Comparison of Aerial Collision Avoidance Algorithms in a Simulated Environment

by

James Holt

A thesis submitted to the Graduate Faculty of
Auburn University
in partial fulfillment of the
requirements for the Degree of
Master of Science

Auburn, Alabama
May 6, 2012

Keywords: Unmanned Aerial Vehicle, Collision Avoidance

Copyright 2012 by James Holt

Approved by

Saad Biaz, Chair, Associate Professor of Computer Science and Software Engineering
David Umphress, Associate Professor of Computer Science and Software Engineering
Hari Narayanan, Professor of Computer Science and Software Engineering

Abstract

In the field of unmanned aerial vehicles (UAVs), several control processes must be active to maintain safe, autonomous flight. When flying multiple UAVs simultaneously, these aircraft must be capable of performing mission tasks while maintaining a safe distance from each other and obstacles in the air. Despite numerous proposed collision avoidance algorithms, there is little research comparing these algorithms in a single environment. This paper outlines a system built on the Robot Operating System (ROS) environment that allows for control of autonomous aircraft from a base station. This base station allows a researcher to test different collision avoidance algorithms in both the real world and simulated environments. Data is then gathered from three prominent collision avoidance algorithms based on safety and efficiency metrics. These simulations use different configurations based on airspace size and number of UAVs present at the start of the test. The three algorithms tested in this paper are based on mixed integer linear programming (MILP), the A* algorithm, and artificial potential fields. The results show that MILP excelled with a small number of aircraft on the field, but has computation issues with a large number of aircraft. The A* algorithm struggled with small field sizes but performed very well with a larger airspace. Artificial potential fields maintained strong performance across all categories because of the algorithm's handling of many special cases. While no algorithms were perfect, these algorithms demonstrated the ability to handle up to eight aircraft on a 500 meter square field and sixteen aircraft safely on a 1000 meter square field.

Acknowledgments

First, I want to thank my Lord and Savior Jesus Christ for providing all the provisions I've needed to make it this far. He's guided me every step of the way, and without him, this would not have been possible.

I would like to express my deepest gratitude to Dr. Biaz for his guidance, support, and opportunity to work with him during my graduate studies at Auburn University. I would also like to thank all the members of the Auburn 2011 REU site for their algorithms development that made this study possible: Waseem Ahmad, Travis Cooper, Ben Gardiner, Matthew Haveard, Tyler Young, Andrew Kaizer, Thomas Crescenzi, James Carroll, Jared Dickinson, Jason Ruchti, and Chip Senkbell.

I'd like to thank the National Science Foundation for their continued support of the REU site at Auburn University and for their support of the Aerial and Terrestrial Testbed for Research in Aerospace, Computing, and maThematics (ATTRACT) through grants CNS #0855182 and CNS #0552627. I'd also like to thank everyone who's worked on the ATTRACT project to get it to the point where it is today. Special thanks to Alex Farrell, Jared Harp, and John Harrison for all the hardware assistance with both the UAVs and circuitry.

I'd like to thank my family who has supported me every step of the way and encouraged me to succeed in computer science. Finally, I'd like to show my utmost gratitude to my wife, Caroline, who has provided unwavering support and love throughout this whole process. Love you, Caroline.

Table of Contents

Abstract ii

Acknowledgments iii

List of Figures vii

List of Tables ix

List of Abbreviations x

1 Introduction 1

1.1 General Problem 1

1.2 Physical Constraints 2

1.3 Analyzing Results 3

2 Survey of Aerial Collision Avoidance Algorithms 4

2.1 Geometric Approