this allows the cells resulting from trapezoidal decomposition to maintain a certain space between their left and right edges.

Because each cell of the decomposition is a trapeze (degenerated case such as triangles and lines are discarded), its centroid can be easily calculated. Each centroid of the trapezoidal decomposition is then used to calculate the distance between two adjacent cells. Because those cells are free zones with adjacency information, they can subsequently be used to build an adjacency undirected graph, whose weights are the distance between cells. It is such a graph which is shown as the end result of step D. This adjacency graph is the STA network mentioned earlier.

Depending on the mission profile, different pair of departure/arrival cells are selected, but it is the STA network generated during step D which allows feedback regarding the existence of a path between the two cells. Using the centroids calculated in step D and the adjacency information regarding the common line segment shared by two adjacent cells, step E generates a geometric route for the two user-selected end points. But as this geometric route is not necessarily suited for an aircraft dynamics it is smoothed into a dynamic trajectory during step F.

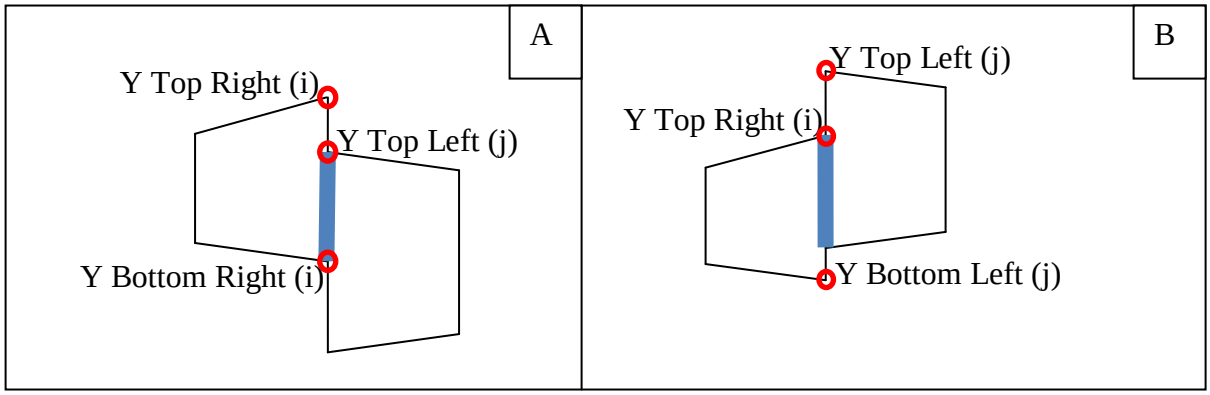
IV - 1 - 1 - The Trapezoidal Decomposition Algorithm Employed by the AMWG

The AMWG follows 6 distinct steps in order to transform an elevation map into a STA file. The 3rd step of this transformation involves a trapezoidal decomposition, whose formal description can be found in ref. [22]. However, because the trapezoidal decomposition used in the AMWG software is used for point location and obstacle removal, a brief description follows.

The trapezoidal decomposition is a method used to decompose a plane into vertical bands. If the initial space contains non-vertical line segments, the vertical bands become trapezoids, hence the terminology. For any vertex part of the space, a vertical segment is drawn through. As the final decomposition treats obstacles and free zones indistinctly, additional logic is needed to remove the obstacles from the trapezoidal decomposition.

Before decomposition, all obstacle contour are known via step A of Figure 18. After decomposition, those obstacles are sliced into trapezes, of which centroids can be calculated. Then, for each centroid, the discretized Jordan curve theorem is applied (Ref. [37]). For each centroid, a line segment is drawn until it hits the plane's boundaries. Then, for each known obstacle contours, intersections with those contours are counted. If the number is odd, it means the centroid belongs to an obstacle, so all the trapezoidal cells whose centroid result in odd tests for the known contours are removed from the decomposition.

After the decomposition is performed, free spaces cells accommodating a given safety radius are the end result of the decomposition, but no connectivity knowledge is established yet at that point. So the next post-decomposition step is to determine for each cell its adjacency with respect to up to four other cells. The algorithm to determine this is given by the next figure.



Abbreviations Employed

Y Top Right	≡	*YTR*
Y Bottom Right	≡	*YBR*
Y Top Left	≡	*YTL*
Y Bottom Left	≡	*YBL*

```
true/false = AreCellsAdjacent( cell i, cell j )
        if(YTL_j>YBR_i and YTL_j ≤ YTR_i)      // part A of figure
                return true
        if( YTR_i > YBL_j and YTR_i ≤ YTL_j) // part B of figure
                return true
        else
                return false
end
```

Figure 19 - Trapezoidal Cell Adjacency Computation

The adjacency computation algorithm is presented in Figure 19. For two cells i and j such that cell i is different from cell j, the algorithm tests if the right end points of cell i and the left end points of cell j allow to for form the non zero-length line segment shown in blue in Figure 19. Because the length of this line segment depends on the y coordinates only, a simple only one subtraction per adjacency calculation is needed.

The STA network is an acyclic graph relating extruded two-dimensional trapezoidal cells with one another provided that each of those have at least one adjacency property defined with another cell, and that each defined adjacency line segment is not reduced to a point.

Because the STA network is an acyclic graph, two different cells can be connected following different cell arrangements, and without further analysis, nothing can help determine which path is the best. In the event of the case where multiple paths can be used to connect a departure to an arrival cell, all the possible paths must be all listed. As mentioned in Ref. [38], the National

Institute of Standards and Technology (NIST) recommends using a depth first search (DFS) algorithm to find all simple paths between two vertices of a network. In order to illustrate the advantages of using DFS for path listing the following figures are used.

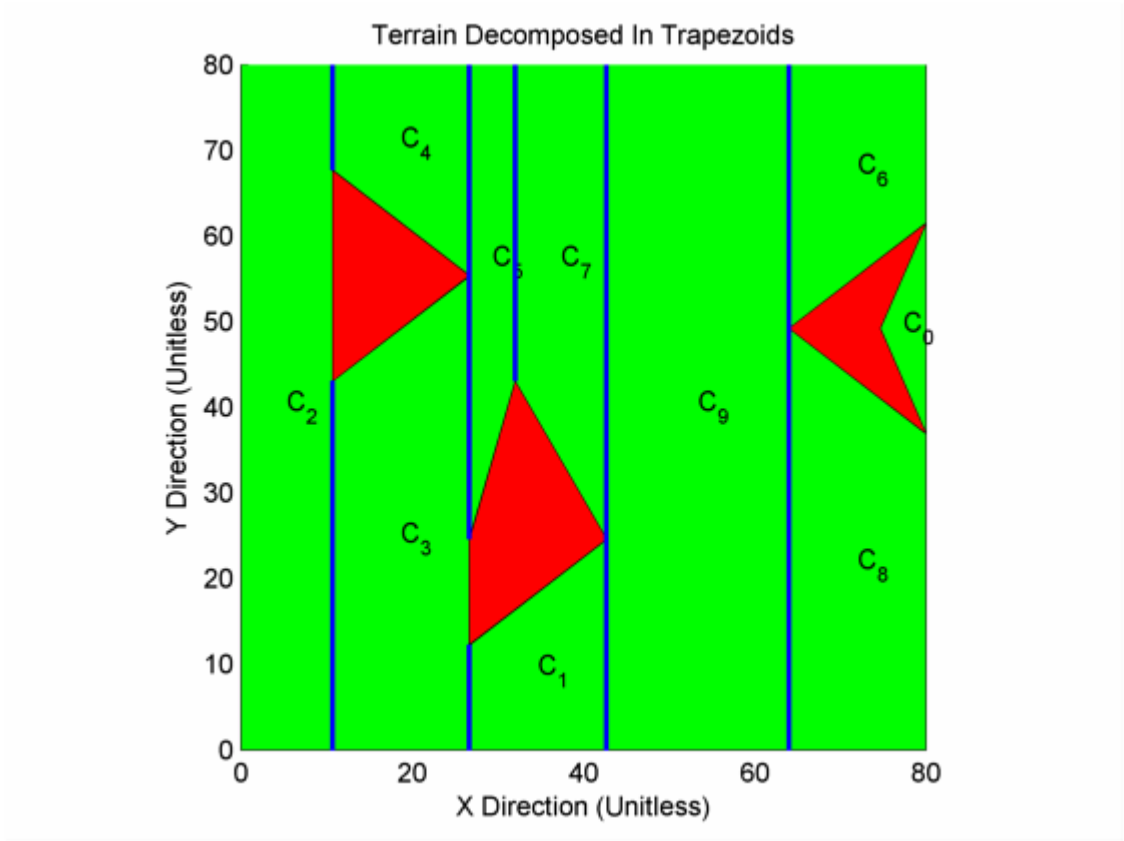


Figure 20 - Trapezoidal Decomposition Applied to a 2D Map

Figure 20 shows 10 cells numbered from 0 to 10 and resulting from a vertical trapezoidal decomposition. Using a distance metric detailed next, it is possible to establish a distance between the centroids of two connected cells (connections marked in blue). If such a distance metric is used as weight between two nodes, the resulting STA network can be observed in the next figure.

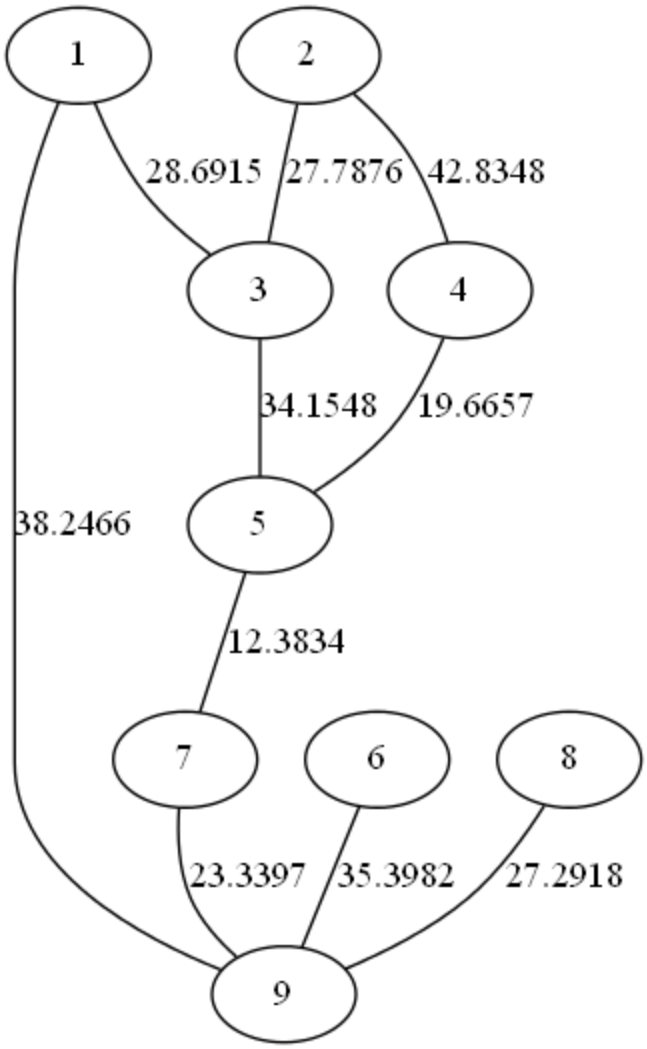


Figure 21 - Undirected Connectivity Graph Generated from Trapezoidal Decomposition Connectivity Property

Figure 24 is the visual representation the STA network. When applying the Depth First Search (DFS) algorithm to all pairs of node in the graph, it is possible to list all the paths between all pair of connected points. This fact is illustrated is illustrated next, with the application of the DFS algorithm to the STA network of Figure 21 producing the following listing for all the paths between all the cells.

path 1 to 2
1 3 2
1 3 5 4 2
1 9 7 5 3 2
1 9 7 5 4 2

path 1 to 3
1 3
1 9 7 5 3
1 9 7 5 4 2 3

path 1 to 4
1 3 2 4
1 3 5 4
1 9 7 5 4
1 9 7 5 3 2 4

path 1 to 5
1 3 5
1 3 2 4 5
1 9 7 5

path 1 to 6
1 3 2 4 5 7 9 6
1 3 5 7 9 6
1 9 6

path 1 to 7
1 3 2 4 5 7
1 3 5 7
1 9 7

path 1 to 8
1 3 2 4 5 7 9 8
1 3 5 7 9 8
1 9 8

path 1 to 9
1 9
1 3 2 4 5 7 9
1 3 5 7 9

path 2 to 3
2 3
2 4 5 3
2 4 5 7 9 1 3

path 2 to 4
2 4
2 3 1 9 7 5 4
2 3 5 4

path 2 to 5
2 3 5
2 3 1 9 7 5
2 4 5

path 2 to 6
2 3 1 9 6
2 3 5 7 9 6
2 4 5 3 1 9 6
2 4 5 7 9 6

path 2 to 7
2 3 1 9 7
2 3 5 7
2 4 5 7
2 4 5 3 1 9 7

path 2 to 8
2 3 1 9 8
2 3 5 7 9 8
2 4 5 3 1 9 8
2 4 5 7 9 8

path 2 to 9
2 3 1 9
2 3 5 7 9
2 4 5 3 1 9
2 4 5 7 9

path 3 to 4
3 1 9 7 5 4
3 2 4
3 5 4

path 3 to 5
3 5
3 1 9 7 5
3 2 4 5

path 3 to 6
3 1 9 6
3 2 4 5 7 9 6
3 5 7 9 6

path 3 to 7
3 1 9 7
3 2 4 5 7
3 5 7

path 3 to 8
3 1 9 8
3 2 4 5 7 9 8
3 5 7 9 8

path 3 to 9
3 1 9
3 2 4 5 7 9
3 5 7 9

path 4 to 5
4 5
4 2 3 5
4 2 3 1 9 7 5

path 4 to 6
4 2 3 1 9 6
4 2 3 5 7 9 6
4 5 3 1 9 6
4 5 7 9 6

path 4 to 7
4 2 3 1 9 7
4 2 3 5 7
4 5 7
4 5 3 1 9 7

path 4 to 8
4 2 3 1 9 8
4 2 3 5 7 9 8
4 5 3 1 9 8
4 5 7 9 8

path 4 to 9
4 2 3 1 9
4 2 3 5 7 9
4 5 3 1 9
4 5 7 9

path 5 to 6
5 3 1 9 6
5 4 2 3 1 9 6
5 7 9 6

path 5 to 7
5 7
5 3 1 9 7
5 4 2 3 1 9 7

path 5 to 8
5 3 1 9 8
5 4 2 3 1 9 8
5 7 9 8

path 5 to 9
5 3 1 9
5 4 2 3 1 9
5 7 9

```
path 6 to 7
6 9 7
6 9 1 3 2 4 5 7
6 9 1 3 5 7

path 6 to 8
6 9 8

path 6 to 9
6 9

path 7 to 8
7 5 3 1 9 8
7 5 4 2 3 1 9 8
7 9 8

path 7 to 9
7 9
7 5 3 1 9
7 5 4 2 3 1 9

path 8 to 9
8 9
```

As shown with the previous listing, the DFS algorithm applied to the STA network of Figure 21 allows to list all the paths between two cells. This is of extreme importance when a chosen path become invalid due to an unexpected threat, because a listing like the above allows to select alternatives routes when possible.

A result of applying the DFS to the STA layer shown in parts E and F of Figure 18 is given next.

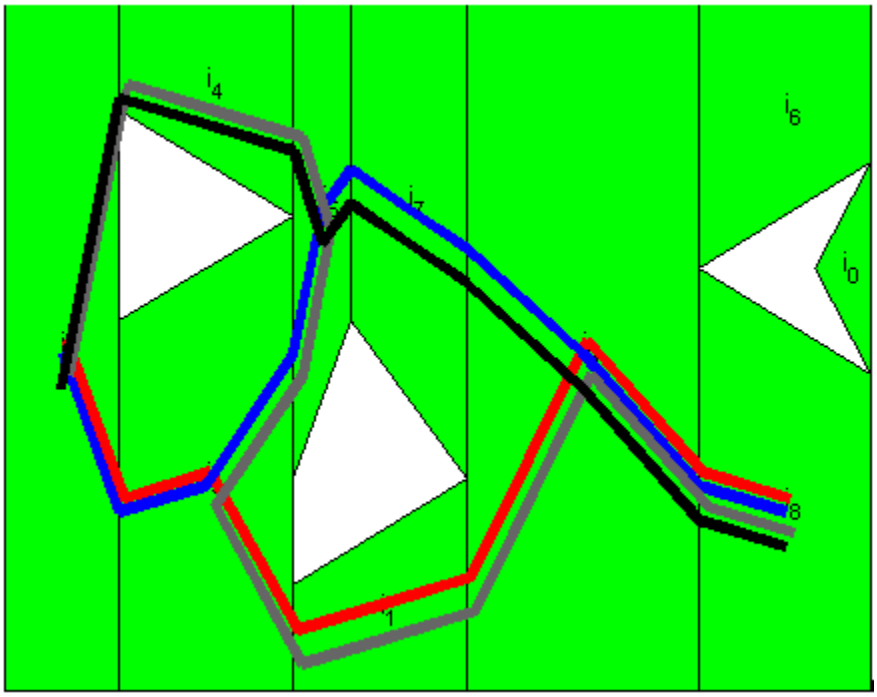
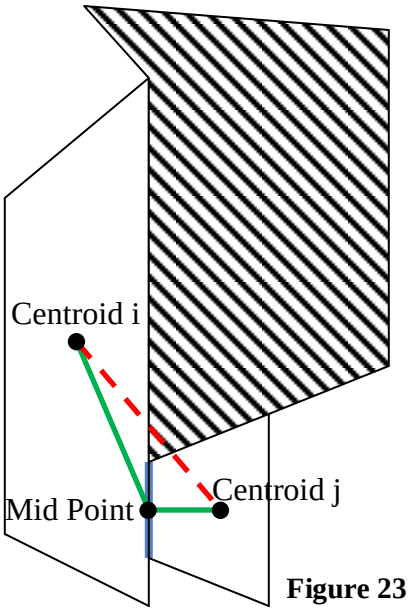


Figure 22 - Four Different Paths to Connect Two Points

Figure 22 is the result of applying the recommendations of Ref. [38] to the STA network. The result is four paths all of which connect cell 2 to cell 8. Some of those paths have overlapping portions, but were drawn slightly shifted one from another in order to clearly distinguish them. Those paths are by no mean optimal in term of length. This is particularly obvious in the case of the red path, where the last "peak" is completely unnecessary. These paths are not minimal length paths because they are generated according to the method described in Figure 19. Because two consecutive cells are not guaranteed to form a convex set, connection between two cells also use the mid-point of the blue line segments depicted in Figure 19. This is better explained in Figure 23.



Two cells with respective centroids C_i and C_j. Connecting the centroids with a direct line segment results in the red segment intersecting one of the 5 edges of the textured 2D polygon. But connecting the two centroids through the mid-point of the common adjacency line segment prevents such collision.

Figure 23 - Computation of the Path Between Two Cells

As previously mentioned, the method detailed in Figure 23 makes possible to connect cells without going through obstacles, but at the same time the resulting paths are suboptimal.

A suggested direction for path simplification is as follows. Cells centroids are not always necessary for path construction. But they are needed when a succession of two flying legs involve a change of direction along a given axis (either x or y). Therefore, given two consecutive flying legs and three points q_1, q_2, and q_3, with x coordinates x_1, x_2, and x_3, y coordinates y_1, y_2, and y_3, if q_2 is a cell's centroid and q_1, q_3 two blue segment's mid-points, it is kept from the initial path only if it contributes to a change in the x direction. In a more formal fashion, the path simplification just described is summarized in the next algorithm.

```
simplified_path = path_simplification( waypointList, numberOfWaypoints )
L01    keptWaypointList( 1 ) = waypointList( 1 )
L02    for( k = 2, until k reaches (numberOfWaypoints-1) by increments of 1)
L03            x1 = waypointList(k- 1,x)
L04            x2 = waypointList(k+0,x)
L05            x3 = waypointList(k+1,x)
L06            if{[(x1 ≤ x2) ∧ (x3 ≤ x2)] ∨ [(x1 ≥ x2) ∧ (x3 ≥ x2)]} // sign change for x?
L07                    keptWaypointList= keptWaypointList ∪ waypointList(k)
L08                    continue for loop
L09            endIf
L10            if( mod(k,2) is equal to 0 ) // is the kth point a mid-point?
L11                    keptWaypointList= keptWaypointList ∪ waypointList(k)
L12                    continue for loop
L13            endIf
L14    endFor
L15    keptWaypointList= keptWaypointList ∪ waypointList(numberOfWaypoints)
```

Algorithm 4 – The Path Simplification Algorithm

Perhaps the most salient point of Algorithm 4 is that it only considers variations along the x direction to decide whether a midpoint must be eliminated or not for path simplification. Because the trapezoidal decomposition employed in this document is a vertical trapezoidal decomposition, any connection between two cells involves a horizontal displacement. A consequence of this is that a STA network without obstacle only allows horizontal displacement. A corollary of this statement is that a two-dimensional layer must contain obstacles in order to allow vertical cell connections if those cells have been generated through a vertical trapezoidal decomposition. So for a vertical trapezoidal decomposition, any type of cell connection (vertical or horizontal) involves a horizontal connection. This is why Algorithm 4 only involves horizontal considerations regarding the logic employed to discard centroids. If instead of vertical, the decomposition is horizontal, then Algorithm 4 would be modified to only involve vertical considerations (variations along the y axis).

When Algorithm 4 is applied to the four paths shown in Figure 22, the result is the simplified paths shown in the next figure.

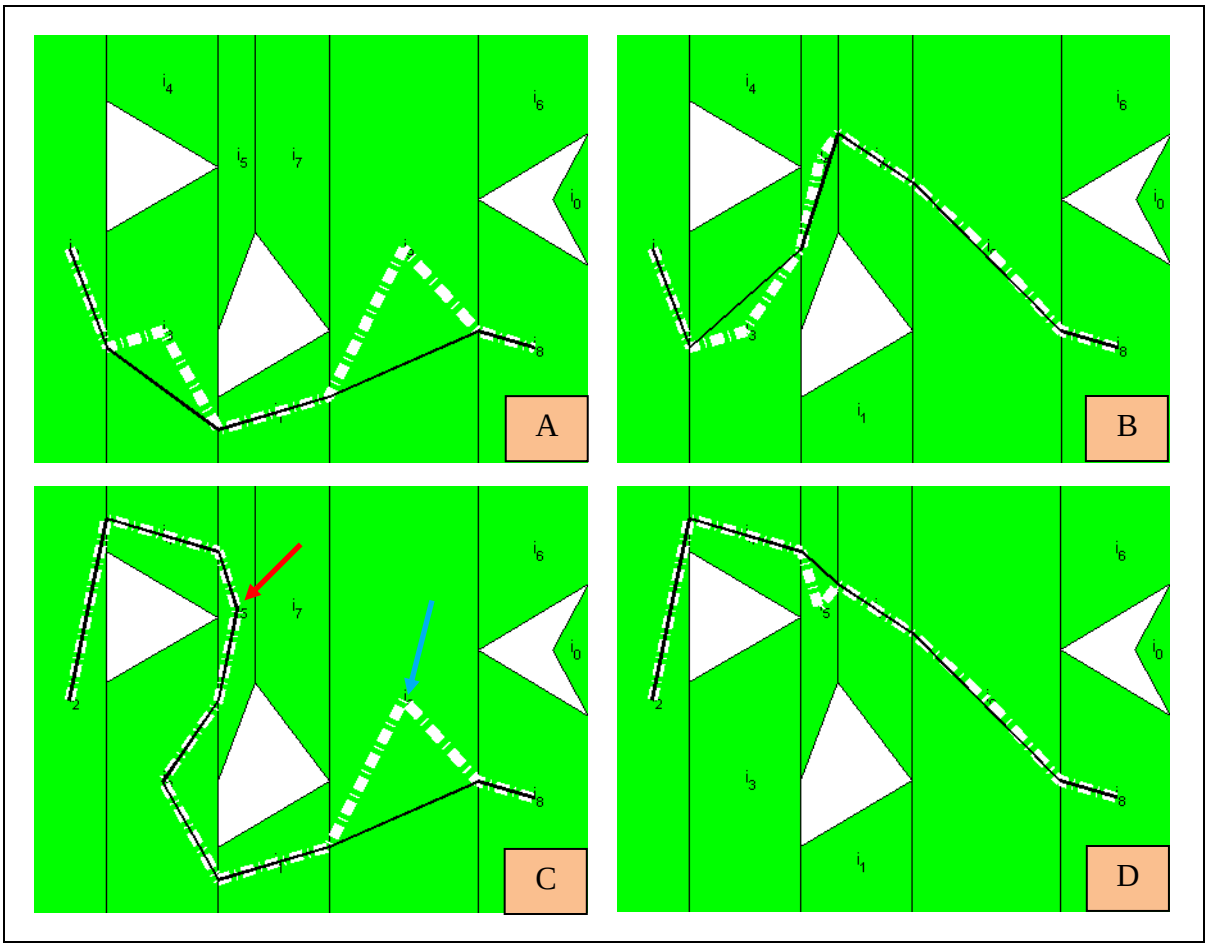


Figure 24 - Four Paths in White and Their Simplification in Black

Figure 24 shows the four original paths of Figure 22 in thick alternated white line, and the resulting simplifications in black lines. The simplification does not bring the black flying legs closer to the obstacles than the original thick white flying legs already are. This is because Algorithm 4 only operates on potential trapezoidal cells' centroids' removal. Using the logic described in Algorithm 4, centroid removal cause an increase in obstacle collision risk with the vehicle if and only if the flying legs surrounding the given centroid result in a change of variation along the x or y axis. This fact is illustrated by part C of Figure 22 when considering the first four green cells from starting from the left of Figure 22. If the centroid marked by a red arrow is removed in part C, then the resulting path enters in direct collision with the left most triangular obstacle. However, when two flying legs surrounding a centroid do not involve a change in the sense of variation along the x or y direction, the centroid can be safely removed, as shown by the blue arrow.

Although the logic presented in Algorithm 4 is at best able to shorten paths, it does not make those paths any safer. This can again be observed in part C of Figure 22 with the left most back flying leg. Because that leg is supported by a straight line, its right most end is dangerously close to the left most triangular obstacle. The end points are not what causes the leg to be so close to the obstacle. It is the fact that the two end points are connected by a straight line which is what causes the danger.

At the moment where this document is being written, experimental methods aiming at improving those paths safety are investigated, but cannot be presented here

Because Figure 24 shows that there are four different paths connecting two points, it is desirable to be able to rank those paths. The ranking can be a linear combination of costs. A first cost is the trajectory length, of which an approximation can be obtained with the path length.

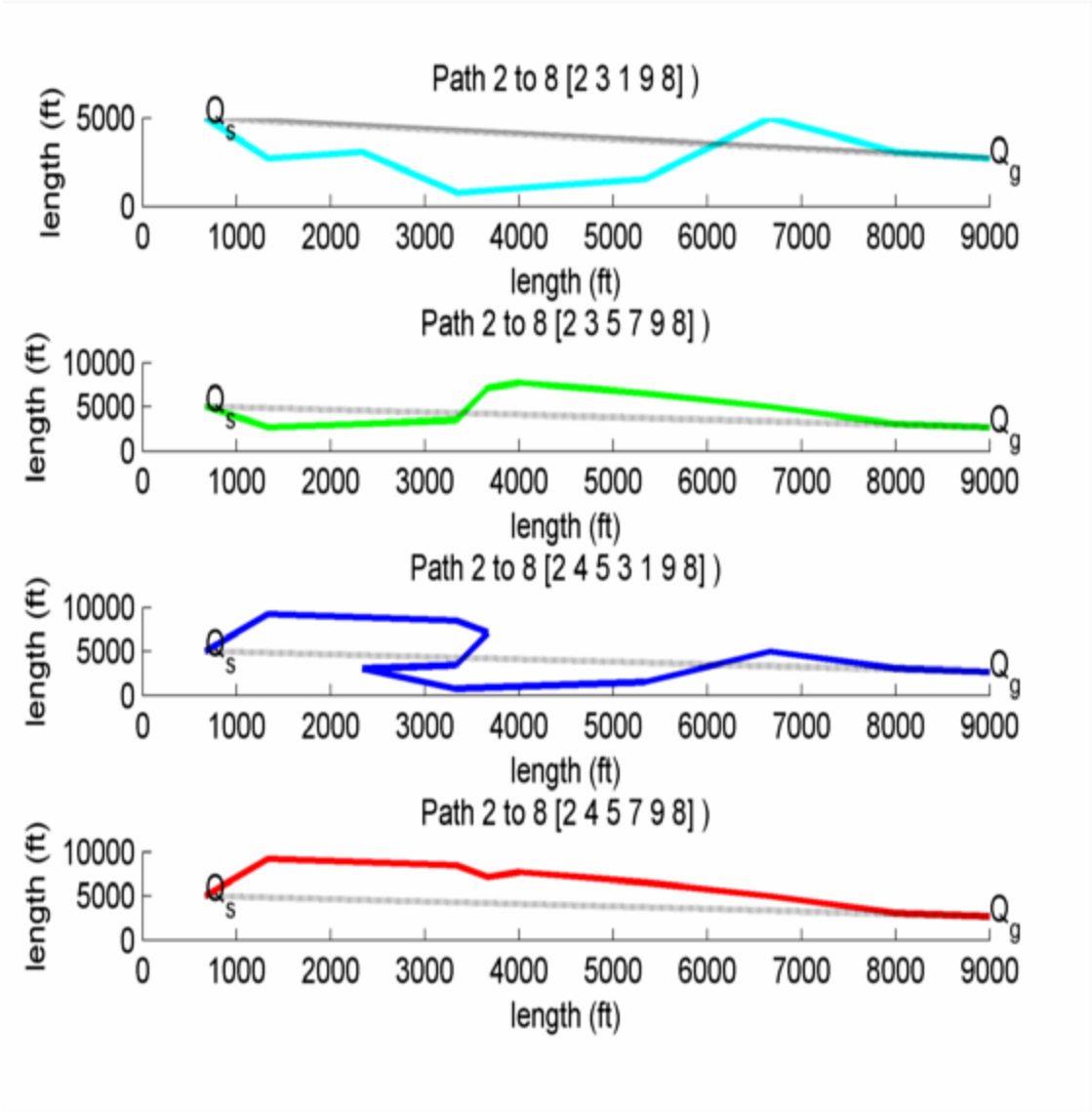


Figure 25 - Path Ranking for Different Criteria

Figure 25 shows the profiles of the four different paths linking cell 2 to cell 8. If the measured criterium is the Euclidian distance between the paths' waypoints, those distances are respectfully (top being first, bottom being fourth): (1) 15252.19 (2) 16090.71, (3) 24331.70, (4) 15689.41. This means that ranking those paths by their increasing length, the rank order is: 1 4 2 and 3. So if the Euclidian distance is the criterium to chose a "best route" (shortest path in this context), then the top path is the best. This shows that the STA network is a very valuable tool because the

relationship between its different cells can be reused for different purposes. The role of Figure 25 is to show that there is no absolute "best path" per se, but a best path for a certain goal which can depend of the objective. Indeed, a shortest path is not necessarily the safest.

Because the matter treated here is the threat avoidance it should be the most important criterium when selecting a "best path". This is however a much more difficult task, but a methodology is suggested here. For each of the paths connecting a departure and rendez-vous cells, the paths are followed by the aircraft safety circle. For a given path, if collisions occur between the safety circle and the obstacles, the circle's area of impact is summed. The path with the smaller sum is the best path. This implementation of this methodology is still in development as this document is being released.

IV - 2 - Dubins Curves for Lateral Maneuvers Following Waypoints

Dubins curves as described in ref. [39] and [40] are curves of minimal length used to join two postures. Each posture is composed of one position and one orientation. In the case of lateral maneuvers, those curves can be parametrized in terms of the aircraft's airspeed and bank angle. If an aircraft is kept at constant airspeed and bank angle, its turning radius never varies.

The next figure shows the four different Dubins curve configurations involved in an aircraft lateral maneuver. Despite the assumption of a constant and minimal turning radius for tight coordinated turns, those configurations are shown with circles of different radii. This is for the sake of generality.

Also, for the record, the following details regarding the Dubins curve generation are the author's knowledge the first document detailing the curve generation for circles of different radii.

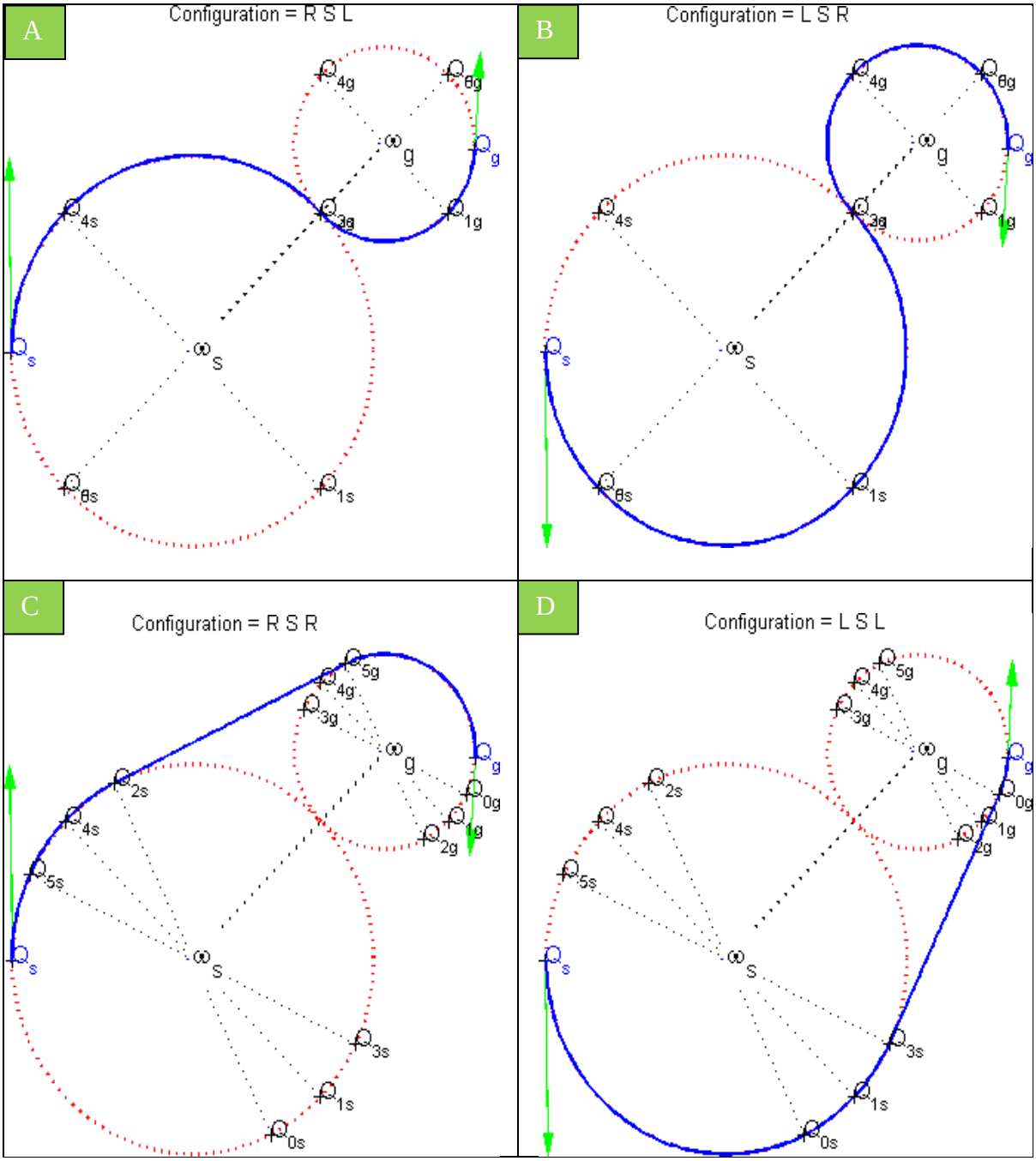


Figure 26 - Four Different Dubins Curves

IV - 2 - 2 - Dubins Curves' Equations

Figure 26 shows the four possible configurations for Dubins curves. Parts A and B are respectively Right Straight Left (RSL) and Left Straight Right (LSR). Because for each of those two curves, the first and last turns are opposite turns, those configurations are referred to as "Belt Problem" configurations. For those configurations, the angle $\left(\overrightarrow{\omega_s\omega_g}, \overrightarrow{\omega_s Q_{2s}}\right)$ is given by

$$\varphi = \left(\overrightarrow{\omega_s\omega_g}, \overrightarrow{\omega_s Q_{2s}}\right) = \cos^{-1}\left(\frac{r1 + r2}{\left\|\overrightarrow{\omega_s, \omega_g}\right\|}\right) \qquad (4.2.1)$$

Parts C and D are respectively Right Straight Right (RSR) and Left Straight Left (LSR). Because for each of those two curves, the first and last turns have same orientation, those configurations are referred to as "Pulley Problem" configurations. For those configurations, the angle $\varphi = \left(\overrightarrow{\omega_s\omega_g}, \overrightarrow{\omega_s Q_{2s}}\right)$ is given by

$$\varphi = \left(\overrightarrow{\omega_s\omega_g}, \overrightarrow{\omega_s Q_{2s}}\right) = \cos^{-1}\left(\frac{r_1 - r_2}{\left\|\overrightarrow{\omega_s, \omega_g}\right\|}\right) \qquad (4.2.2)$$

For the locations of Q_{0s}, Q_{1s}, Q_{2s}, Q_{3s}, Q_{4s}, Q_{5s} and Q_{0g}, Q_{1g}, Q_{2g}, Q_{3g}, Q_{4g}, Q_{5g} the vectors $\overrightarrow{v_s} = \frac{\overrightarrow{\omega_s, \omega_g}}{\left\|\overrightarrow{\omega_s, \omega_g}\right\|} r_1$ and $\overrightarrow{v_g} = \frac{\overrightarrow{\omega_g, \omega_s}}{\left\|\overrightarrow{\omega_g, \omega_s}\right\|} r_2$ are defined, along with the points $P_s = \omega_s + \overrightarrow{v_s}$, $P_g = \omega_g + \overrightarrow{v_g}$.

The "s" points are given as rotations of P_s relative to $\overrightarrow{v_s}$ about the center ω_s.

The "g" points are given as rotations of P_g relative to $\overrightarrow{v_g}$ about the center ω_g.

The relative angles of the rotations for all the points are given by.

$$
\begin{array}{lclclcl}
Q_{0s} & = & (|\varphi| - \pi) & \qquad & Q_{0g} & = & -(|\varphi| - \pi) \\
Q_{1s} & = & (-\pi/2) & & Q_{1g} & = & -(-\pi/2) \\
Q_{3s} & = & (-|\varphi|) & & Q_{2g} & = & -(-|\varphi|) \\
Q_{2s} & = & (+|\varphi|) & & Q_{3g} & = & -(+|\varphi|) \\
Q_{4s} & = & (+\pi/2) & & Q_{4g} & = & -(+\pi/2) \\
Q_{5s} & = & (-\pi - |\varphi|) & & Q_{5g} & = & -(-\pi - |\varphi|)
\end{array}
\qquad (4.2.3)
$$

For the set of angles in equation (4.2.3) is that if r_1=r_2, some of the points overlap. For example when r_1=r_2, Q_{5s} = Q_{4s} and Q_{0s} = Q_{1s}, but practically, the fact that generally distinct points overlap in special cases has no impact logic-wise for the Dubins curve generation.

The next table gives all the point configurations for the different cases.

	RSL	LSR	RSR	LL
$r_1 < r_2$	(Q_sQ_{2s}) [$Q_{2s}Q_{2g}$] ($Q_{2g}Q_g$)	(Q_sQ_{3s}) [$Q_{3s}Q_{3g}$] ($Q_{3g}Q_g$)	(Q_sQ_{5s})[$Q_{5s}Q_{3g}$]($Q_{3g}Q_g$)	(Q_sQ_{0s})[$Q_{0s}Q_{2g}$]($Q_{2g}Q_g$)
$r_1 > r_2$			(Q_sQ_{2s})[$Q_{2s}Q_{5g}$]($Q_{5g}Q_g$)	(Q_sQ_{3s})[$Q_{3s}Q_{0g}$]($Q_{0g}Q_g$)
$r_1 = r_2$			(Q_sQ_{4s})[$Q_{4s}Q_{4g}$]($Q_{4g}Q_g$)	(Q_sQ_{1s})[$Q_{1s}Q_{1g}$]($Q_{1g}Q_g$)

Table 2 - Details of Point Sequences for the Different Dubins Configurations

The logic used to decide which of the four lateral maneuvers to select depends of a succession of at least three postures, or two flying legs. But this logic can only be applied if a certain condition on the minimal flying leg is met. For two successive waypoints Q_s and Q_g, the distance $\left\|\overrightarrow{\omega_g, \omega_s}\right\|$ must be such that

$$\left\|\overrightarrow{\omega_g, \omega_s}\right\| \geq (r_1 + r_2) \qquad (4.2.4)$$

Besides the Dubins path existence condition expressed by equation (4.2.4), an algorithm responsible of the maneuvers required to connect two postures is needed. This algorithm is presented in the next figure.

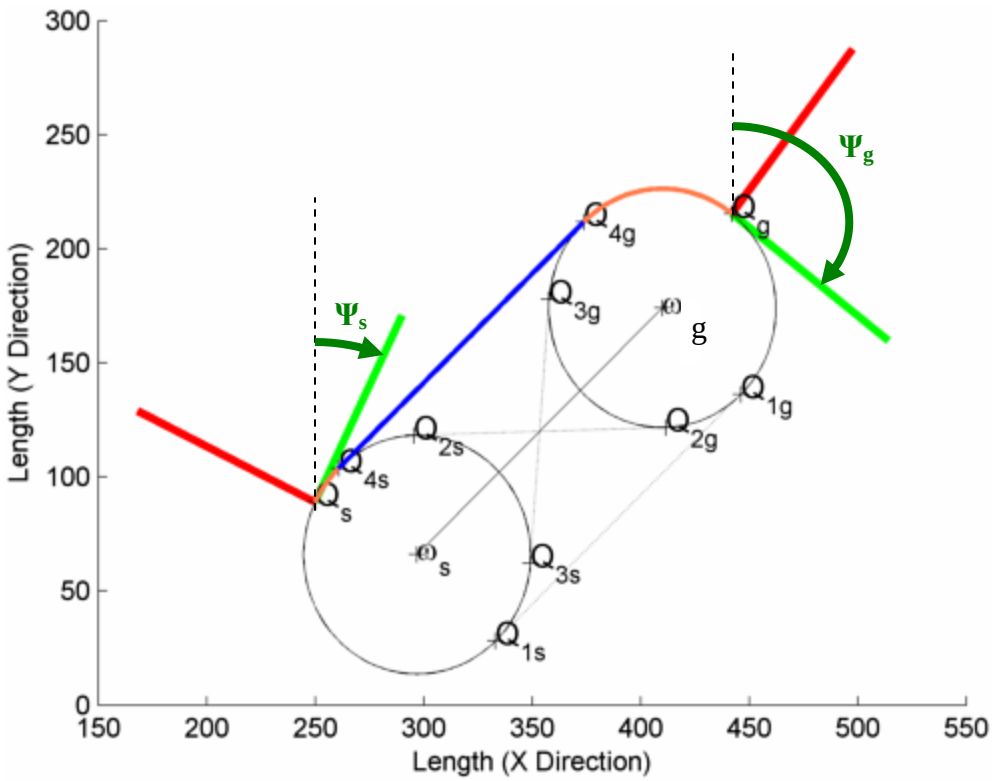


Figure 27 - Example of a Dubins Curve Joining Two Points Q_s and Q_g By Following a Right Turn, Straight Line, Right Turn

Figure 27 shows an example where two poses (Q_s,Ψ_s) and (Q_g,Ψ_g),are connected with a right arc circle, straight line, right arc circle (RLR). The bright green line segments show the velocity vectors at the two positions. The red line segments are the vectors perpendicular to the velocity vectors. Once normalized, they are used to find the centers Ω_s and Ω_g of the two circles. In Figure 27, (Q_s,Ψ_s) and (Q_g,Ψ_g) are connected using a right turn, straight line, and finally a right turn. By defining a coordinate frame $\mathscr{F}$(X_s,Y_s,V_s)whose origin is Q_s, the Y coordinate of Q_g is negative with respect to $\mathscr{F}$(X_s,Y_s,V_s), so in a first instance, a right turn must be performed. Likewise, by defining a coordinate frame $\mathscr{F}$(X_g,Y_g,V_g)whose origin is Q_g, the Y coordinate of Q_s

is negative with respect to $\mathscr{F}(X_g,Y_g,V_g)$, so in a first instance, a right turn must be performed. This explains why in the example of Figure 27, the maneuver connecting the two points is RLR. While observing Figure 27, it can be observed that the Dubins curve connecting the points s and g does not follow the line segment $[Q_s, Q_g]$. This is however not a shortcoming of applying the Dubins path to connect two poses, but rather a consequence of not specifying under which constraints two poses of a path should be connected.

One of the focus of the presented research is the use of Dubins paths threat avoidance by fixed-wing systems. Therefore, one of the important constraints of those Dubins paths is that they must respect the directions of each flying leg composing a path. In order to achieve this, triplets of waypoints must be transformed according to the next figure.

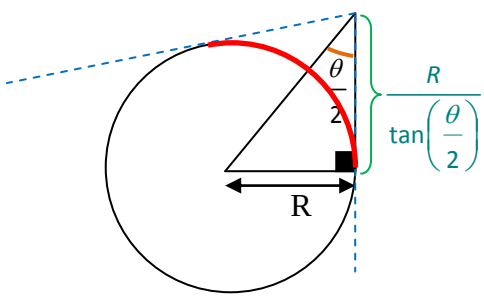


Figure 28 - Transformation of a Triplet of Waypoints So a Dubins Path Respects the Flying Leg Directions

Figure 28 shows that by using geometric consideration and creating w_{i+1}' such that

$$w_{i+1}' = w_{i+1} + \frac{R}{\tan\left(\frac{\theta}{2}\right)} \frac{\overrightarrow{w_{i+1}w_{i+2}}}{\left\|\overrightarrow{w_{i+1}w_{i+2}}\right\|} \tag{4.2.5}$$

IV - 2 - 3 - Dubins Curves With Added Logic for Perfect Flying Leg Tracking

Dubins paths do not assume that for a arc circle - line arc circle the radii have to be the same. However, when dealing with collision avoidance, it is often more interesting to reason in term of minimum turning radius, since smaller radii guarantee closer flying leg following. During a turn involved in a Dubins path, the radius of curvature is by definition maintained constant. So by keeping the same minimum turning radius for two arc circles, maximum agility is achieved.

The next figure shows how the red path of Figure 22 is smoothed using the Dubins curves with and without application of equation (3.2.4) .

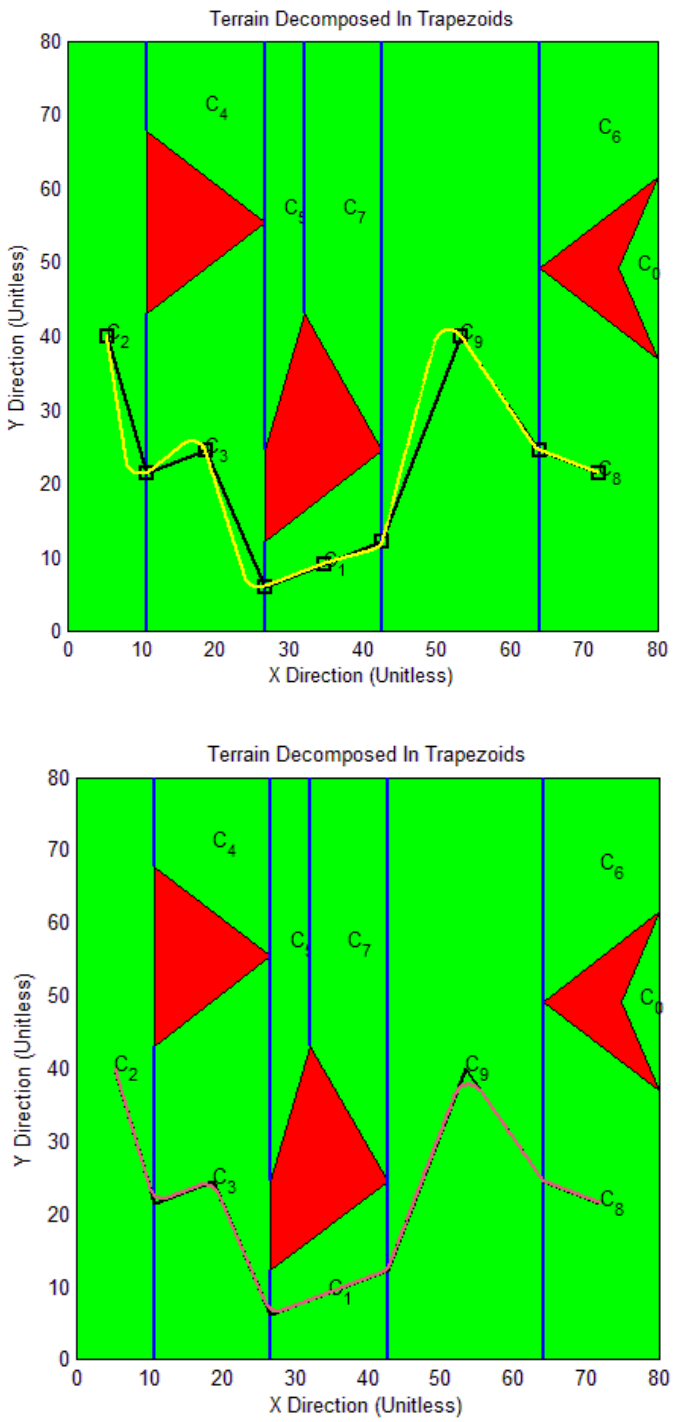


Figure 29 - Static and Dynamic (Dubins) Trajectories.

Uncorrected Dubins (Yellow), Corrected Dubins (Pink)

As explained in Figure 28, Figure 29 shows how the Dubins paths themselves do not guarantee flying leg direction following (top, yellow), but how by applying the transform described by equation (3.2.4) , it is possible to correct that problem (bottom part). The next two pictures show the effect of varying the turning radius of formula (3.2.4) by respectively playing with different bank angles (Figure 30), and different airspeeds (Figure 31).

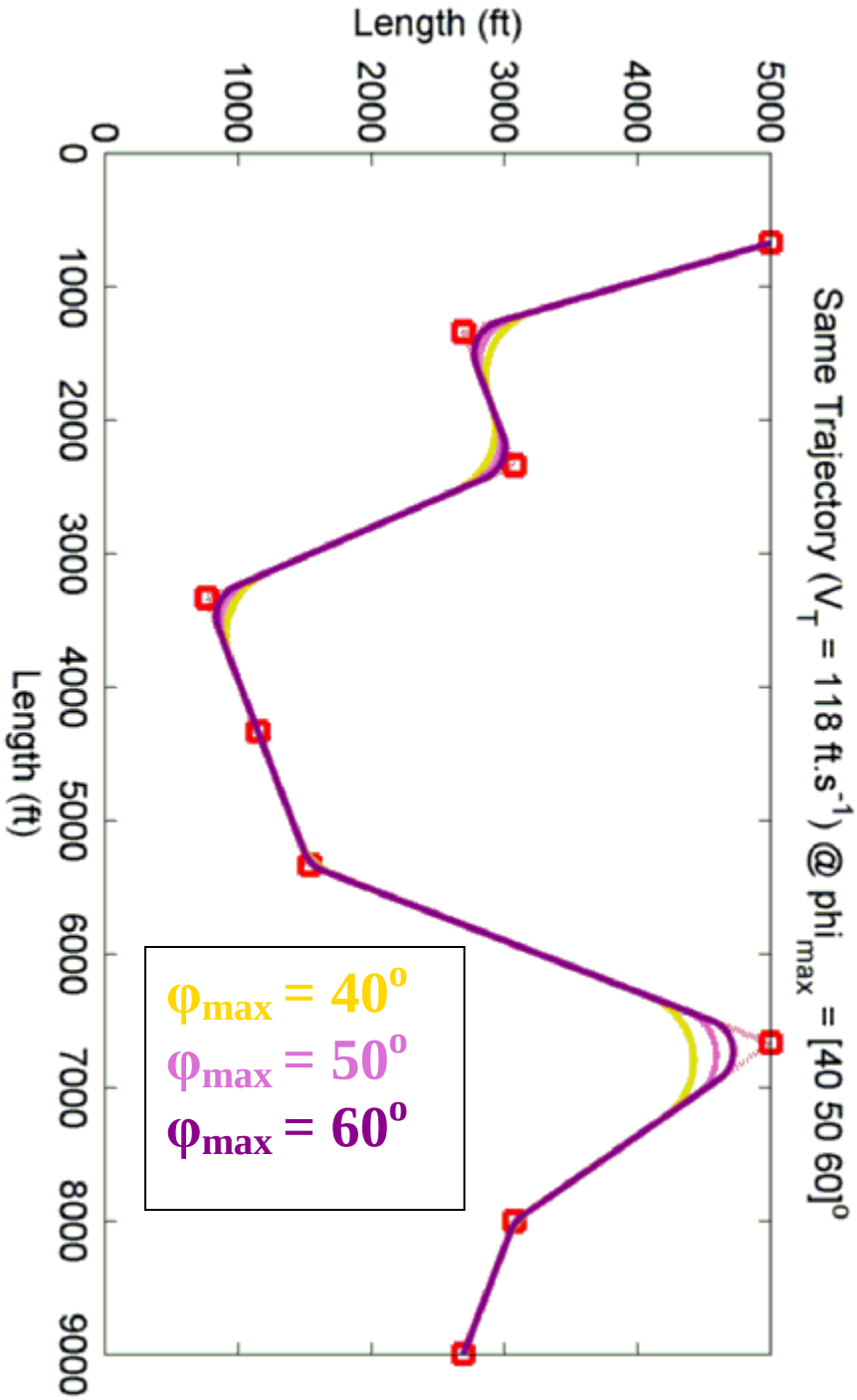


Figure 30 - Trajectory Following at Constant Speed for Different Maximum Angles

In Figure 30, the dynamics of the trajectory are embedded into the turning radius. With increasing bank angles, the denominator in the formula ($R \triangleq \frac{V_T^{\,2}}{g \times \tan\phi}$) becomes greater, and the turning radius become smaller, allowing tighter turns. This explains why for a given constant airspeed, greater bank angles provide better trajectory tracking (closer to the reference in red).

The next figure shows how variations in airspeed affect the trajectory tracking when the bank angle is kept constant.

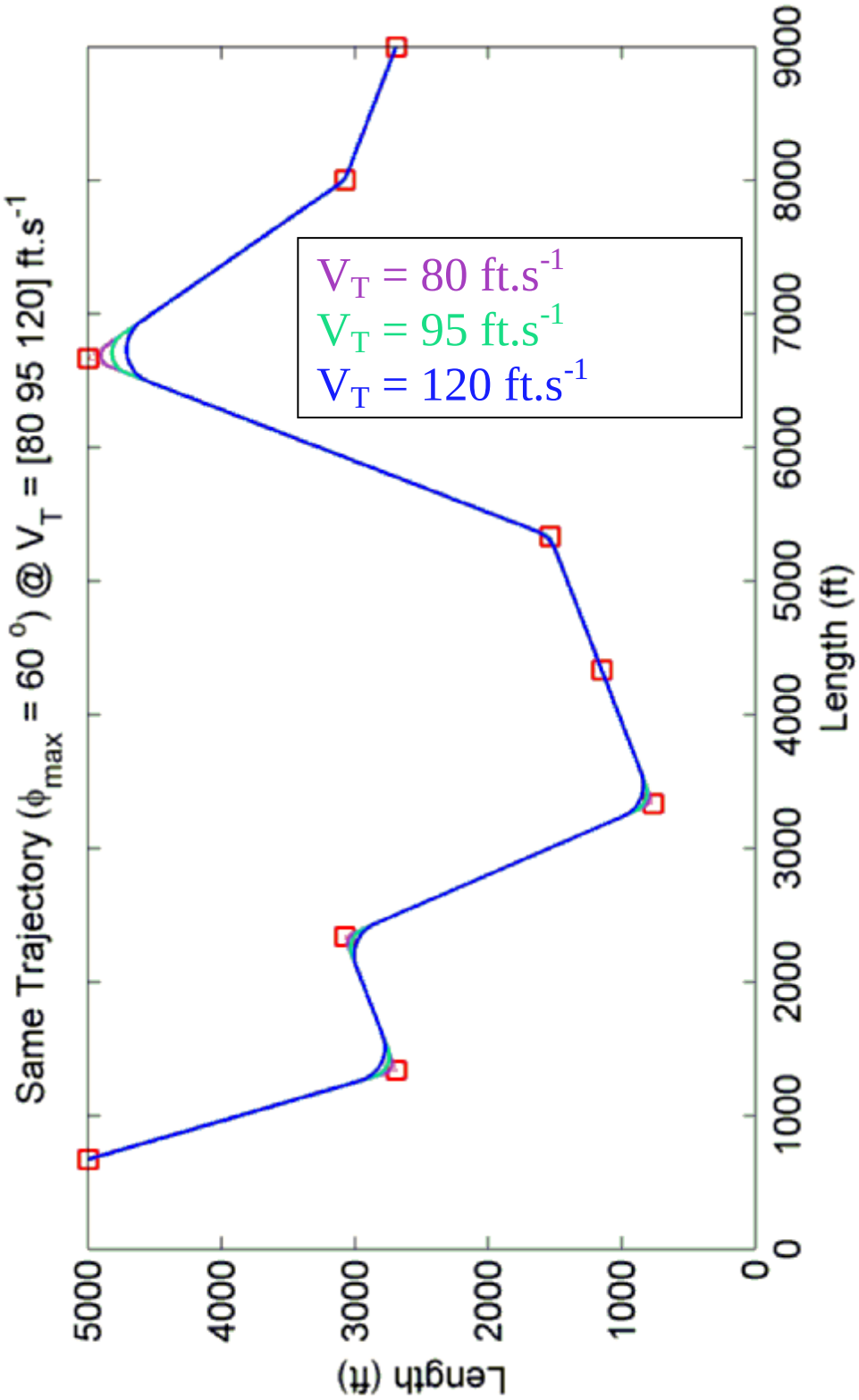


Figure 31 - Trajectory [2 3 9 1 8] at Different Airspeeds, Same Maximum Bank Angle

For Figure 31, As expected, increases in airspeed increase the numerator in the formula (3.2.4) ($R \triangleq \frac{V_T^2}{g \times \tan\phi}$) becomes greater, and the turning radius increases as well, resulting in wider turns.

Dubins curves connect two postures with a "circle-straight line-circle" sequence of primitives. But in the process, discontinuities in maneuver sequences are introduced. This is best observed in the next figure.

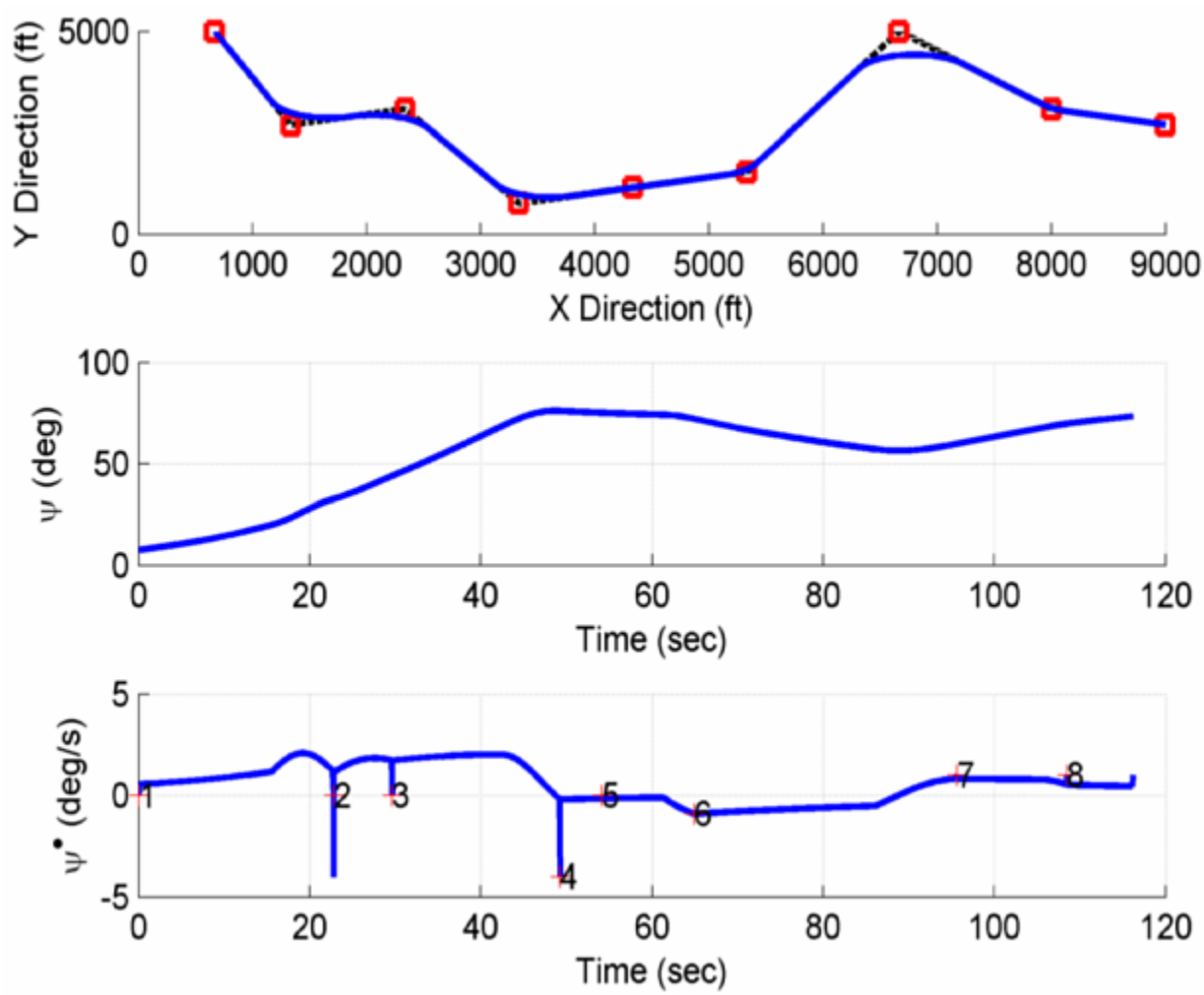


Figure 32 - Heading Rate Discontinuities Introduced by Heading Changes

As shown in Figure 32, the greatest disadvantage of using Dubins curves for path smoothing and generation of dynamic trajectories are the discontinuities in heading rate introduced by alternating arc circles and straight lines. An arc circle has a finite radius, while a straight line is a circle of curvature k = 0. Hence, it is no surprising to observe in Figure 32 a perfect coincidence between waypoint transitions (alternating arc circles with straight lines), and discontinuities in heading rate.

This problem of discontinuity is however not a permanent problem. In fact, if can be solved through the use of Euler spiral which allow smooth curvature transitions between straight line and turns. This however suppresses the assumption of a constant and minimum turning radius.

IV - 3 - The Double Dispersion-reduced RRT

Since its first use is 2000, RRT algorithms have the subject of additional research in order to overcome the limitation of probabilitically complete algorithms. Hence reference [41], where RRT search is reduced through Voronoi Bias, or the use of multiple RRTs as in reference [42], or even the improvement of optimal RRT spread such as reference [43].

The Double Dispersion-reduced RRT (DDRRT) presented here aims for the same goal: to reduce spatial search and allow real-time exploration for detection and avoidance of potential dangers. But the great difference between the previously mentioned efforts and the work presented here is that the DDRRT uses two simultaneous techniques to achieve space exploration dispersion reduction.

The first technique which the DDRRT uses is to take advantage of the trajectories generated during the offline phase presented in sections IV - 1 and IV - 2 in order to periodically spread new trees along the previously determined dynamic trajectories. This enables the DDRRT to limit its spread along feasible trajectories while not wasting resources on zones which are not worth flying by. By forcing the DDRRT to break the offline trajectory joining the departure and end point into smaller portions, the coverage of the RRT is limited, but the randomized coverage is not compromised. This means that the DDRRT still uses a random variable to select its exploration samples, but those are grown around smaller regions which belong to feasible trajectories.

The second technique which the DDRRT uses is to take advantage of the obstacle-free zones embedded in the STA networks produced during the offline. As the multiple RRTs are periodically grown along a given trajectory, that same trajectory is bounded by the various

trapezoidal cells which it traverses. During the DDRRT, samples drawn must remain within the boundaries of the trapezoidal cells traversed. This constraint is the result of limiting the dispersion of the DDRRT to the trapezoidal cells part of the offline trajectory. In two dimensions, random samples are drawn from a [0 1]x[0 1] "square" Cartesian product, which does not fit trapezoidal cells of more general shapes. Hence, a mapping function transforming the samples drawn in a "master" square to samples part of four-node quadrilateral is needed. Such a mapping function is actually used in finite element methods and bear the name of shape function (Ref. [44]). Using a shape function, the samples drawn during the exploration are first drawn in a unit square which is then mapped to the four-node quadrilateral of the trapezoidal being explored. The next figure shows the results of sampling points in a 2D square and mapping them to a quadrilateral, which is what trapezoids are in essence.

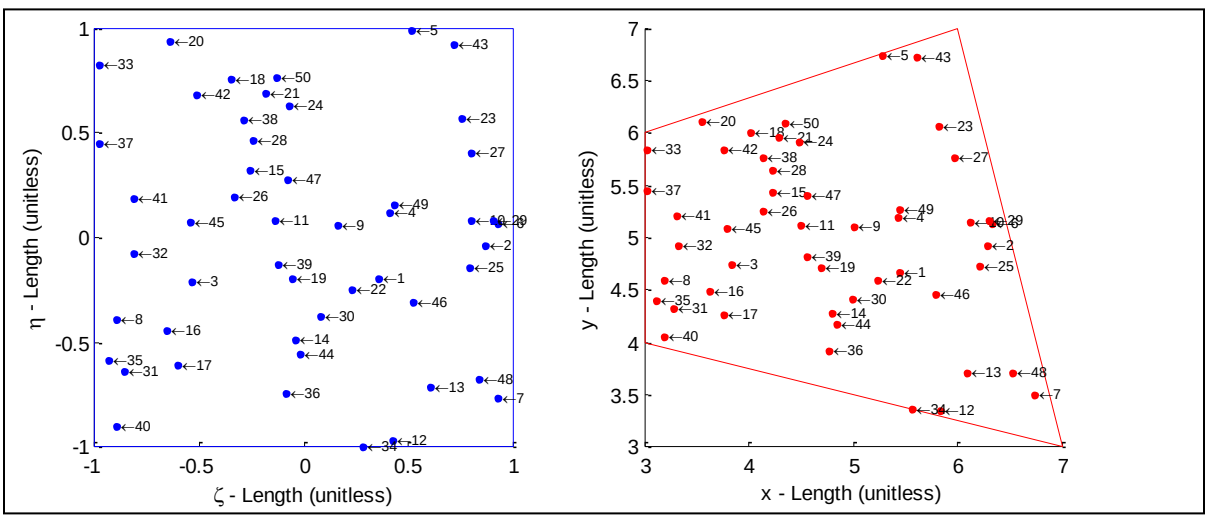


Figure 33 - Example Showing Point Sampling in Natural Coordinates and Mapping to the Configuration Space

Figure 33 is a mapping from 50 points $P(\zeta,\eta) \in [-1 \quad 1]x[-1 \quad 1]$ to the quadrilateral defined by vertices $v_1 = [3 \quad 4]$, $v_2 = [7 \quad 3]$, $v_3 = [6 \quad 7]$, $v_4 = [3 \quad 6]$. Mapping each point P from the square

to the quadrilateral is performed by multiplying the P natural coordinates shape function matrix $\begin{bmatrix} N_1 & N_2 & N_3 & N_4 \end{bmatrix}$ by the quadrilateral vertex matrix $\begin{bmatrix} v_1 & v_2 & v_3 & v_4 \end{bmatrix}^T$, where N_1, N_2, N_3, N_4 are given by:

$$\begin{aligned} N_1 &= \frac{1}{4}(1-\xi)(1-\eta) \\ N_2 &= \frac{1}{4}(1+\xi)(1-\eta) \\ N_3 &= \frac{1}{4}(1+\xi)(1+\eta) \\ N_4 &= \frac{1}{4}(1-\xi)(1+\eta) \end{aligned} \tag{4.3.1}$$

The technique of mapping points from natural coordinates to the configuration space (a simple quadrilateral in the previous example) allows sampling the trapezoids created by the trapezoidal decomposition such that the resulting points can remain within the limits of the trapezoids. This is particularly useful when developing tree searches inside the cells generated by the decomposition, and facilitates targeted tree branch local dispersion control during real-time collision and obstacle avoidance.

The question which is raised by exploring obstacle-free zones is obvious: what for? It is important at this point of the discussion to remember that the trapezoidal cells embedded in the STA networks are *a priori* obstacle-free. Unless stated otherwise, elevation maps collected by US general surveys are not guaranteed to be reliable. Consequently, while a UAS follows a predetermined trajectory generated offline, it is not guaranteed that during the mission, previously undetected obstacles might not compromise its flight safety.

V – Results

The present section focuses on the real-time mid-air threat avoidance algorithm developed for the cases where new threats not detected during offline trajectory planning must be detected during the real-time phase. First, a brief summary of the available RRT technology is presented in order to make the reader aware of their shortcoming for real-time threat avoidance. Next, the DDRRT, a new type of intelligent RRT is presented and the logic which it uses is broken down into two parts showing what those separate parts do when taken apart. Then its complete algorithm is detailed, and finally its result is shown in an example.

V – 1 – A brief overview of the Existing RRT Variations and Why They Are Not Well Suited for Real-Time Realistic Terrain Avoidance

RRT and its variations only work with a finite number of samples, which means that their spatial coverage must be optimized in order to target the exploration to the zones of interest. Classically, the RRT algorithm and its variations are used to connect two points in space. If the connection is intended for a non-holonomic vehicle, such as a UAS, this connection is called a trajectory. When the same connection is accounting for a holonomic vehicle such as a robotic arm, it is referred to as a path.

The RRT algorithm and its variations all share a same common logic which is iterated for a fixed number of samples, and based on initial conditions.

The initial conditions are:

- The operating region of the RRT (e.g. a rectangle), also referred as the world.
- The goal region, a zone in the operating region which a vehicle has to reach, such as a circle in 2D
- Potential obstacles present in the operation region, which the vehicle must avoid to preserve its safety.
- An initial vehicle position, which is referred to as the root of the RRT, and which is vertex from which the RRT is expanded from.

Once the above initial conditions are set, the RRT iterates the following logic:

1) a random variable draws D samples (typically between 0 and 1, and D the dimension of the operating region)
2) the D samples forming a vertex are mapped to the operating region (rectangle mapping in 2D, rectangular prism in 3D)
3) The RRT inspects its vertices to select the best candidate for which the metric it uses will minimize the connection cost between that candidate and the previously drawn vertex
4) An attempt at connecting the RRT best candidate to the vertex selected in step 2 is made. The RRT algorithm gives a very broad direction of how the connection must be made(*).
5) If the connection attempt performed during step 4 is found to be possible (practically no obstacles are "in the way" of connecting the best candidate of step 3 to the vertex of step 2), the vertex of step 2 is added as a node of the RRT. These 5 steps are repeated for a fixed number of iterations OR until the goal region has been reached.

As can it can be noticed, the basic (also referred to as naïve) RRT algorithm does not explicitly aims at reaching a goal region. All the RRT algorithm is good at is performing a good coverage

of the operating region it is performing the spatial coverage. Hence, a RRT reaching a goal region is consequence of its spatial coverage.

In step 4, statement is made that the basic RRT algorithm gives room to interpretation when it comes to connecting the best candidate of step 3 to the vertex of step 2. This is because the RRT algorithm can be used for different purposes, and under different additional user-defined constrains. In the case of mid-air threat avoidance targeting fixed-wing Unmanned Aerial Systems (UAS), RRTs are used in a simulation context, which typically is a scenario where a UAS tries to reach a goal while flying among obstacles in order to reach a rendezvous point (the goal region). Because such simulation is a function of time, the connection described in step 4 can be performed in two ways:

- In a first way, connecting the best node of step 3 to the random vertex of step 2 is limited by the simulation sampling time, and the speed of the vehicle. For example, a UAS with a true airspeed of 118 fps observed every 0.1 seconds can only move 11.8 feet per simulation step. Hence if the vertex of step 2 and the best node of step 3 are more than 11.8 feet apart, the connection is stopped as dictated by the vehicle motion over the sample time.
- In a second way, the RRT extension is not limited by the simulation sample time, and the connection between the best node and the selected vertex is attempted.

These two interpretations on how a vertex and a “best candidate node” must be connected give for a same set of random samples, same initial conditions, and same vehicle, very different results, as can be seen in the next set of pictures.

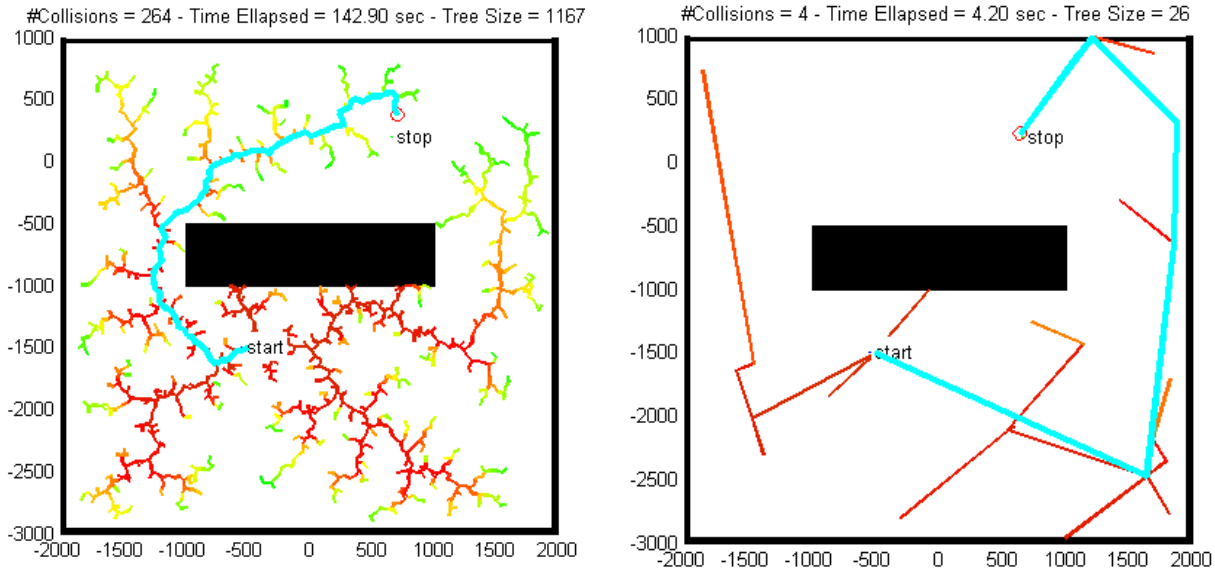


Figure 34 - Comparison between Simulation-Time-Bounded RRT Expansion and Free RRT Expansion

Figure 34 shows the difference between running (on left side) a simulation-time-bounded RRT expansion (steps 2 through 4) and without that limitation (right side). In both figures, the collision number refers to the number of occurrences of tree branches entering in contact with the black rectangular obstacle. Figure 34 was generated using the same samples which were prerecorded beforehand in order to be able to reproduce the same behavior. A simulation-time-bounded RRT expansion is characterized by a limitation of the magnitude a branch can grow during a simulation step. For example, a vehicle travelling at a 100 feet per second for a RRT updating its growth every 0.1 seconds corresponds to a maximum RRT expansion of 10 feet per simulation step. On the other hand, a simulation-time unbounded RRT growth leaves as much time to the vehicle to reach a given location. The simulation-time-unbounded RRT expansion yields metrics of 4 collisions, and 25 nodes against 264 collisions and 1167 nodes. Clearly, the naïve RRT shows much greater performance when ran in “simulation-time-unbounded”.

Perhaps one of the most important questions when using RRT algorithms is to understand what such a use implies. On the left side of Figure 34, where the RRT has 1166 branches, the tree is colored following a red to green gradient, according to the simulation time at which a given branch of the tree was created. As such red branches correspond to an early RRT construction, whereas greener branches correspond to the end of the RRT construction. Using such coloring makes it easier to understand why the RRT is qualified of "rapid". On the left side, half of the branches are red, meaning that roughly 50% of the branches were created during the first half of the RRT run. The rest of those branches were added during the second part. The following URL provides an animation of "simulation-time-bounded" run: Original RRT Simulation-Time-Bounded Expansion. To watch the "simulation-time-unbounded" counterpart, it is encouraged to visit Original RRT Simulation-Time-Unbounded Expansion.

Earlier in the discussion, a statement was made that RRT and its variations only work with a finite number of samples, which means that their space coverage must be optimized in order to target the exploration to the zones of interest. Such a reason is trivial: even with fast computers with lots of memory, drawing an infinite number of samples just is not possible. But beside this trivial reason, a more philosophical question is which of the use of RRTs. This reason can more easily be understood by watching the two previously mentioned videos. As RRT are dynamics trees, their expansion occurs at different locations in time. **This means that if an extension occurred at the North West of the operating region during iteration number i, the next extension could very well occur at the South East of the same operating region during iteration i+1. How to abruptly transition from NW to SE? By returning from NW to the root, and going from the root to SE**. While this is possible for a holonomic vehicle, this concept of tree makes less sense for a real-time moving vehicle such as a fixed-wing UAS. **In**

other words, RRTs offer an elegant solution to the problem of path planning, but can still be improved for the purpose of real-time reactive trajectory generation, because a UAS cannot disperse in real-time. **So dispersion must be reduced**.

In the following paragraphs, two variations of the RRT algorithm are discussed, and finally the Double-Dispersion-reduction RRT (DDRRT) is presented in the context of mid-air threat avoidance for fixed-wing UAS. After its presentation, a method is suggested to provide comparison between the DDRRT and those other RRT variations with no access to a STA network.

V – 1 – 1 – RRT Variation One – The Goal – Biased RRT

The goal-biased RRT is simple but potentially effective modification to the standard naïve RRT. Instead of extending the RRT branches based on a random number generator only, we add a condition: if a random sample falls within a certain interval, we extend the RRT toward the goal, hence the "goal-biased" qualifier. The Mersenne Twister pseudo-random number generator issues approximately 2^{19937}-1 numbers which follow a guaussian distribution when performing a QQ-plot. This practically guaranties that for n values drawn by the MT, each value within the interval [0, 1] has an equal chance to be chosen. Any algorithm using a similar algorithm is qualified of "randomized". So because RRTs use a "good" random generator (the MT), all values within the range [0, 1] have the same chance to be drawn. This is this same fact which is used to shape the probability distribution. A number N such that [0,1]=[0,N[U[N,1] is used to partition the sampling interval [0,1]. Then, for a given draw of the RRT, the drawn value d_{rand} falls within [0, N[, we choose to point toward which we extend the RRT to be the goal point. Hence if N = 0.5, the goal-biased RRT gives 50% chances to extend its branches toward the goal. As N gets closer to 1, the more the chances to select the goal as a direction toward which the RRT will be extend. This means that as N increases, the more the RRT is biased. The next figure shows what happens for 10 different biases.

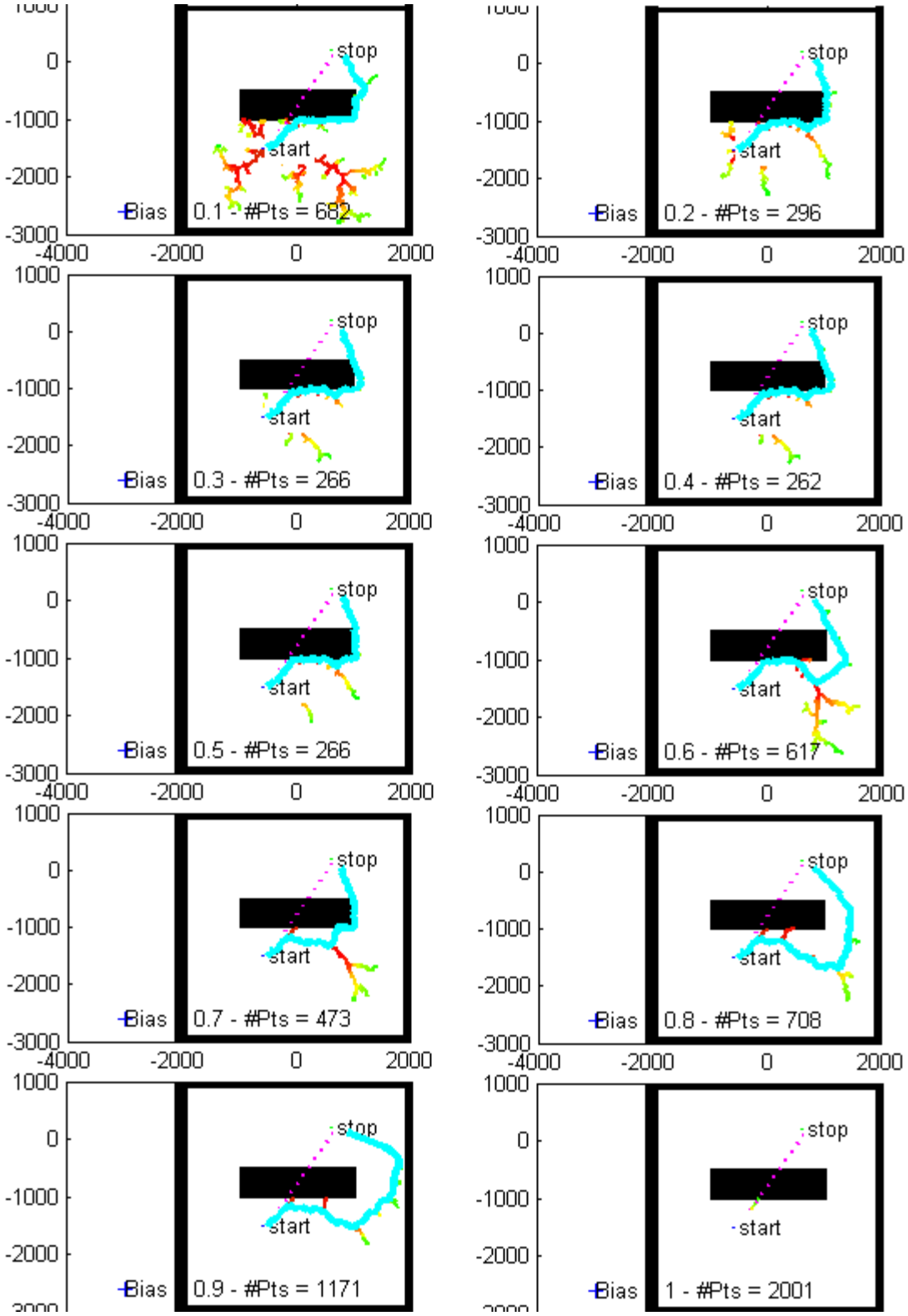


Figure 35 - Goal - Biased RRT - Ten Biases

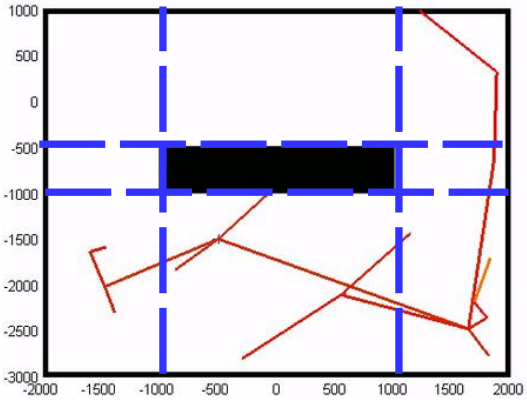
As can be seen in Figure 35, the Goal-biased RRT spread varies greatly depending on the biasing. Figure 35 is divided into two columns, and the RRT biasing increases from left to right, and top to bottom. This means that the first biasing values can be found on the first row, increasing from left to right (0.1 and 0.2 corresponding respectively to 10% and 20% biasing) and the last biasing values can be found on the fifth row, increasing from left to right (0.9 and 1 corresponding respectively to 90% and 100% biasing). The branches in cyan show the path from the start point to the end point when it has been found. **From Figure 35, it is can be seen that an increase in goal biasing can result in a decrease in RRT dispersion. However, this is not systematically true, because the RRT is a randomized algorithm. This means that for the unpredictable MT seed sequence, if the drawn seed do not fall within the biasing interval [0, N], RRT dispersion occurs**. Therefore, the effect of biasing a RRT toward the goal point is to be taken generally, meaning that biasing an RRT toward a goal point generally yields a decrease in RRT dispersion, **but no qualitative conclusion can be drawn from comparing different biases for a same environment (world and obstacles), because of the randomized nature of the algorithm (the seeds drawn may not necessarily fall conveniently within the biasing interval), and because of the obstacles' locations**.

V – 1 – 2 – The Multiple RRT

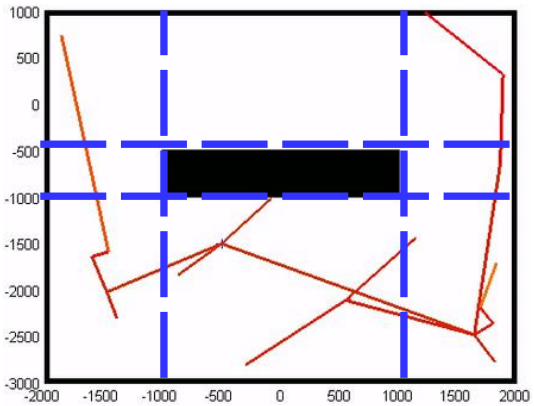
This short section does not have explicit figures demonstrating how a Multiple RRT (MRRT) expands to find a path between two points while avoiding obstacles. The reason can easily be understood by taking a pair of vehicles, each of which using a RRT for real-time goal reach and threat avoidance. Vehicle 1 starts from the starting position to reach the goal position, while Vehicle 2 starts from the goal position to reach the starting position. Basically, the two vehicles converge with each other. This scenario is plausible because there are as many vehicles as there are RRT (two). In the context of Aerospace, and more specifically regarding UAS, MRRT could be employed in the context of a swarm, but in the context of non-collaborative (no swarm) threat avoidance targeting unmanned fixed-wing aerial systems, MRRT do not show a practical use.

V – 1 – 3 – Conclusion Regarding the Existing RRT Algorithms and Why another Type of RRT Must be Developed for Real-Time Reactive Threat Avoidance

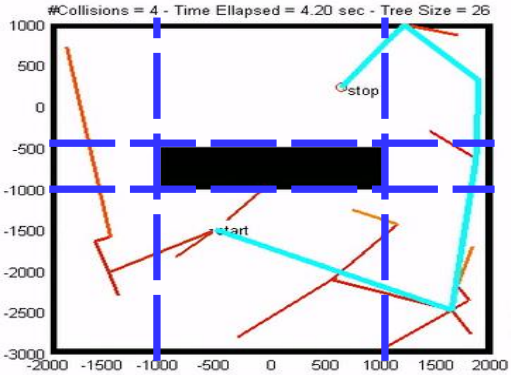
The Goal-biased RRT presents the potential of dramatically reducing the world explored before converging to the goal. However, Figure 35 shows that a fixed biasing can lead to obstacle trapping, and that the efficiency of a fixed-goal biasing depends on the terrain it is applied to. The multi-RRT however does not fit the purpose of non – collaborative threat avoidance, as its use would only serve a collaborative scenario. However, both algorithms show a common defect: the tendency to "jump" in the exploration from different points of view around an obstacle, and such problem is emphasized in the next figure.



A – Beginning of Path Finding.
Majority of exploration going to the RHS.



B – Middle of Path Finding.
Exploration switches back to LHS.



C – End of Path Finding.
The exploration on LHS was useless.

Figure 36 - A Common Trait to All Previously Developed RRT Algorithms - The Tendency to Alternatively Grow on Two Opposite Sides of an Obstacle

Figure 36 shows the lack "ubiquitous" nature of previously developed RRT algorithms. In its part A, it can be observed that the majority of the RRT has spread within the bottom and right hand side of the black obstacle. But in part B, which shows a further development of the RRT, it can be seen that the RRT exploration has gone back to exploring the left hand side (LHS). But the convergence of the RRT toward the goal in part C shows that at the end of the exploration, the LHS of the black obstacle has been abandoned. Hence Figure 36 shows that RRTs explore the space given to them by "jumping" across obstacle, a behavior which is not acceptable for

real-time threat awareness. The blue dashed lines in all three subparts of Figure 36 give a hint on how trapezoidal decomposition can remove this problem, by forcing RRTs to stay within some boundaries before expanding the exploration. This concept is further explored and explained in the next sections which describe the Double Dispersion reduced RRT (DDRRT).

V – 2 – The DDRRT

The double dispersion reduction of the DDRRT comes from two results of the offline trajectory generation:

1) At the spatial level: Availability of dispersion reduction at the trapezoidal cell level for different flying layers. This spatial dispersion reduction is abbreviated as SDR.
2) At the punctual level: Availability of the dispersion reduction at the flying leg level. This punctual dispersion reduction is abbreviated as PDR.

V – 2 – 1 – The DDRRT Spatial Dispersion Reduction

The Trapezoidal decomposition performed during the offline trajectory planning by the Advanced Mapping and Waypoint Generator (AMWG) generates cells whose centroids are used to create Dubins path trajectories. Those cells are inherently 2D for a given altitude layer, but since they have fixed contours between two altitudes, they also allow a dispersion reduction at the volume level. This means that in the best case scenario, the DDRRT spreads its branches within the boundaries of a sequence of cells which were judge safe enough such as to be used for the creation of a safe trajectory between two locations. If the two-dimensional aspect only is investigated, then the DDRRT spreads its branches among the flyable cells used for a given trajectory it is trying to follow. If the three-dimensional aspect is investigated, the DDRRT spreads its branches among the extruded flyable cells used for the trajectory it is trying to follow. Referring to **Figure 29**, the best offline trajectory is made using the centroids of the following cells: [2, 3, 1, 9, 8]. It is assumed that those cells can be used for altitudes comprised within the altitude range 0 to 10 elevation units. The next figure shows the principal cells of interest for the area/volume dispersion reduction of the DDRRT.

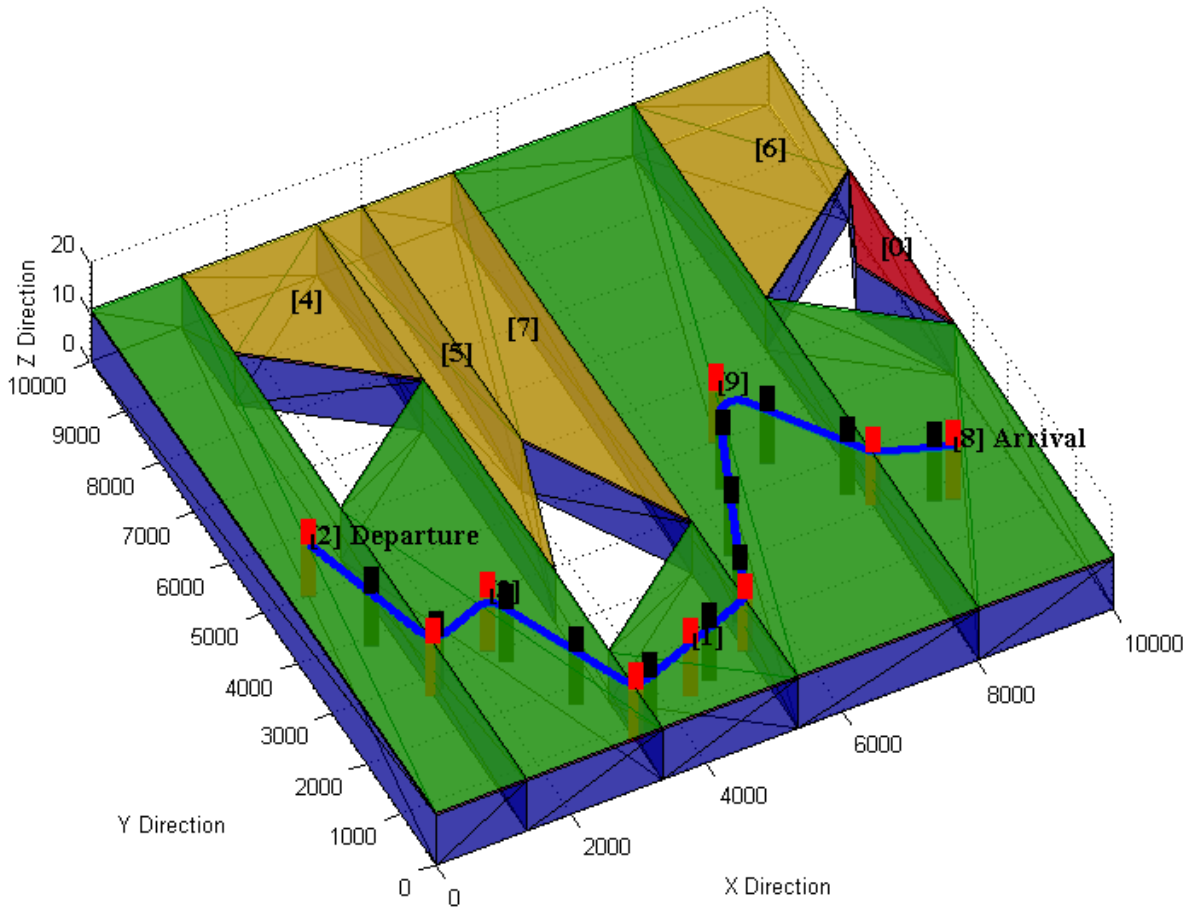


Figure 37 - The First Dispersion Reduction Level of the DDRRT Showing Green Volumes as Primary Areas to Target the DDRRT Real-Time Exploration

Figure 37 shows the extruded trapezoidal cells corresponding to the path [2, 3, 1, 9, 8], whose cells' surfaces are colored in transparent green, while the rest of the cells is colored in transparent yellow. The transparent blue faces are only used to show that Figure 37 is an extruded trapezoidal decomposition. The white "holes" are the result of removing the extruded obstacles from the decomposition. This means that Figure 37 shows the decomposition only without the extruded obstacles.

The yellow volumes are not to be discarded from the decompositions. They do not contribute directly to the path from cell 2 (departure) to cell 8 (arrival), but the original safety of the green cells is compromised, those yellow volumes can be used to find alternative routes, as it has been explained in a previous discussion regarding the Synthetic Terrain Avoidance (STA) network file. **The interest brought by the spatial dispersion reduction of the STA network is that it allows prioritization of the areas of real-time exploration, while allowing coverage and the choice of pre-computed alternative paths**. The following figure shows a DDRRT run using 200 samples and following the cell sequence [2, 3, 1, 9, 8].

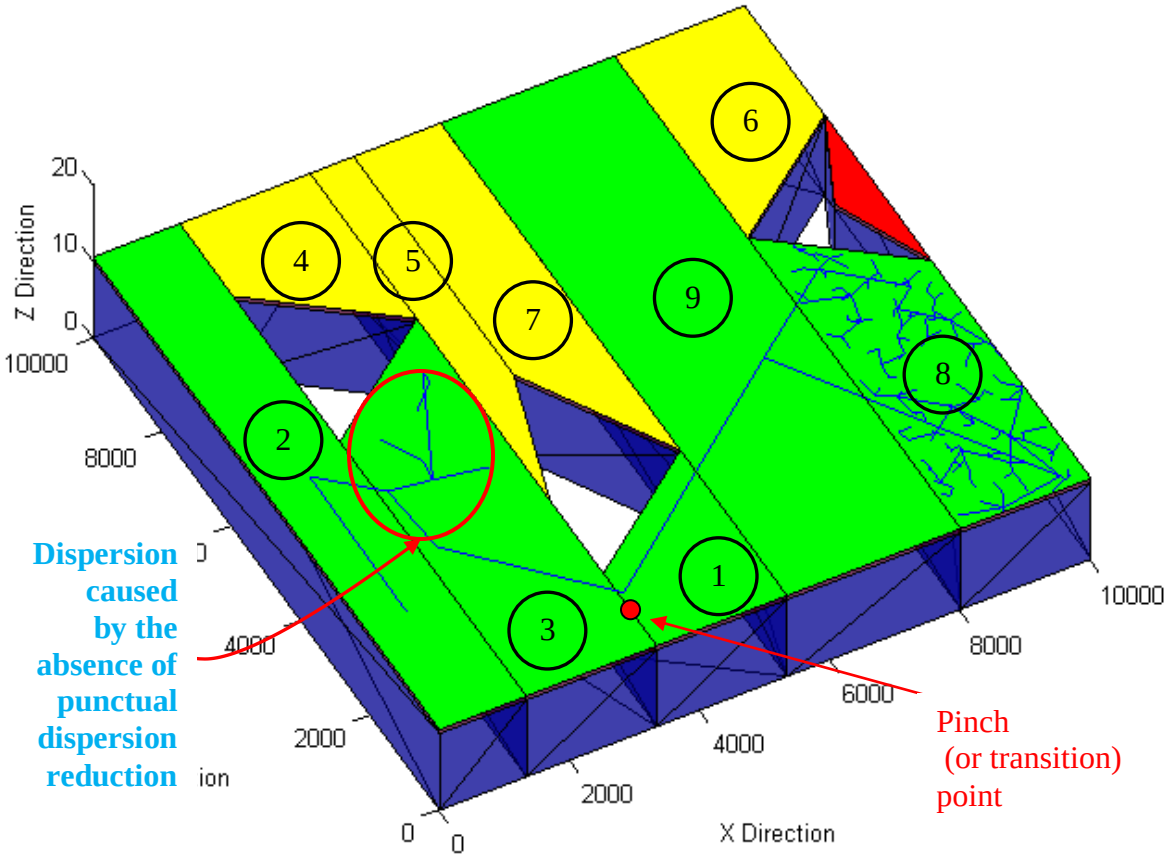


Figure 38 - The DDRRT Using the Spatial Dispersion Reduction Only and No Alternative Routes

As it shown in Figure 38, the DDRRT depicted only employs the spatial dispersion reduction with no punctual dispersion reduction, and no alternative route suggested. The absence of punctual dispersion reduction is intended here in order to show how the DDRRT expands with only one dispersion reduction enabled. This explains why the transition from [3] to [1] involves some extra branch spreading in region [3] before transiting to cell [1] without even going through the pinch point. With the addition of the punctual dispersion reduction, the DDRRT would go from cell [3] to cell [1] by passing through the pinch point, thus further limiting the dispersion in cell [3].But even without the punctual dispersion reduction enabled, it can be seen that most of the tree expands in cell [8], meaning that the progression to the cell containing the goal only take a few step. The second option disabled from the DDRRT is the goal reach checking. 186 cells are spread in cell [8] which contains the goal region. The reason for which the goal reach algorithm breaking condition is not enabled in Figure 38 is because the intent is to show the DDRRT spread in the case of a trapezoidal cell. The seed mapping from the sample space to the trapezoidal space is performed using the shape functions previously described in the discussion based on Figure 33.

Another way of looking at the results of Figure 38 is that by using the trapezoidal cells, although the DDRRT is not using punctual dispersion reduction, it evolves in a connected free space, and the only collision are the result of a lack of punctual dispersion or intermediate goal biasing. But because the path from the centroid of cell [2] to the centroid of cell [8] is determined at the special level by a sequence of trapezoids, the DDRRT is implicitly biased toward the main goal, even in the absence of intermediate waypoints.

Furthermore, the absence of seeds in the yellow regions shows that the DDRRT is not fully using the capabilities of the STA network when it comes to alternative routes. This is **again**

intentional as the logic to decide of alternative routes cannot solely depend on failures depending on the spatial dispersion reduction. In order to gain a better understanding of this phenomenon, it is best to use an example. This example is shown in the next figure.

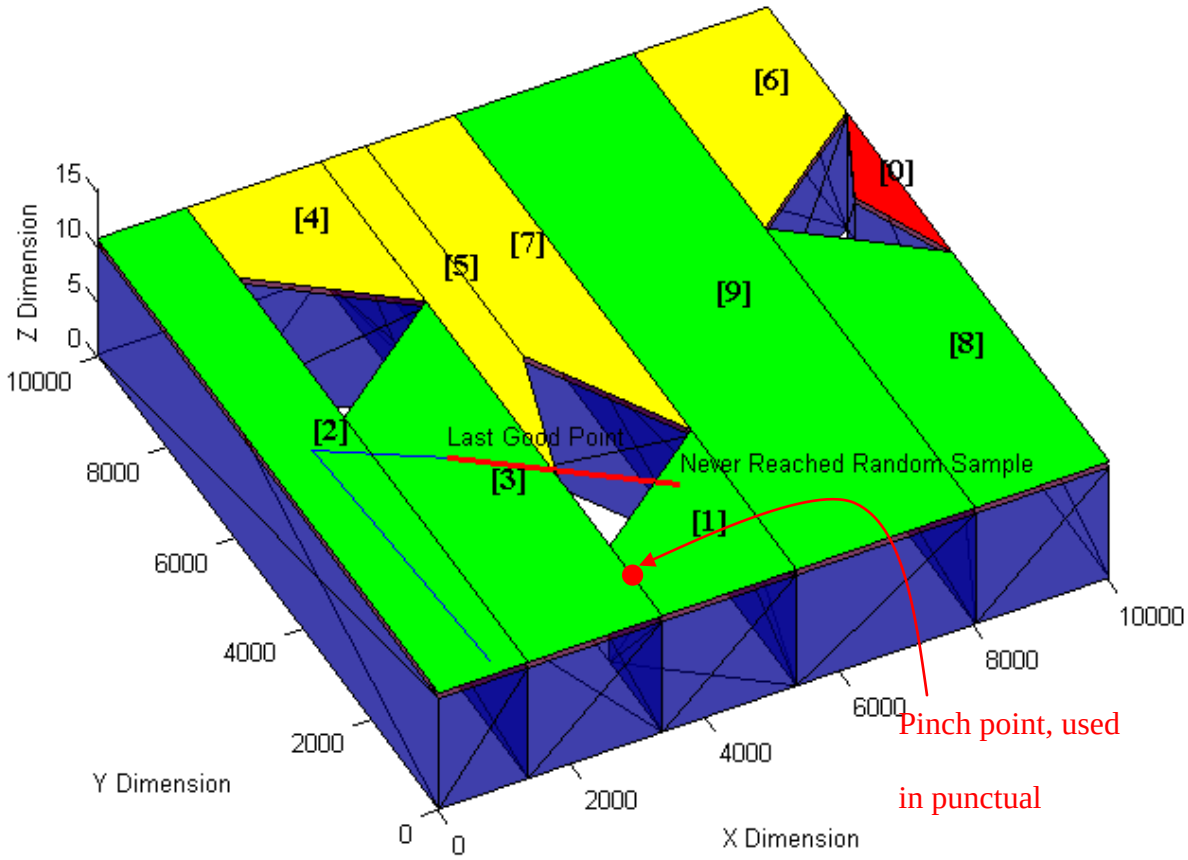


Figure 39 - DDRRT Using Spatial Dispersion Reduction Only for Alternative Route Decisions

Figure 39 shows a scenario where the DDRRT using the spatial dispersion reduction (SDR) only attempt to connect a position located in cell [3] (the last good point) to a position located in cell [1] (the never reached random sample). Because the DDRRT is not using punctual dispersion reduction (PDR), it is unaware of the pinch point shown in red, and does not use it. Because the free-space made by cell [3] and cell [1] is concave, the straight connection between the two positions can and does form outside of the free-space, causing a failure. But this failure is not bound to happen. However, when designing algorithms, a good practice is to imagine worst case

scenarios. Because the DDRRT only uses SDR without PDR, failure to move from cell [3] to cell [1] will lead to decide that the transition from cell [3] to cell [1] is dangerous, while the last safe position is in cell [3]. The DDRRT in this case is only using SDR, and thus does not know how to overcome convexity problems. Knowing that the last "good" cell is cell [3], the problem of reaching the goal located at cell [8] is which of finding alternate routes connecting cell [3] to cell [8], assuming (incorrectly) that the transition from cell [3] to cell [1] is impossible. Interrogating the STA network for alternative routes is always possible, but the alternative routes are not guaranteed to exist. In the case of Figure 39 however, alternative routes between cells [3] and [8] not containing the sequence [3 1] do exist, and are returned by the DDRRT as follows:

[3 2 4 5 7 9 8]

[3 5 7 9 8]

Based on the distance metric calculated using pinch points and centroids, the next best alternative path is [3 5 7 9 8], with a total flying leg distance of 29488 units versus 50019 units for the sequence [3 2 4 5 7 9 8]. In the worst case scenario, the SDR-limited DDRRT fails while attempting to connect cell [3] to cell [5], rendering the sequence [3 5] unsafe. Assuming the same faulty logic is employed for the last alternative sequence [3 2 4 5 7 9 8], the worst case scenario yields to a failure for the SDR-limited DDRRT to connect cell [3] to cell [2]. **In such scenario, all the alternative sequences are made unsafe because the DDRRT only employs the SDR alone which can result in breaking world concavity proof brought by the pinch points.**

Figure 37 contains the two dispersion levels of the DDRRT: **the spatial level** with the trapezoidal cells in green (preferred), yellow (alternative) and red (disconnected), but also the **punctual level** with the vertical sticks representing the waypoints locations for the punctual dispersion reduction, which is discussed next.

V – 2 – 2 – The DDRRT Punctual Dispersion Reduction

The DDRRT's second type of dispersion reduction is the punctual dispersion reduction, in contrast with the first type which offers a **targeted coverage**. For a given altitude layer, trapezoidal cells' centroids are used to generate flyable trajectories. Those continuous trajectories which are made of Dubins paths are then discretized in order to form a succession of flying legs delimited by waypoints. Those same waypoints are as many intermediate goals separating the departure from the arrival, and necessitate goal-biasing or punctual dispersion reduction (PDR) in order to limit the DDRRT dispersion. This is best seen in the following picture.

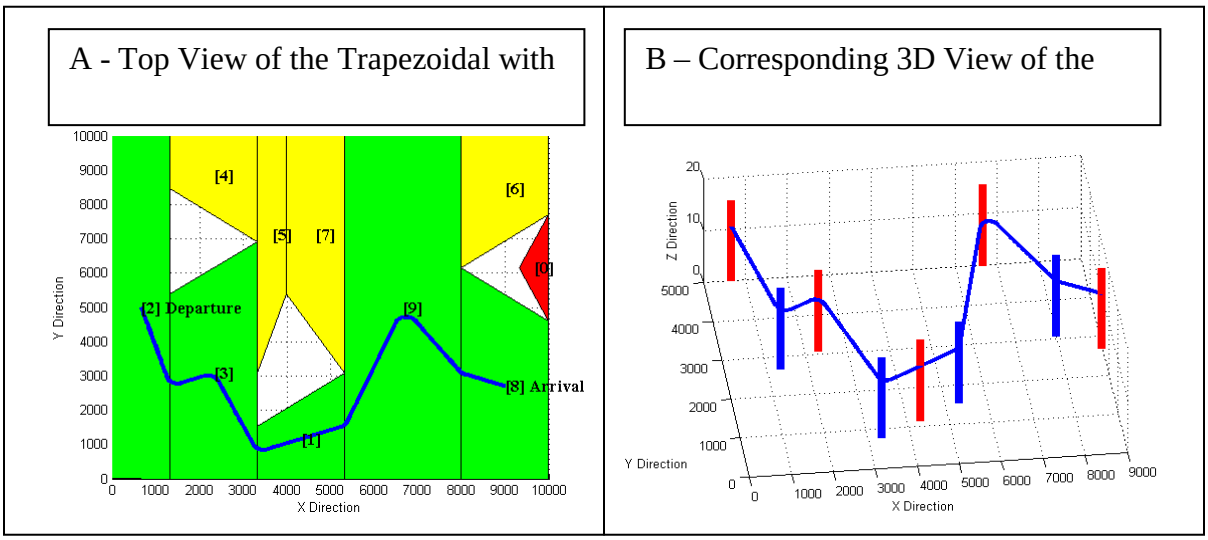


Figure 40 - The Waypoints Designed for the Punctual Dispersion Reduction of the DDRRT

Figure 40 contains two parts. The part labeled A shows how the Dubins trajectory fits with the trapezoidal cells it is generated from. The part B shows a perspective view of the Dubins trajectory of part A, with the addition of the waypoints employed for its design. Those waypoints are shown in red when they correspond to a centroid and in blue when they correspond to pinch points whose design is explained in Figure 23. Because those waypoints are safe "a priori", the best case scenario for a DDRRT is to rely on those waypoints only. Hence, assuming zero

uncertainty regarding potential threats (ideal case), the punctual dispersion reduction of the DDRRT must allow a complete goal biasing toward the intermediate waypoints between the departure and arrival points. But the whole purpose of this research is to produce an algorithm which can save a UAS when a flight mission does not succeed as planned.

An obvious danger of keeping the a constant PDR for the DDRRT is that if the PDR is too goal oriented, an obstacle in between the DDRRT current exploration point and that goal very likely lead to a collision. Therefore, if the PDR is too high and there is a thread in the way of the next goal, RRT such as the trivial goal-biased RRT previously discussed would be more likely to fail. But the inverse problem is not more appealing: if the PDR is too low, the RRT dispersion reaches its maximum dispersion potential, and for a limited number of samples, can yield to a failure to reach the a goal while having wasted resources spent on exploring areas of little interest. **That is why the PDR of the DDRRT is an adaptive goal biasing**.

Hence the design choice for the DDRRT punctual dispersion reduction is to count the number of consecutive hits with threats, and perform the same counting for consecutive absences of hits. This concept is best explained with an example of a holonomic vehicle whose exploration is driven by the PDR only of the DDRRT.

Assuming that regardless of the random sample drawn, this sample is discarded in order to select the goal as the actual position for the DDRRT to extend toward. But in the way of that goal lays a threat, and the DDRRT fails to connect to the goal. The PDR initially set to 100% of goal biasing is reduced to 90 percent. During the next 4 consecutive draws, samples are drawn with respective goal biasing of 80, 70, 60, and 50, each 10% decrease being the result of a previous collision with a threat. For the sixth draw, no threat is encountered, and therefore the PDR is increased back to 60%. Because this is an example, it is assumed that although the probability for

the goal to be selected is 60%, the next draw falls within the interval]0.6, 1], and the PDR is further increased to 70% of chances for the goal point to be chosen. **In the absence of threat, the PDR of the DDRRT alone force the DDRRT to have a zero-dispersion**. Such fact is shown in the next figure.

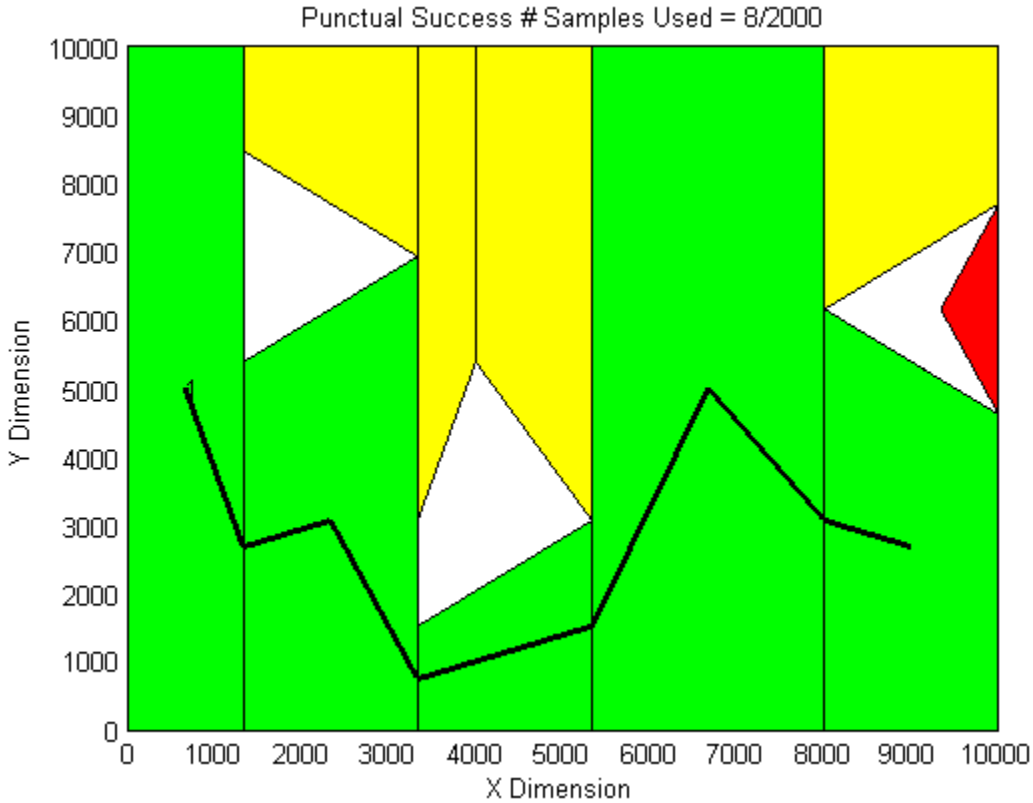


Figure 41 - DDRRT Limited to PDR Only and Following a Succession of Waypoints With Zero Dispersion

Figure 41 shows that by only using the PDR of the DDRRT it is possible to achieve a zero-dispersion for the DDRRT. This result is trivial but important: in a best case scenario with a perfect description of flying zones, any RRT algorithm should limit its dispersion as much as possible. This is the case of Figure 41, which shows close tracking of the blue trajectory of Figure 40 part B. **In other words, assuming a perfect knowledge of the flying zone, and given a set of flyable waypoints, there is no need to rely on the SDR of the DDRRT**. The waypoints

of Figure 41 are generated using spatial considerations (centroids, decomposition connectivity), so those waypoints are inherently an implicit consequence of spatial reduction, and if there is no reason to react until necessary, the PDR offers a much finer trajectory tracking than the SDR. However, like the sectional charts in general aviation, STA networks cannot anticipate change in terrain topology. Hence, what can happen for example is that the STA network used throughout this chapter is not accounting for an obstacle or threat whose presence is posterior to the time at which the STA network was created. **What happens in such a case is that if the PDR is scheduled to follow a sequence of waypoints (such as it is the case in Figure 41) of which at least one is obstructed by obstacles, then the entire sequence is rendered invalid**. The only reason why a PDR follows a sequence of waypoints is when that sequence has been designed through the Advanced Waypoint and Mapping Generator (AWMG) assuming perfect knowledge of obstacle locations. But those waypoints' purpose is to allow a UAS to follow a flyable path. If one of the waypoints is corrupted, the entire sequence is not flyable anymore and needs to be redesigned, if possible. **Later in this chapter, it will be shown that a corrupted sequence of waypoints can be achieved by reactivating the SDR**. But because this section is about the PDR only, the next figure shows why when the sequence is corrupted only the final waypoint should be kept.

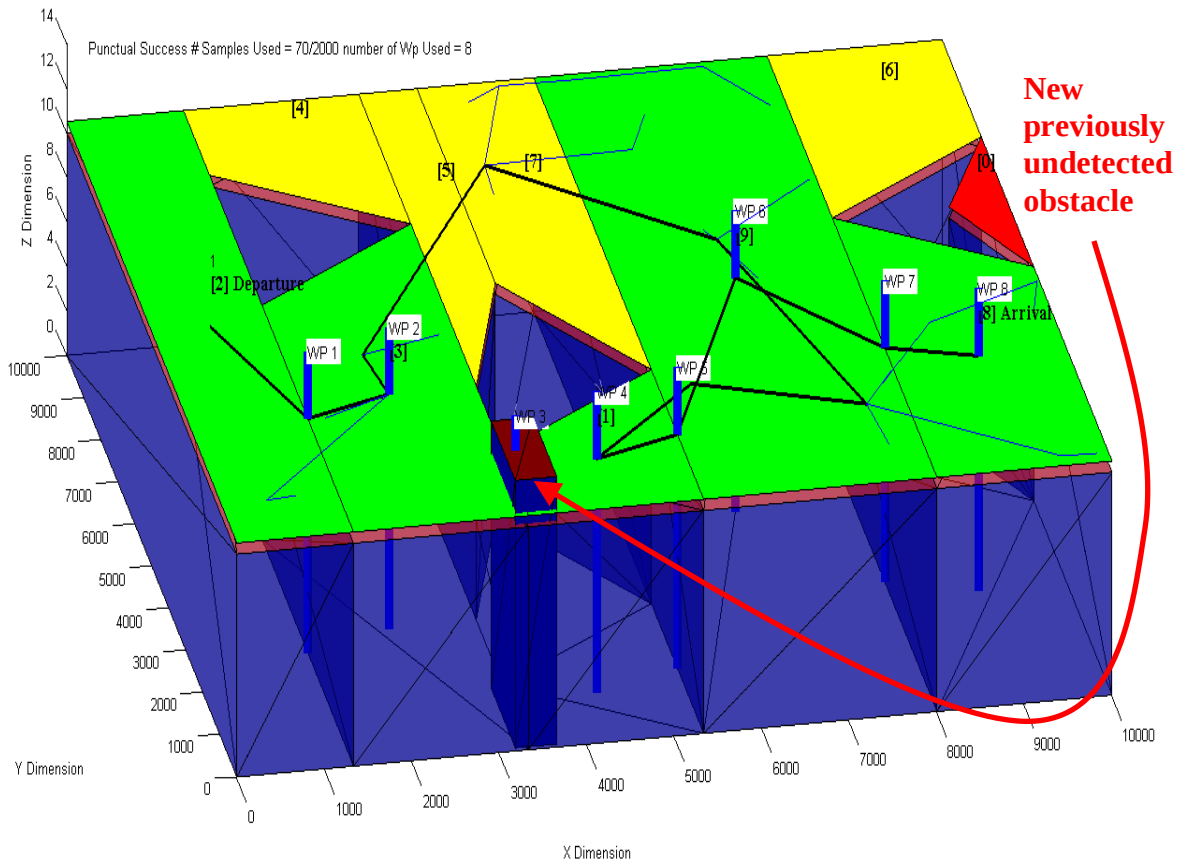


Figure 42 - DDRRT in PDR Mode Only Skipping Only WP3 among the Corrupted Sequence

Figure 42 shows explicitly the reason why a waypoint sequence must be completely discarded when the safety of one of its constituents is compromised. This figure shows the same exact waypoints as in Figure 40 part B, but for color choice limitation reasons, those same waypoints are shown as blue sticks. The PDR starts decreasing its dispersion reduction around waypoint WP3, close to the undetected obstacle. After expanding a few branches, around WP2 to find an alternative route, the tree expansion passes close to WP6 and WP5 in order to converge to WP4, the next waypoint after the corrupted waypoint WP3. Then the tree extends to WP5 which it has already reached while on its way to WP4. The rest of the tree progression is equally inefficient. **Hence, Figure 42 is what justifies why when a waypoint in a flyable sequence is corrupted**

all the sequence must be discarded in order to only keep the last waypoint or goal, as following the rest of the sequence can lead to a worst case scenario. In contrast, the next figure shows a PDR-limited DDRRT tree expansion for the same configuration, but discarding completely the initial waypoint sequence in order to conserve the last waypoint (goal) only.

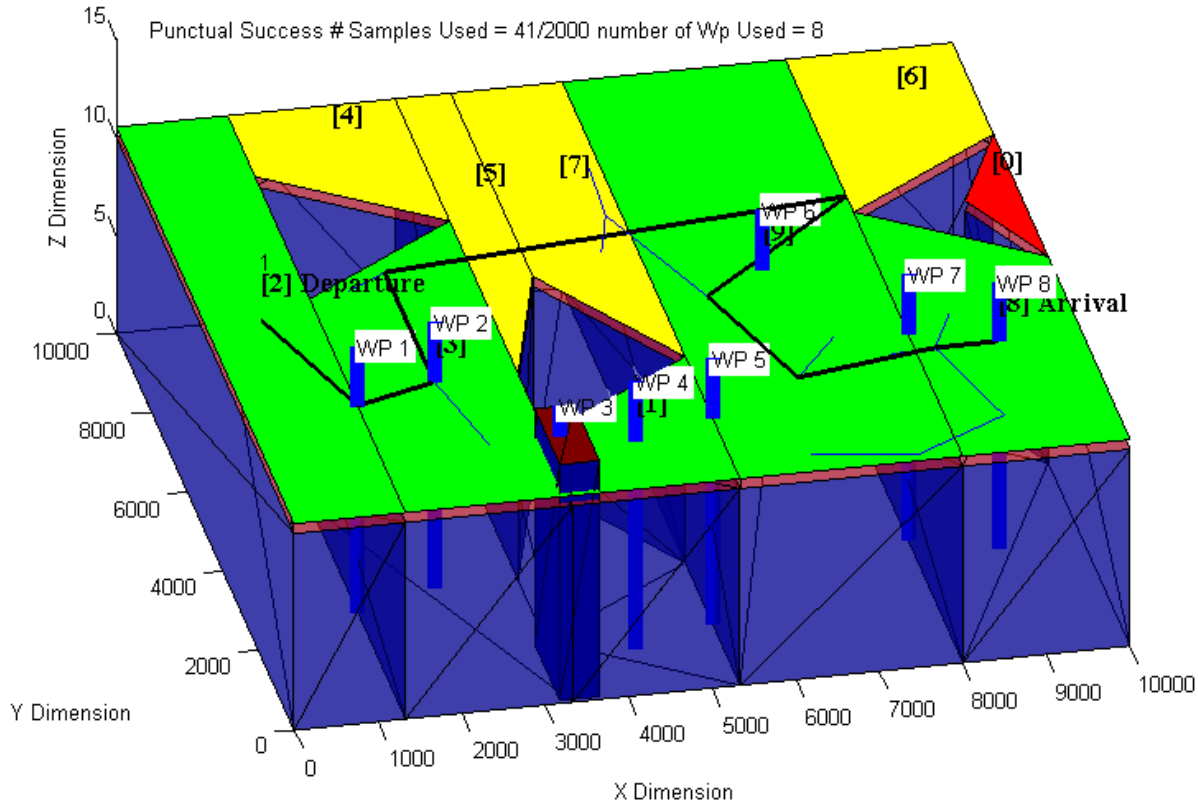


Figure 43 - PDR-Limited DDRRT Showing Its Tree Extension after Discarding the Original Waypoint Sequence In Order to Converge Directly to the Goal

Figure 43 shows a PDR-Limited DDRRT expansion reaching the final goal, waypoint 8, once waypoint safety corruption has been identified around waypoint 3. The goal is reached after 41 seeds, but this result in itself cannot be used to draw a general conclusion regarding the efficiency of the dispersion reduction. The main reason for this is that although any goal-biasing introduced in an algorithm breaks partially its randomized nature, and even the little leeway left

can result in vast fluctuations among several run of the algorithm. As explained previously, the **PDR dispersion reduction is based on an adaptive linear goal biasing**. The moving holonomic vehicle is a sphere of radius 50 units. **Every time the sphere hits an obstacle, its bias is decreased by 10% which is a design choice**. But as explained in the discussion regarding fixed goal biasing, shaping the probability distribution only influences the chance of goal-biasing, **but not the result of the goal-biasing**. Although trivial, this last statement is important to explain why the result of Figure 43 cannot be used to draw conclusions regarding the evolution of the departure to arrival tree connection as a function of the number of seeds used. But the number of times the algorithm fails for a certain number of runs is an indicator of its potential. The next figure shows the results of 100 runs of the PDR-limited DDRRT for the scenario involving an unexpected obstacle on waypoint 3.

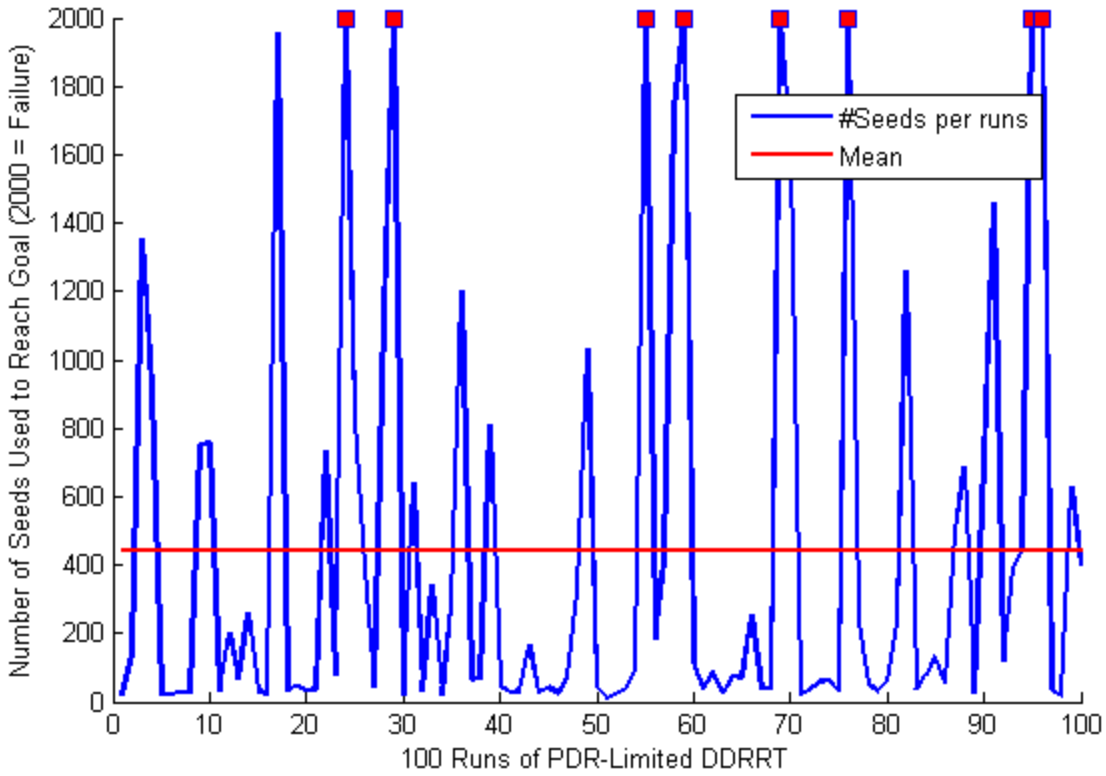


Figure 44 - Results of 100 Runs of PDR-Limited DDRRT

Figure 44 shows that out of 100 runs, only 8 runs result in a failure to reach the goal. But 100 runs to estimate a partially randomized algorithm such as the PDR-limited DDRRT is not enough to draw a conclusion regarding the success rate of the algorithm. Also, in addition to being randomized, the PDR-limited DDRRT is not using the STA network in Figure 42 through Figure 43, so it is subjected to navigation among four obstacles. But instead of perfecting the PDR-restricted DDRRT, the next section shows how the SDR and PDR work together to dramatically reduce the number of seeds involved to reach the final goal.

V – 2 – 3 – The SDR and the PDR used Altogether in the DDRRT

The previous two sections respectively detailed the logic employed in the Spatial Dispersion Reduction (SDR) and the Punctual Dispersion Reduction (PDR), for a holonomic vehicle. This section explains how the two types of dispersion reduction are used altogether.

As shown in Figure 38, the PDR alone allows the PDR-limited DDRRT to grow like a naïve RRT, but along trapezoidal cells of choice, which as a consequence limit the DRRTT dispersion. On the other hand the PDR works by performing an adaptive punctual dispersion reduction at the waypoint level, those waypoints being the result of offline computations from the AWMG. The strength of the SDR is that it uses available knowledge to limit dispersion. The strength of the PDR is that it is reactive: it follows waypoints assumed to be safe, and increases dispersion (decreases bias or PDR) when a threat limit has been reached. The SDR makes explicit use of the trapezoidal decomposition, while the PDR does so only implicitly (the waypoints are generated by the AWMG) until a threat is met. **It has been shown the previous section that the linear probability shaping for the PDR allows a fallback mechanism when an obstacle is encountered while follow waypoints assumed to be flyable**. But for the sake of efficiency, it is preferable to interrogate the STA network in order to determine if alternative routes are available. Not only the latter approach leaves less control to randomizations processes, but it also takes advantage of the STA network. The next figure shows how the STA network is used to relieve the PDR-limited DDRRT from its adaptive exploration.

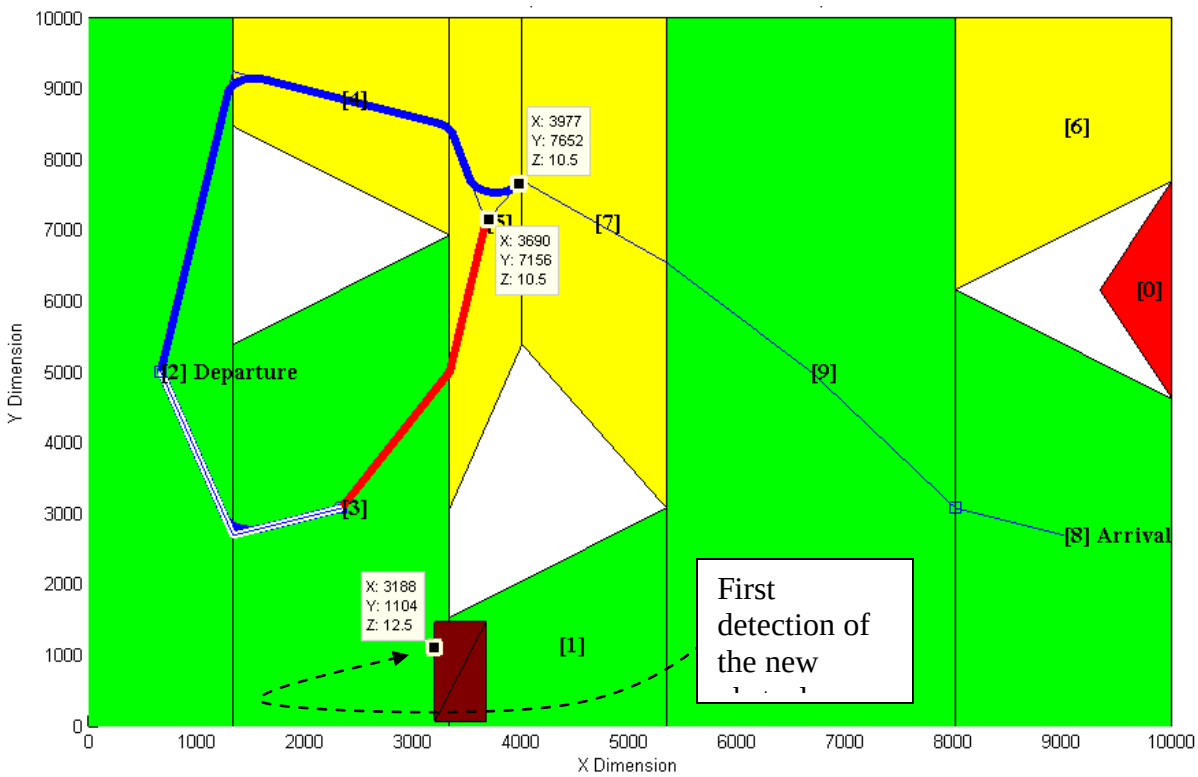


Figure 45 - PDR-Limited DDRRT Using the STA Network for Rerouting - Showing Two Alternative Routes

Figure 45 shows a PDR-limited DDRRT following the first two waypoints before (two flying legs in white) before encountering the new obstacle around waypoint 3. Instead of blindly adjusting the biasing of the PDR, the DDRRT consults the STA network to determine if alternative routes can be used to reroute the DDRRT progression. The last safe position reached by the DDRRT is the centroid of cell [3], corresponding to its second branch or flying leg. But the DDRRT detects the new red obstacle in the same cell. This fact is not enough to conclude that cell 3 must be discarded from potential alternative routes. What can be concluded however is that an obstacle was detected while the PDR-limited DDRRT was following the cell sequence [3 to 1]. The STA network suggests three alternative routes:

- [3 1 9 8], discarded
- [3 2 4 5 7 9 8], in blue
- [3 5 7 9 8], in red

The blue trajectory involves a 180° change in heading, while this change in heading is much less pronounced for the red trajectory. But both suggested cells path cannot be converted completely into Dubins trajectory. The reason for this inconvenience is that the Dubins curves generated have their waypoints going through the transformation in Figure 28 and then the circle centers are checked against condition (4.2.4). This condition is broken for the blue trajectory at waypoint number 7, because de distance between the corresponding Dubins centers of transformed waypoints 7 and 8 is 255.60 units while twice the turning radius is 499.72 units, which violates condition (4.2.4). For the same reason, the red trajectory generated using Dubins paths is broken at the transformed waypoints 3 to 4. It is true that the two partial trajectories generated do not make use of the trajectory simplification given by Algorithm 4. **But even that that simplification, because the waypoints used for the trajectories are the result of a trapezoidal decomposition, the Dubins curve generation is never guaranteed**. **This limitation is explored and resolved in the following sections**. Either by using the blue or the red trajectory, the DDRRT can make use of the STA network to reach 5. From cell 5 to the final goal in cell 8, three options are possible:

1. try to reach the goal located at cell 8 via a PDR and the STA network
2. try to reach the goal located at cell 8 using a SDR
3. try to bridge the last point reached by the trajectory with the centroid of cell [5], and then attempt to use the STA network to generate a Dubins trajectory from cell [5] to cell [8]

The red obstacle encountered in the vicinity of original waypoint 3 shows that the STA network used is obsolete, but the alternative routes allow the PDR offered by the STA network to get closer to the goal. The red and blue curves in Figure 45 are trajectories. They show a continuous evolution of a UAS position between postures. But at the DDRRT level, those trajectories are just waypoints to follow by the PDR-limited DDRRT. A legitimate question is thus to doubt of the usefulness of the SDR, which is made available through the trapezoidal decomposition. The PDR allows performing exploration at the cell level. The red and blue curves in Figure 45 are feasible trajectories "a priory", based on an STA network which has proven to be obsolete. Thus, the blue and red trajectories must only be seen as "options" but cannot be trusted blindly. The purpose Figure 45 is to show the potential offered by the STA Network. But in order to proceed with caution with the potential paths, it is important to develop awareness, so while the PDR allows fast reactive goal biasing, the SDR allows refining the exploration of the trajectories suggested by the STA network.

The red trajectory of Figure 45 lies within cell [3] and cell [5], which represents 17.75 percent of the total trapezoidal decomposition area which is 91, 024, 387 square units. The DDRRT is given 2000 seeds to explore the world shown in Figure 45. For space coverage, this means that after meeting the red obstacle, three seeds have been consumed and that 17.75 percent of 1997 or 354 seeds can be dedicated to exploring cells [3] and cell [5] with respect to the percentage they represent with respect to the current altitude layer.

The result of this spatial coverage between cell [3] and cell [5] is shown in the next figure.

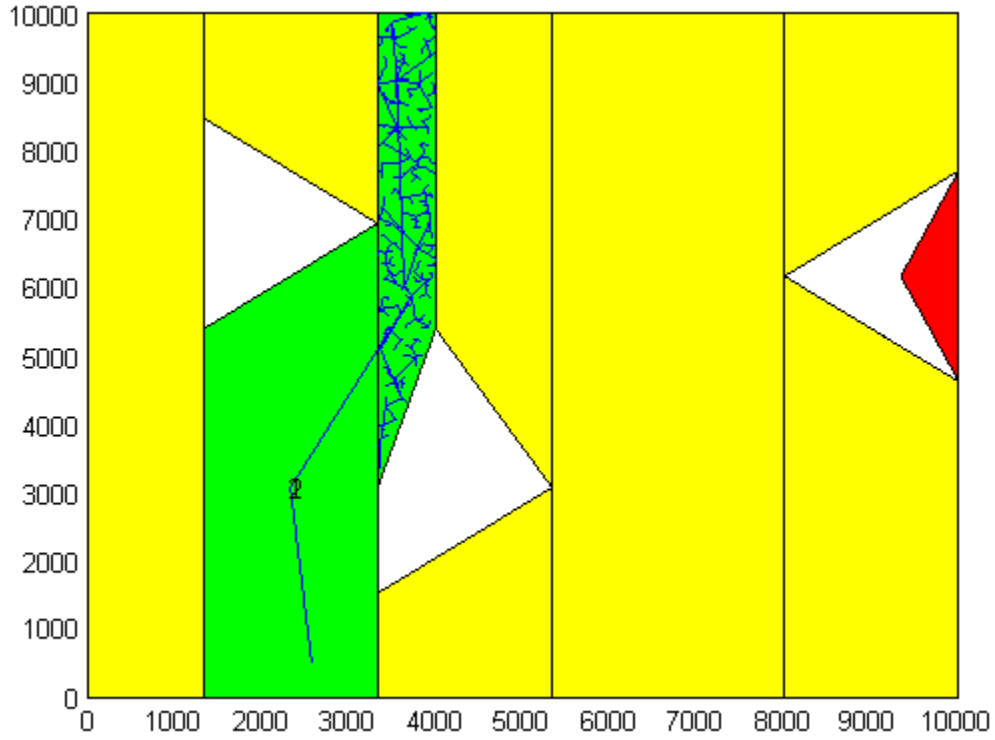


Figure 46 - DDRRT SDR Proportional Area Coverage for Surrounding Awareness

Figure 46 shows a spatial dispersion reduction for the cell sequence containing the red trajectory of Figure 45. Because the red trajectory in Figure 46 is only assumed to be safe, its corresponding cell sequence must be further explored in order to promote threat awareness. A philosophical question remains: why working with a limited amount of seeds proportional to the percentage of area of cells [3] and [5]? **The spatial dispersion reduction allows targeted coverage which balances full goal biasing (narrow exploration) with surrounding awareness (SDR)**. A full example explaining the contribution of both can be found in the next section.

V – 2 – 4 – A Practical Example of Real-Time Threat Avoidance Using the DDRRT

The following 6 figures show a practical example of the DDRRT encountering a red obstacle and using PDR and then SDR to provide local surroundings awareness before failing to breach from cell 3 into cell 1. The failure is declared when the maximum number of branches bounded by cell 3 has been deployed. This number is calculated by summing the areas of cells 3 and 1 and calculating the corresponding percentage they represent with respect to area of the whole network. The calculated percentage is 18.6% and 18.6% of 2000 samples is 371.

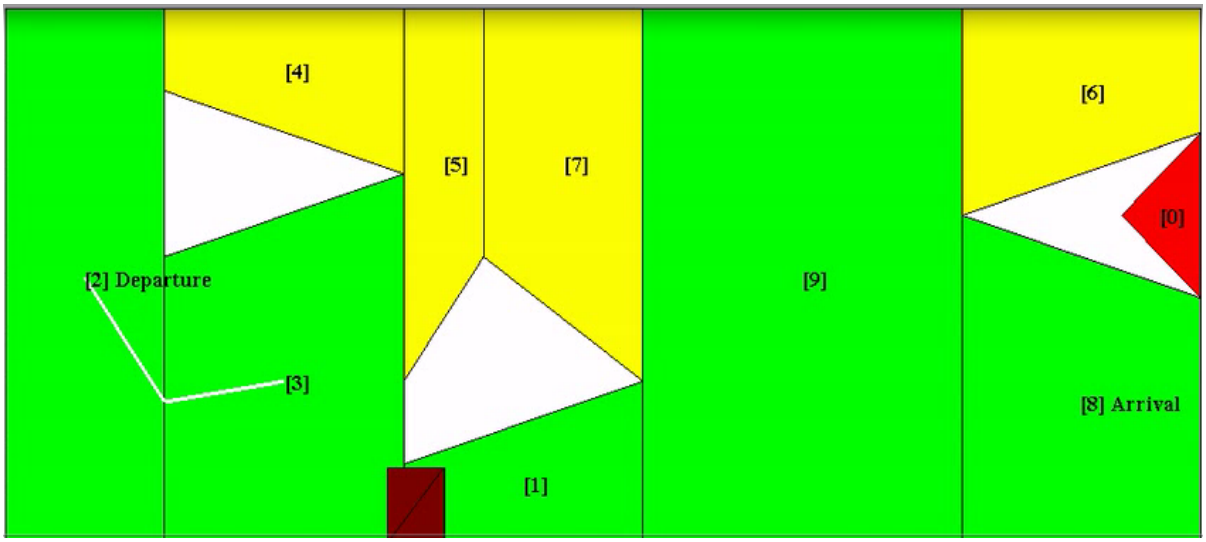


Figure 47 - The UAS Follows Two Flying Legs Designed During Offline Trajectory Planning

Figure 47 shows the first two initial flying of an offline trajectory designed to follow the cells in green. In the next figure an attempt is made for the DDRRT to follow a flying leg from the centroid of cell 3 to the midpoint between cell 3 and cell 1, but the red obstacle of Figure 47 makes it impossible.

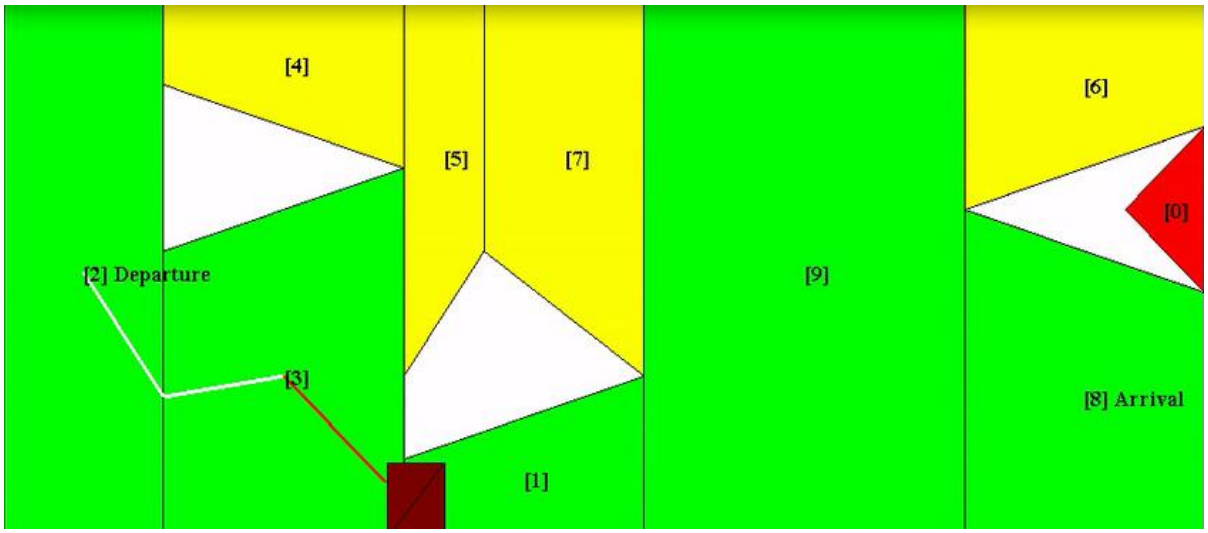


Figure 48 - The UAS Encounters a Red Obstacle Between Cell 3 and Cell 1.

In Figure 48, the DDRRT encounters an obstacle, as previously explained. This is shown by a red segment deployed to mark the corresponding collision. The next figure shows the progression of the PDR of the DDRRT.

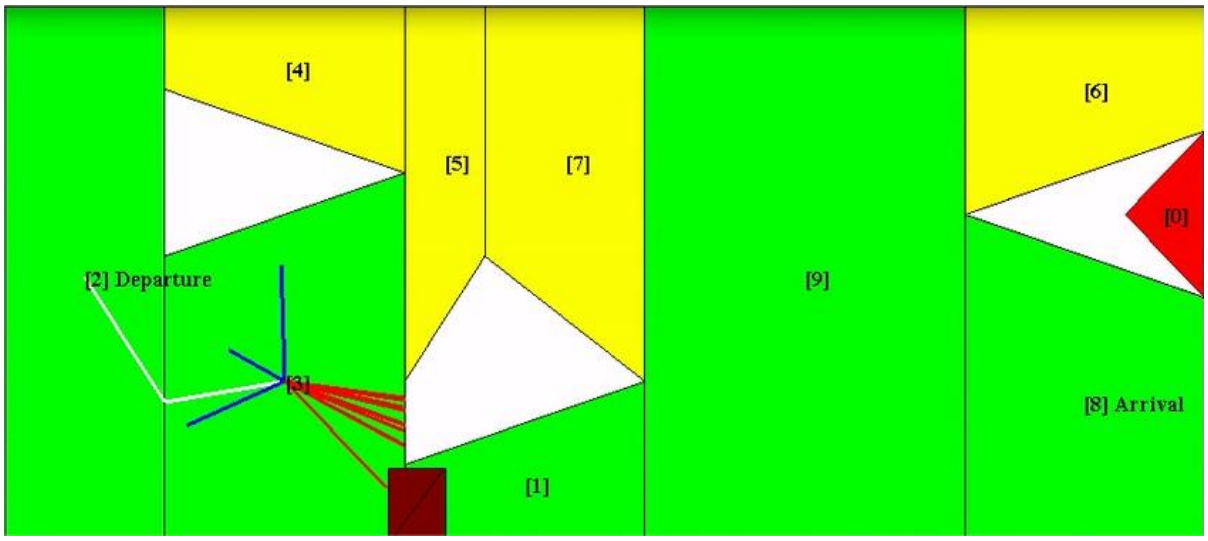


Figure 49 - The DDRRT In Decreasing PDR Mode

Figure 49 shows the multiplication of red segment (collisions) and the deployment of blue segments which shows that the goal biasing has decreased and the DDRRT exploration has started to broaden.

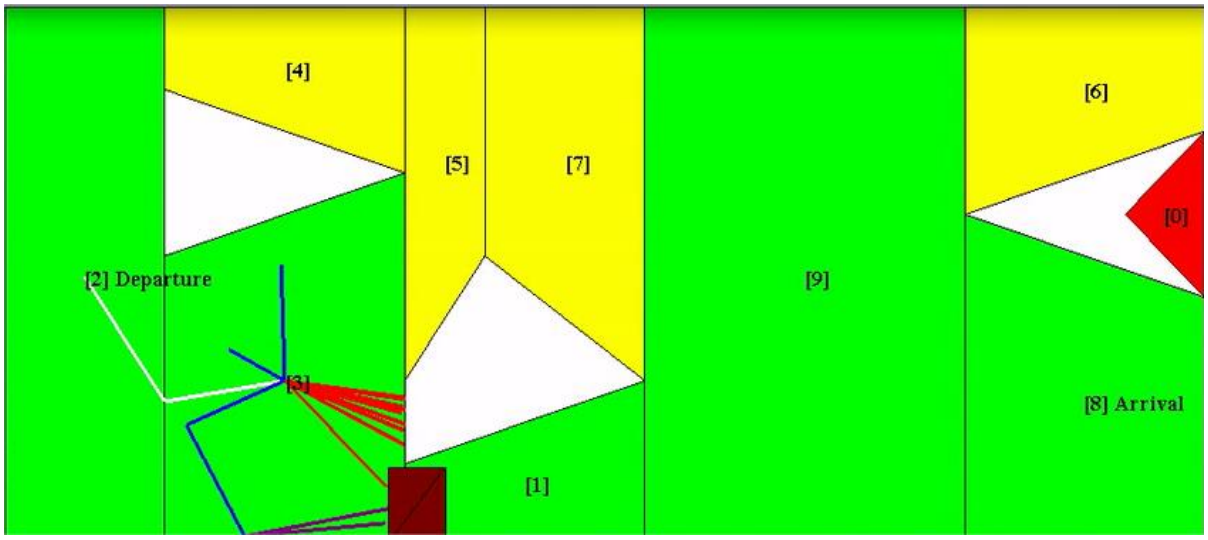


Figure 50 - The PDR Goal Biasing Has Decreased to Fall Below 50%.

In Figure 50, the PDR has decreased below 50%. Collisions with the red obstacle are now marked in purple in order to show the critical decreasing in biasing. As the 50% limit is what trigger the DDRRT to switch to SDR mode only, the next figure shows the evolution of the DDRRT for the remaining of the 371 branches employed to gain perception of the threat between cell 3 and cell 1.

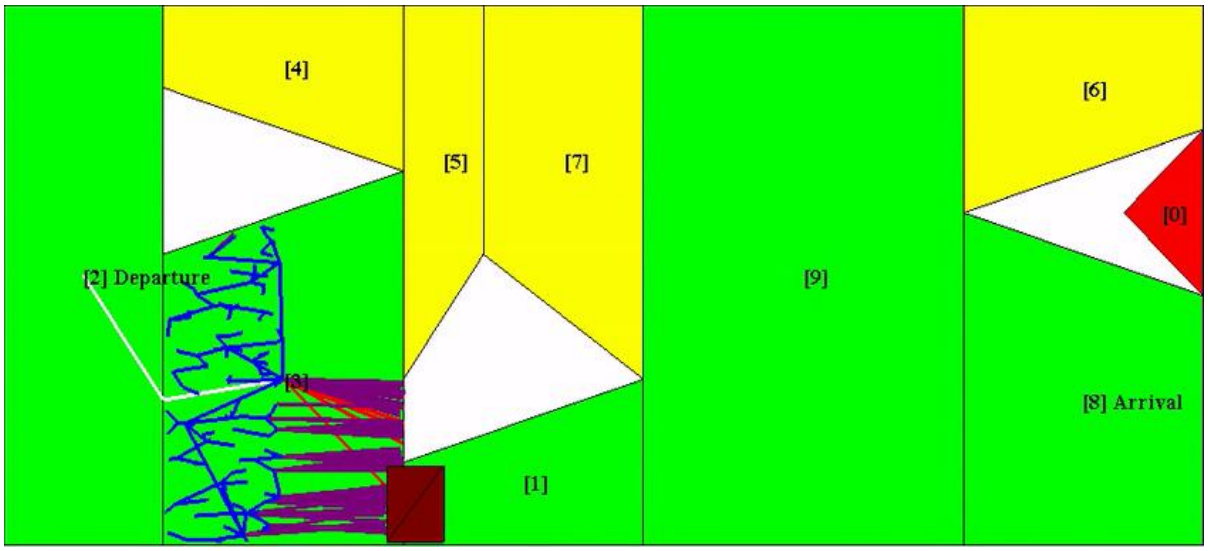


Figure 51 - The DDRRT Is In SDR Mode Only, Developing 371 Branches Out of the 2000 Possible.

Figure 51 shows the failure for the joint efforts of PDR and SDR to find an alternative route between cell 3 and cell 1 while remaining in one of the two cells. As a result, the DDRRT requests the STA network to provide it with an alternate route, which is shown next.

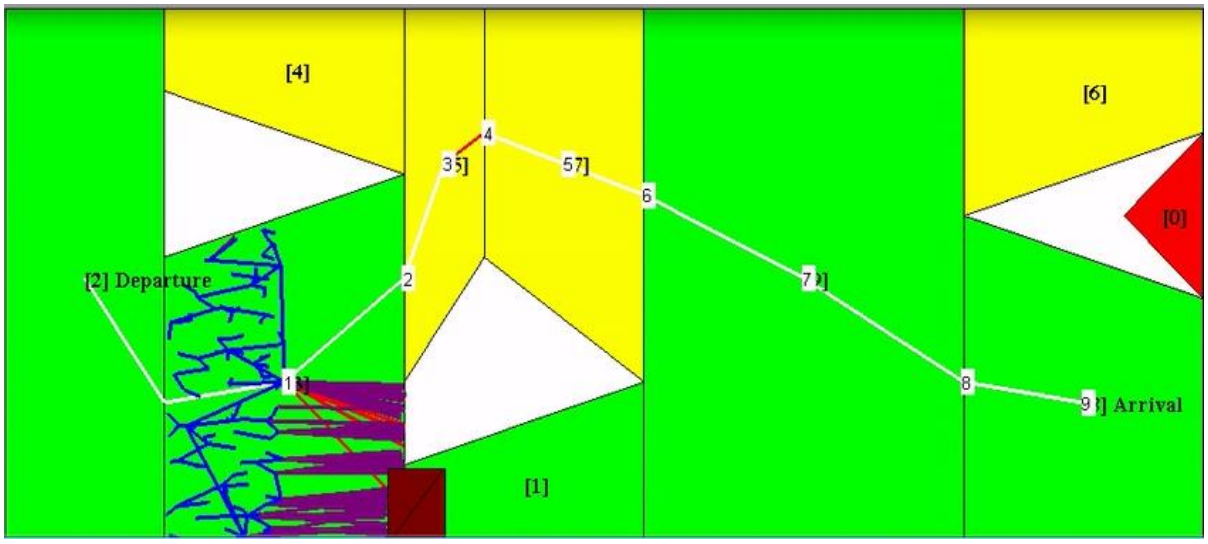


Figure 52 - The DDRRT Has Spent 371 Branches Trying to Breach Safely Into Cell 1,

Figure 52 shows that the DDRRT has spent the totality of its allowed 371 branches to breach from cell 3 to cell 1, and that its time for the STA network to provide rerouting. So the STA

network provides an alternative route using cells 3, 5, 7, 9, and 8. As the Dubins condition is broken between waypoints 3 and 4, a system of ODE is integrated instead (red segment).

As seen in Figure 45, the STA network sometimes fails to provide a complete Dubins Trajectory because the spacing between two Dubins circles in lesser than twice the minimum turning radius. To solve this problem, two postures with different headings can be connected by the following system of ordinary differential equations:

$$\begin{cases} u_1 & = & V_t \\ u_2 & = & \phi \end{cases}$$

$$\begin{cases} \dot{x} & = & u_1 \cos\psi \\ \dot{y} & = & u_1 \sin\psi \\ \dot{\psi} & = & \dfrac{g \tan u_2}{u_1} \end{cases} \tag{5.2.1}$$

In order to solve equation (5.2.1), a fixed-step Runge-Kutta ODE solver is used. This solver is then applied to the following algorithm.

X_1 = initial posture
X_2 = final posture
Initialize $d_{\min} = \infty$
$U_{best} = \begin{bmatrix} V_t & \phi \end{bmatrix}_1$ (first input vector from all the available ones belonging to the set of lateral inputs)
For all input vectors $U \equiv \begin{bmatrix} V_t & \phi \end{bmatrix}$

- t_{guess}=Guess time needed to go from X_1 to X_2 at V_t
- $t_{integration} = 10 * t_{guess}$
- Integrate the ODE given by equation (5.2.1) such that the integration process is stopped when the integrated vector x is such that $\left\| d\left(x, x_2\right) \right\| \le \varepsilon$
- For the final value of the integration process, if $\left\| d\left(x_{achieved}, x_2\right) \right\| \le d_{\min}$
  - $d_{\min} = \left\| d\left(x_{achieved}, x_2\right) \right\|$
  - $U_{best} = \begin{bmatrix} V_t & \phi \end{bmatrix}$
- endIf

endFor

Algorithm 5 – Algorithm to Generate a Feasible Lateral Maneuver Between Two State Vectors (x,y,φ)

Algorithm 5 shows how “bridging” two postures in a same STA network can be achieved when the connection cannot be made by via Dubins trajectories. Equation (5.2.1) is general, but for this research, V_t is assumed constant, so the only variable inputs are the different achievable bank angles. Using the Euclidian distance between the two postures gives an estimate of the time needed to join them at constant speed. In turn, this estimate is taken one order of magnitude higher to guarantee that enough integration time is given to the Runge-Kutta solver to reach the final posture X_2 close to a given epsilon.

Figures 47 through 52 illustrate how the DDRRT unlike other RRT algorithm stays on the same side of an obstacle and try progressively to develop awareness of its surrounding before requesting for the STA network to provide it with an alternate route.

V – 2 – 5 – The DDRRT Algorithmic Strategy When the Current Altitude Layer Cannot Provide Planar Rerouting

The previous paragraph establishes that the DDRRT follows a predefined Dubins trajectory when available, and this trajectory following is achieved through the PDR of the DDRRT until an obstacle is encountered. If a threat is identified while the PDR is following the waypoints of the original Dubins trajectory, the STA network is then interrogated to find alternative trajectories for the same altitude layer. It is however possible for the current STA network to not have any other trajectory to follow for the current altitude layer.

In case the STA network related to the current altitude layer does not provide alternative trajectories for a UAS trapped in its current altitude layer, layers of higher altitudes are tested for connectivity of their cells with those of the current altitude layer. The next figure shows a superposition of two altitude layers.

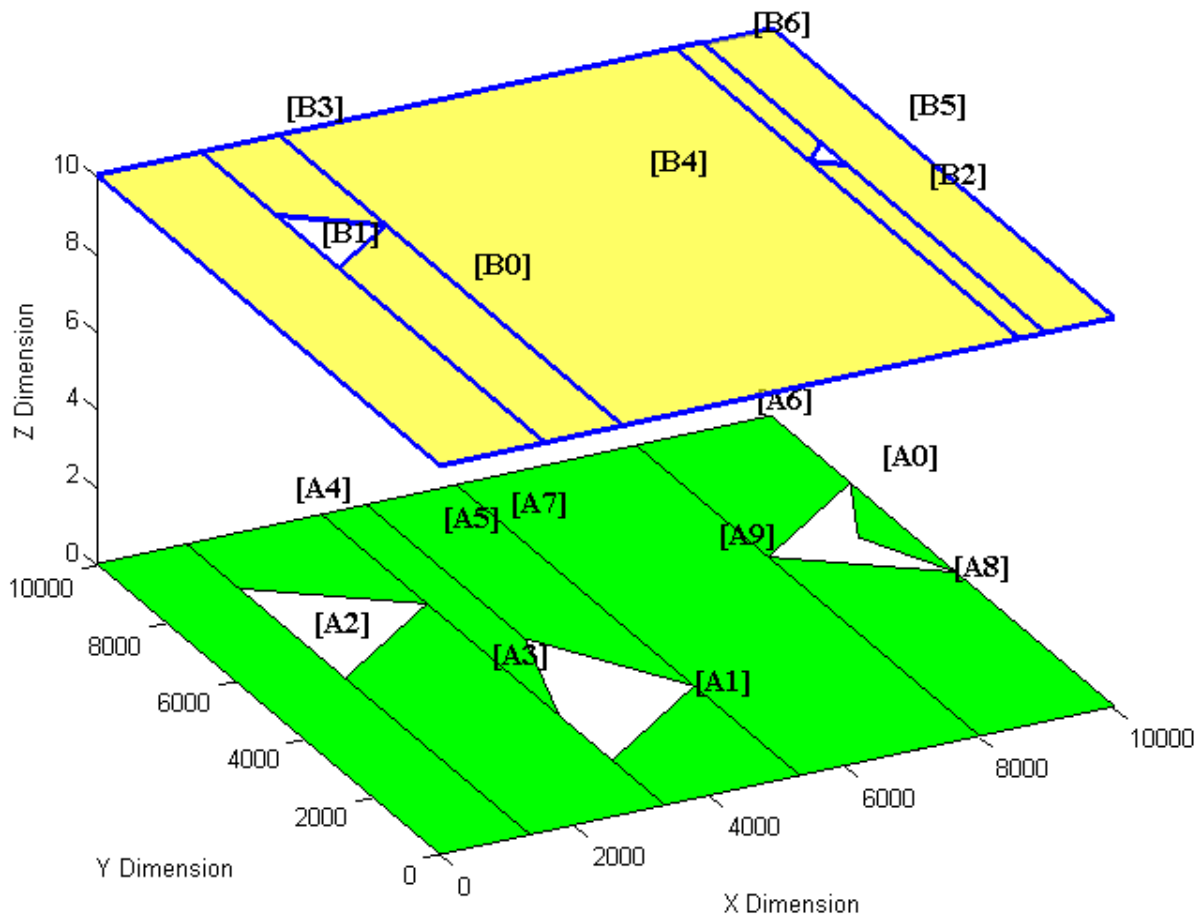


Figure 53 - Two STA Networks Used for Inter-Layer Path Queries

Figure 53 shows tow altitude layers. The green altitude layer has been decomposed in 10 cells numbered A0 through A9. The yellow altitude layer is located 10 units of altitude over the zero altitude layer, and contains 7 trapezoidal cells. The two altitude layers correspond to two different STA network, which must be combined in order to provide inter-layer path queries.

In order to detected connectivity between two cells of two different layers polygon clipping tests using the MATLAB implementation of the Sutherland–Hodgman algorithm are performed.

Those tests allow determining if two polygons are overlapping. The next figure shows an example of this overlapping test for the cells A3 (green layer) and B0 (yellow layer).

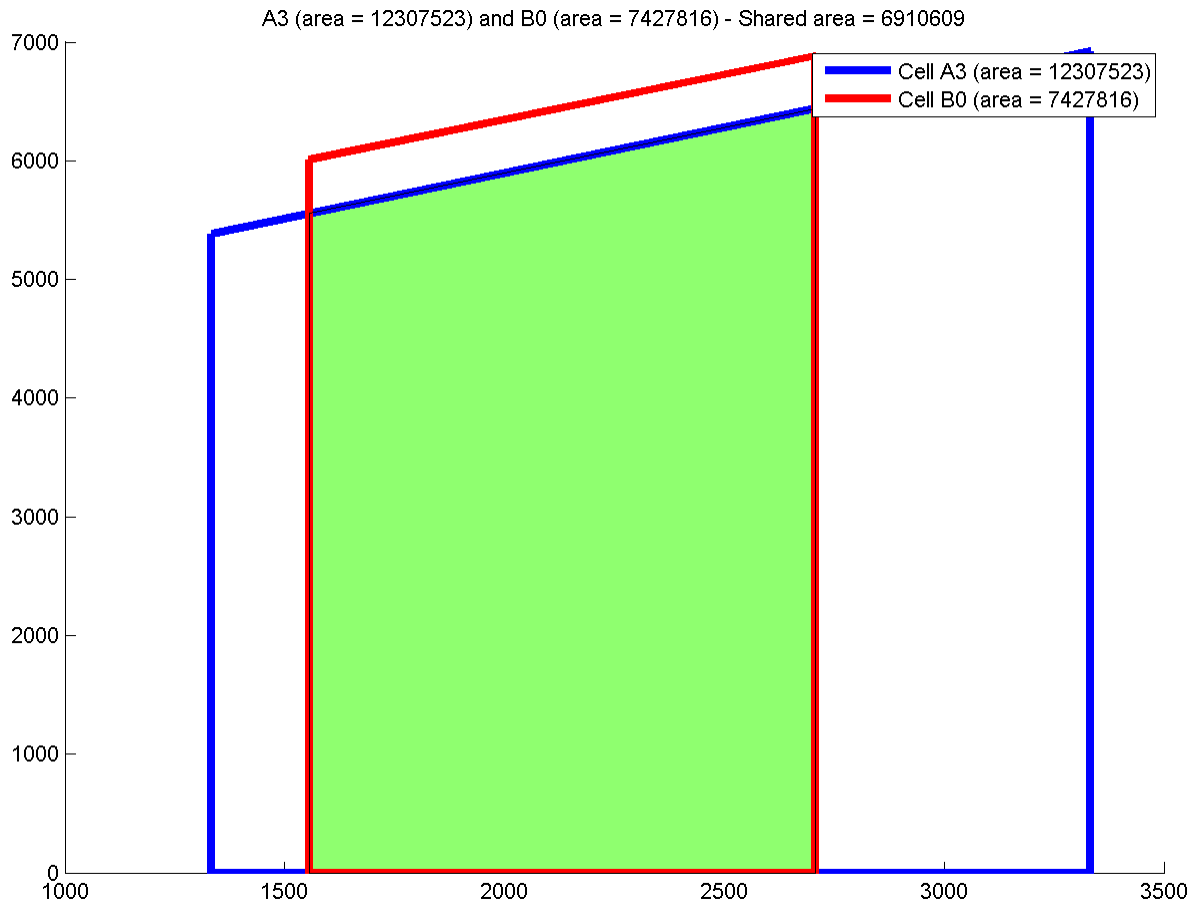


Figure 54 - Polygon Clipping Test Performed Between Two Cells of Two different STA Networks

Figure 54 shows the result of clipping the polygon of cell A3 with the contour of cell B0 which is the green surface. Because the clipping of two trapezes is a trapeze, the computation of the clipped polygon (green surface) is always possible. If the area of the clipped polygon is non-zero, then there is a potential connection between the two cells of the two different networks. In the example of Figure 53, there are 10 trapezoidal cells for the green altitude layer which means that a “10 by 10” adjacency matrix is holding the corresponding connectivity information. The

same logic applies to the yellow altitude layer of Figure 53, whose corresponding STA network contains a “7 by 7” adjacency matrix. Using the inter-layer connectivity tests between the green and yellow layers allows to create a “17 by 17” adjacency matrix. This augmented 17 by 17 adjacency matrix is build using the following formula:

$$M_{\text{layer a to layer b}} = \begin{bmatrix} A & C \\ C^T & B \end{bmatrix} \tag{5.2.2}$$

The 10 cells of the green layer make the A matrix of equation (5.2.2). The 7 cells of the yellow layer make the B matrix of equation (5.2.2). In order to form the C matrix in equation (5.2.2), the 7 cells of layer LB (yellow) are numbered after the 10 cells of layer LA (green), which in the case of Figure 53 means that accessing the connectivity test result for B0 and A0 is made possible by looking up the value of $M_{\text{layer a to layer b}}(11,1) = C(1,1) = M_{\text{layer a to layer b}}(1,11) = C^T(1,1)$. In the example of Figure 53, the polygon overlapping tests gives the following matrix (C matrix).

$$\begin{array}{cc} & \text{Yellow cells Ids} \\ & \begin{pmatrix} 0 & 1 & 2 & 3 & 4 & 5 & 6 \end{pmatrix} \\ \text{Green cells Ids} \begin{pmatrix} 0 \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7 \\ 8 \\ 9 \end{pmatrix} & \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 \end{bmatrix} \end{array}$$

The concatenation of the A (green layer), B (yellow layer), C and C^T (cross-layer) allows building $M_{\text{layer a to layer b}}$ which in turn permits inter-layer connectivity queries. But those queries only indicate if trapezoidal cells of two superposed altitude layers are overlapping. **The queries themselves do not contain longitudinal maneuver feasibility information**. In order to determine longitudinal flyability between cross-layer cells, the two cells must be connected via a climbing cone test. For a climb maneuver, the cone's apex is located at the UAS' current position within the lower layer. Determining the flyability between two cells of different layers is performed following the calculations showed in the next figure.

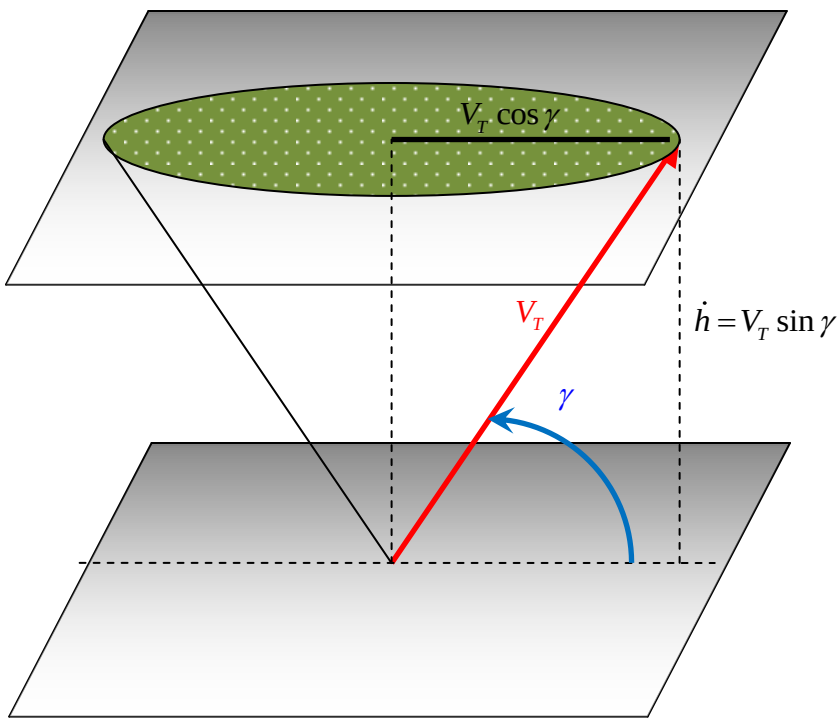


Figure 55 - Longitudinal Maneuver Cone

Figure 55 illustrates how the longitudinal maneuver cone is used in conjunction with formula (5.2.2) to determine if two cells of different layers can be joined for a UAS flying at airspeed V_T and for a given flight path angle. Formula (5.2.2) only informs if two cells overlap, but it is the maneuver cone of Figure 55 which makes it possible to determine if a flyable trajectory exists

between the two cells from two different layers. For a climb maneuver between two layers, the base of the cone must be comprised within the boundaries of the higher altitude cells. But in addition to the connection between cells of different layers, the inter-layers cell sequences must support additional selection criteria to discard cells sequence which are unsafe for the UAS to follow. The Depth First Search (DFS) algorithm is used to list all paths between two trapezoidal cells. When the DFS is applied to two or more altitude layers, the numbers of connected cell sequence increases significantly. This is because the DFS algorithm does not account for flyability, and therefore even sequences involving hazardous successions of climbs and descents are included.

But perhaps the most important question is: if the DDRRT double dispersion reduction is the result of a clever use of the STA network, how can it work in the absence of an elevation map? The next paragraph suggests a method to create an STA network in the absence of one by modeling an obstacle detector.

V – 3 – 6 – Adding Logic to Threat Detection In Order to Create an STA Network When No Elevation Data is Available

As it has been shown previously, some knowledge even coarse of a flying zone by a UAS allows the DDRRT to make substantial savings in the number of seeds employed to find a feasible path. By making clever use of trapezoidal maps embedded in STA network, the DDRRT can trigger a punctual dispersion reduction exploration, while the corresponding trapezoidal cells are used for a broader type of dispersion reduction, the spatial dispersion reduction.

In other words, while the STA network suggest potential trajectories to the PDR of the DDRRT, the corresponding cells to those trajectories can be used with the SDR of the DDRRT to promote potential threat awareness for the immediate surroundings. But since those benefits are the results of the STA network, it can be unfair to compare the DDRRT to those other RRT variations which do not have an STA network available. The present section provides a solution to DDRRT when no STA networks are available.

STA networks are collections of trapezoidal cells with connectivity information, relating obstacle-free trapezoidal zones. When there are no elevation maps, there is no access to trapezoidal decomposition, and hence fast trajectory re-planning is “a priori” impossible. However, the present section suggests a method for creating a trapezoidal decomposition out of collision points computed during PDR DDRRT growing.

In previous paragraphs punctual contact tests are performed between a sphere and the terrain’s obstacles. By keeping track of those contact points when they occur, it is possible to reconstruct a coarse topology of the terrain being explored. However, gathering contact points resulting from a sphere to obstacle contact tests is inefficient. A solution to gather more contact points per

obstacle contacts tests is to model a detection volume which allows gathering multiple contact points per tests. To this end a virtual radar sheet of with a 210 degree field of view, +/- 15 degree in elevation, and 0.5 nautical mile range is used. Such detection sheet is presented in the next figure.

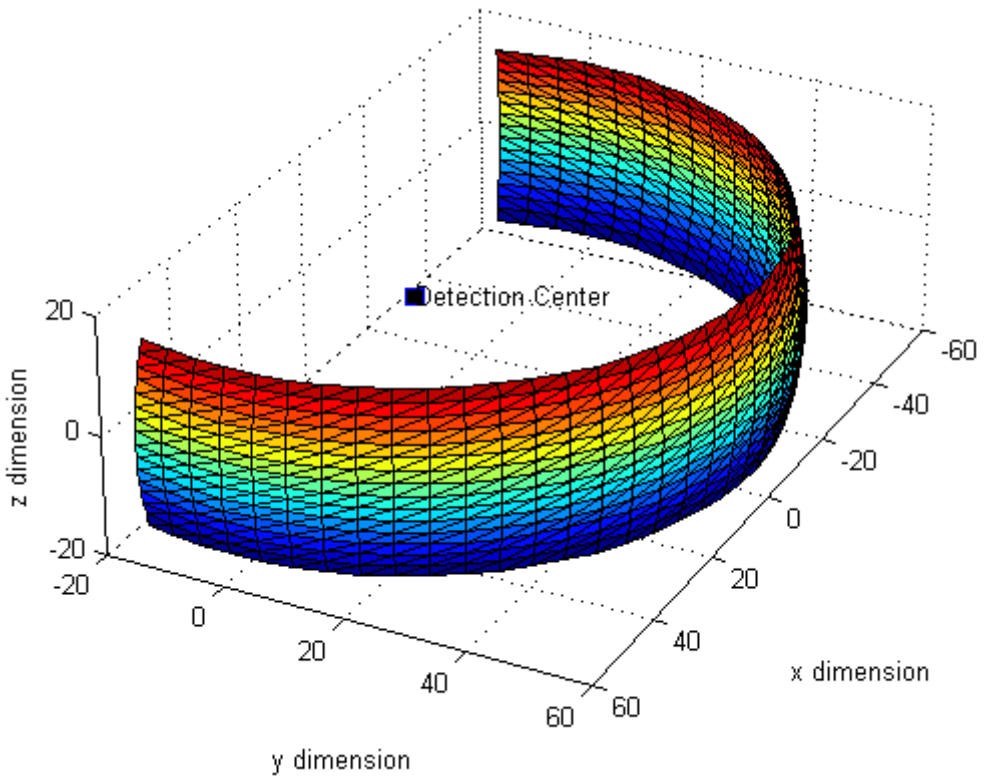


Figure 56 - A Detection Sheet

As can be seen in Figure 56, using the a detection sheet offers more points for contacts tests with potential obstacles or moving threat than the collision sphere enveloping the UAS. In turns each point of the detection sheet in contact with obstacles can be gathered in used to build bounding boxes, which in turns are used for trapezoidal decompositions.

VI – Closure

VI – 1 – Summary

The motivation for the research presented in this dissertation is to provide a solution to the problem of non-cooperative reactive mid-air threat avoidance for fixed-wing unmanned aerial systems. This is achieved by solving the problem in two sequential phases.

The first phase consists of taking advantage of available terrain elevation maps which can be accessed from the internet. Because those maps contain too many data points for path planning purposes, they are processed by a computer program which sequentially performs the following tasks:

A1. Create altitude layers

A2. Extract obstacle contours

A3. Simplify obstacle contours

A4. Perform vertical trapezoidal decomposition on altitude layers from which obstacles have been removed in order to produce a stratified, tessellated two-dimensional obstacle-free trapezoidal flying zones

A5. Extrude the altitude layer cells so they can be included in realistic threat avoidance of obstacles in 3D

A6. Adds connectivity information for the trapezoidal cells so the depth-first search can list all cell sequences between two cells

A7. Transform a decomposed cell sequence into a sequence of geometric waypoints based on the cell centroids and the shared edge pinch points

A8. Transform the geometric waypoints into flyable waypoints by using an enhanced Dubins algorithm which respects flying leg directions when possible, or by integrating a system of ODEs when the first algorithm cannot be applied when the Dubins condition is not met.

The AWMG which sequentially goes from steps A1 through A8 produces a Synthetic Terrain Avoidance network: enhanced elevation terrain data with embedded intelligence for rerouting. The second phase involves applying the DDRRT algorithm to the STA network generated during the first phase. The purpose of applying the DDRRT algorithm is to force an inherently unfocused and "ubiquitous" original algorithm (the RRT) to explore mid – air threat from one orientation at a time. To do so, the DDRRT performs the following tasks at the beginning of a real-time mid – air threat avoidance:

B1. The DDRRT starts with a 100% Punctual Dispersion Reduction (PDR) toward the next waypoint in its list.

B2. While approaching the next waypoint, if an obstacle is encountered, the PDR adapts its biasing until potentially reaching 50% of goal biasing.

B3. If the PDR reaches 50% goal biasing or below, the DDRRT switches to pure Spatial Dispersion Reduction (SDR) in order to decrease waypoint focus and develop awareness of its immediate surroundings, while staying within the confines of the last cell visited.

B4. Once in SDR mode, the DDRRT uses a calculated maximum number of branches to find a path going around the detected threat. This maximum is calculated based on the area of the next two cells in the original cell sequence and the percentage their combination represents with respect to the total sum of all the cells of the STA network.

B5. If the SDR reaches the maximum number of cells it can use to find an alternative route around the obstacle, the implication is that the SDR failed.

B6. If the SDR fails, the DDRRT interrogates the STA network to provide an alternative route. If no alternative route can be found for the current layer, the layers above and below are combined with the STA network in use in order to find a trans-layer alternative path.

VI – 2 – Conclusion

Based on the design choices for the two phases of mid-air threat avoidance previously summarized, the following conclusions can be drawn:

- The present research uses a fixed-tolerance Douglas-Peucker algorithm which was successful at converting elevation point clouds to actual vectorized closed-contour obstacle areas for both the Grand Canyon map and the academic example used throughout the document.
- The 1D vertical trapezoidal decomposition was successfully used on extruded 2D layers in order to enable point location capability for all the Synthetic Terrain Avoidance (STA) networks generated for both the Grand Canyon map and the academic example used throughout the document.
- A combination of Punctual Dispersion Reduction (PDR) and Spatial Dispersion Reduction (SDR) successfully works and in a complimentary way: the PDR is suited for smaller obstacles, but when those obstacles are too big, its linear punctual targeting decays and gives way to the cell-bounded spatial dispersion reduction.
- The example of part V shows that the DRRT successfully handles a safe mid-air dynamic rerouting by first following a flyable trajectory before detecting an unanticipated obstacle and developing targeted threat awareness using PDR and PDR for decision making regarding a potential rerouting. Because the DDRRT exploration is double-targeted, the DDRRT develops a threat awareness which is proportional to the area of the last visited cell. In doing so, the DDRRT allows to create an upper bound for the number of exploration points to be used in order to develop threat awareness regarding the immediate vicinity of the detected obstacle. Upon reaching

such bound, the DDRRT interrogates the Synthetic Terrain Avoidance (STA) network to investigate a potential reroute, which allows the DDRRT to perform dynamic trajectory evolution through STA network rerouting.

VI – 3 – Recommendations and Future Work

The following recommendations can be made regarding the maintenance and improvement of the AWMG:

A1. The Geographic Data Abstraction Layer (GDAL) is the software library used to decode elevation maps in different formats. It is recommended to use GDAL directly, as it is updated more frequently than the version shipped with MATLAB.

A2. The fixed-tolerance Douglas-Peucker used in this research is prone to generate contours that are too conservative for obstacles. While this forced conservatism can be viewed as an increase in safety for offline trajectory generations, the drawback is that it can obliterate previously existing free zones. Future work should investigate other contour approximation methods. A suggestion is to consider supplying positional accuracy with respect to the original contour in order to calculate the corresponding tolerance for the Douglas-Peucker algorithm.

A3. As mentioned in the conclusion, the trapezoidal decomposition used throughout this research is vertical only. This limitation has the inconvenience of decreasing the choices of available paths. A cross-trapezoidal decomposition is recommended as this would not require much more effort and would allow more selection.

A4. In its current state, cell sequences are transformed into waypoint lists through cell centroids and edge pinch points. But to further the obstacle avoidance aspect of the present work, it would be worth discretizing each cell border in order to apply Voronoi diagrams to the corresponding points, as done in previous research. This would allow the convenient use of trapezoidal cells (convex objects) while increasing path safety.

A5. Future work should consider investigating Euler spirals and adapt them to the current Dubins path methodology. The advantage of using Euler spirals is that the radius of curvature can vary continuously, avoiding heading rate discontinuities, which is the main inconvenience of the Dubins circle.

A6. When the Dubins path method is not applicable during trajectory (re)planning, a system of ODEs is solved iteratively for a range of achievable bank angles and then the solution the closest to the goal is retained. The retained solution is then tested for collisions against the existing geometry. While such approach works, sequential integration for the different achievable bank angles can be conducted in parallel, by reprogramming the available code in Compute Unified Device Architecture (CUDA), a C-based language which allows modern NVIDIA GPUs to be programmed in order to perform tasks in parallel. But the recommendation for future work is to also embed collision test code so that the segments generated during the integration can be tested for collision during the integration. The immediate recommendation to perform integration and collision detection simultaneously is to replace the Optimized Collision Detection (Opcode) library currently used for collision test with Bullet, another collision test library. The advantage of using Bullet over Opcode is that Bullet can be programmed in parallel, and thus used with CUDA, which is not the case with Opcode.

The following recommendations can be made regarding the maintenance and improvement of the DDRRT code:

B1. Never use an exact geometry package to perform collision tests in the DDRRT code. Early versions of the DDRRT took 40 minutes to solve the collision scenario shown in the example of part V. The collision tests were 2D only, involving exact

(analytical) circle-segment intersections tests, and yet several minutes were required to perform the tests with the Computational Geometry Algorithm Library (CGAL). When the same code was transferred from CGAL to Opcode, even 3D tests took less than a second. Thus, the recommendation for future work on the DDRRT is to use a dedicated collision detection library used in the video game industry (Opcode, Rapid, PQP, PhysX, and Bullet), but never a library only handling analytical formulations for intersections tests, as those are too slow, even for the sake of simulation.

B2. The Spatial Dispersion Reduction (SDR) of the DDRRT behaves like a regular RRT except that it is guaranteed to be bounded by a trapezoidal cell. But even within a given cell, the PDR of the DDRRT is greedy, and tends to use as many cells as allowed to explore its surrounding cell. Future work should research additional logic to further limit the number of cells. In this research, all simulations were performed with a UAS bounding sphere of radius 50 ft. But for a given altitude, that sphere is a disc of area 7854 ft^2, so an additional notion of area covered by the disc can be used to prevent the DDRRT in SDR mode from using too many seeds while exploring a cell.

B3. Future work should also be done on improving the Punctual Dispersion Reduction (PDR) of the DDRRT. The current PDR adaptive goal biasing could be changed to account for the relative directions in which collisions are encountered. By establishing a cost function updated to reflect the most detrimental relative directions of growth, the PDR adaptive goal biasing could follow an optimal control theory approach such as the Linear Quadratic Regulator formulation.

Appendix A – The DDRRT SDR Algorithm

New_tree_growth=false
Select initial cell
For N iterations

- Draw a seed s_{rand}
- Map s_{rand} to the current trapezoidal cell using shape functions
- If a potential new tree T_{new} is scheduled
    - Select the next potential trapezoidal cell TC_{next} to sample from the current sequence of cells
    - Map s_{rand} in TC_{next} to get a state $\rightarrow x_2$
    - Find the nearest neighbor of x_2 within the current tree T $\rightarrow$ x_1
    - Try to connect x_1 to x_2 without collisions
    - If no collision occurred in the TC_{next}
        - Create a new Tree T_{new} such that its root is x2 and its 1st extension is toward x1
        - Add Tnew to the collection of existing tree and make it the current active tree for nearest neighbor search
        - Make TC_{next} the current trapezoidal cell. $TC_{current}$=TC_{next}
    - Else if a collision has occurred in TC_{next}
        - Map srand in $TC_{current}$ $\rightarrow$ x2
        - Find the nearest neighbor of x_2 within the current tree T $\rightarrow$ x_1
        - Try to connect x_1 to x_2 without collisions
    - End of actions to execute when a collision has occurred in TC_{next}
- Else if no potential new tree T_{new} is scheduled
    - Map s_{rand} in $TC_{current}$ to get a state $\rightarrow x_2$
    - Find the nearest neighbor of x_2 within the current tree T $\rightarrow$ x_1
    - Try to connect x_1 to x_2 without collisions
- End of Actions to execute if no potential new tree T_{new} is scheduled
- Check I_{seq}, index of $TC_{current}$ in the sequence of cells to determine if it has reached $I_{seq.max}$
- If $\left(I_{seq.} \geq I_{seq.\max}\right)$
    - Do not schedule a new tree growth $\rightarrow$New_tree_growth=false
- Else
    - Schedule a new tree growth $\rightarrow$ New_tree_growth=true
- End of decisions to make regarding potential new tree growth
- If current sample in the tree has reach the goal within a certain tolerance
    - Stop the iterations
- End of actions to execute when current sample has reached the goal

End of the algorithm's iteration

References

[1]Anonymous, "Piccolo Flight Management System", Technical Documentation, URL: http://www.cloudcaptech.com/Sales%20and%20Marketing%20Documents/Piccolo%20Autopilot%20System.pdf, retrieved 03/11/2011

[2] Keshmiri S., Leong E., Jager R., Hale R,, "Modeling and Simulation of the Yak-54 Scaled Unmanned Aerial Vehicle Using Parameter and System Identification", AIAA Atmospheric Flight Mechanics Conference and Exhibit 18 - 21 August 2008, Honolulu, Hawaii

[3]Leong E., Keshmiri, S., Jager R., "Evaluation of a COTS Autopilot and Avionics System for UAVs," AIAA Infotech Conference, Seattle, Washington, April 6-9, 2009.

[4] Anonymous, "wePilot2000 Technical Data Sheet", URL: http://www.wecontrol.ch/images/stories/documentation/wepilot2000.pdf, retrieved 04/06/10

[5] Anonymous, "Unmanned Aircraft Systems Operations in the U.S. National Airspace System", Interim Operational Approval Guidance, January 2008

[6] Anonymous, "Code of Federal Regulations part 91.1131, Right-of-Way Rules", Doc. No. 18334, 54 FR 34294, August 18 1989

[7] Clough B. T., "Metrics, Schmetrics! How The Heck Do You Determine A UAV's Autonomy Anyway?", proceedings of the 2003 Performance Metrics for Intelligent Systems (PerMIS) Workshop, Gaithersburg, MD, August 16 - 18, 2003

[8] Anonymous, "Unmanned Aerial Vehicles Roadmap 2000-2025", Office Of The Secretary Of Defense, Washington DC. April 2001

[9] Borrelli F., Subramanian D., Raghunathan A. U., Biegler L. T., "A comparison between MILP and NLP Techniques for Centralized Trajectory Planning of Multiple Unmanned Air Vehicles", American Control Conference, 2006, Volume , Issue , 14-16 June 2006 Page(s):6 pp. Digital Object Identifier 10.1109/ACC.2006.1657644

[10] Culligan K. F., "Online Trajectory Planning for UAVs Using Mixed Integer Linear Programming", Master Thesis, Massachusetts Institute of Technology, September 2006

[11] Reinl C., von Stryk O., "Optimal Control of Multi-Vehicle-Systems Under Communication Constraints Using Mixed-Integer Linear Programming", Simulation, Systems Optimization and Robotics Group, Technische Universit¨at Darmstadt, proceedings of ROBOCOMM 2007 - First International Conference on Robot Communication and Coordination (Athens, Greece, October 15 - 17, 2007)

[12] Richards A. G., How J. P., "Aircraft Trajectory Planning with Collision Avoidance using Mixed Integer Linear Programming" in Proceedings of the American Control Conference, 2002

[13] Larsen M., Beard R. W., McLain T. W., "Real-Time Trajectory Generation for Autonomous Nonlinear Flight Systems", AF02T002 Phase II Final Report, July 2006

[14] O. Khatib, “Real-time obstacle avoidance for manipulators and fast mobile robots”, International Journal of Robotics Research, vol. 5, p. 90, 1986.

[15] K. M. Passino, “Biomimicry for Optimization, Control, and Automation”, Springer-Verlag London Limited 2005

16]Scherer S., Singh S., Chamberlain L., Elgersma M., "Flying Fast and Low Among Obstacles: Methodology and Experiments"

[17] Trottenberg U., Oosterlee C., Schöller, "Multigrid", Academic Press, 2001

[18] Park J., Oh H., Tahk M., “UAV Collision Avoidance Based on Geometric Approach” SICE Annual Conference 20-22 August 2008, The University Electro-Communications, Japan

[19] Krozel J., Peters M., “Strategic Conflict Detection and Resolution for Free Flight”, proceedings of the 36th Conference on Decision & Control, December 1997, San Diego, California

[20], Schild R. "Rule Optimization for Airborne Aircraft Separation", PhD Dissertation, Institut fur Wirtschaftsmathematik, 1998

[21] Tsourdos A., White B. A., Shanmugavel M., "Cooperative Path Planning of Unmanned Aerial Vehicles", 2011 John Wiley & Sons. ISBN: 978-0-470-74129-0

[22] Choset H., Lynch K. M., Hutchinson S., Kantor G., Burgard W., Kavraki L. E., Thrun S., "Principles of Robot Motion - Theory, Algorithms, and Implementations", The MIT Press Massachusetts Institute of Technology Cambridge, Massachusetts 02142, 2005

[23] LaValle S., “Planning Algorithms”, Cambridge University Press, 2006

[24] Balakrishnan V. K., "Graph Theory - First Edition", 1997 - McGraw-Hill

[25] Dijkstra E. W., "A note on two problems in connection with graphs", Numerische Mathematik 1 - 269–271, 1959.

[26] Hart P. E., Nilsson N. J, Raphael B. "A Formal Basis for the Heuristic Determination of Minimum Cost Paths", IEEE Transactions on Systems Science and Cybernetics SSC4: pp. 100–107, 1968.

[27] Lozano-Pérez T., Wesley M. A., "An Algorithm for Planning Collision-Free Paths Among Polyhedral Obstacles", Communications of the Association for Computer Machinery (ACM), Volume 22 Issue 10, Oct. 1979

[28] Bygi M. N., Ghodsi M., "3D Visibility Graph", Proceedings of 12th Annual Conference of Computer Society of Iran, Tehran, Iran, 2007

[29] Pettersson P. O., Doherty P., "Probabilistic Roadmap Based Path Planning for an Autonomous Unmanned Aerial Vehicle", Journal of Intelligent and Fuzzy Systems, Volume 17, Number 4/2006

[30] LaValle S. M., "Rapidly-Exploring Random Trees: A New Tool for Path Planning", IEEE Transactions on Reliability, 1998

[31] Ogata K., "Modern Control Engineering- Third Edition", 1997, Prentice-Hall

[32] Philips W. F. "Mechanics of Flight – Second Edition", 2010, Wiley & Sons

[33] Marchman J. F. III, Professor of Aerospace & Ocean Engineering, Virginia Polytechnic Institute & State University - "Introductory Aircraft Performance", retrieved September 2012. URL: http://www.scribd.com/doc/37041057/Intro-Aircraft-Performance-Text

[34] Geo COMMUNITY website. URL: http://data.geocomm.com Retrieved July 4th 2011

[35] MICRODEM website. URL: http://www.usna.edu/Users/oceano/pguth/website/microdem/microdem.htm Retrieved July 4th 2011

[36] Douglas D., Peucker T., "Algorithms for the reduction of the number of points required to represent a digitized line or its caricature", The Canadian Cartographer 10(2), 112–122 (1973)

[37] Hales, T. C., "The Jordan curve theorem, formally and informally", The American Mathematical Monthly, (2007a), 114 (10): 882–894

[38] Black, P. E., "all simple paths", in Dictionary of Algorithms and Data Structures [online], U.S. National Institute of Standards and Technology. 2 September 2008. (accessed 12/29/2012) Available from: http://www.nist.gov/dads/HTML/allSimplePaths.html

[39] Dubins, L.E., "On Curves of Minimal Length with a Constraint on Average Curvature, and with Prescribed Initial and Terminal Positions and Tangents", July 1957, American Journal of Mathematics 79 (3): 497–516.

[40] Hota S., Ghose D., "A Modified Dubins Method for Optimal Path Planning of a Miniature Air Vehicle Converging to a Straight Line Path", 2009 American Control Conference, Hyatt Regency Riverfront, St. Louis, MO, USA, June 10-12, 2009

[41] Lindemann, S., LaValle S., “Incrementally Reducing Dispersion by Increasing Voronoi Bias in RRTs”, in proceedings of the IEEE International Conference on Robotics and Automation, 2004, New Orleans, LA, USA

[42] Clifton M., Gavin P., Kwok N., Liu D., Wang D., "Evaluating Performance of Multiple RRTs", proceedings of the Mechtronic and Embedded Systems and Applications conference, Beijin, 2008

[43] Karaman S., Frazzoli E., "Sampling-based Algorithms for Optimal Motion Planning", 2011

[44]Chandrupatla T. R., Belegundu A. D., "Introduction to Finite Elements in Engineering, Third Edition", Pearson Education, 2006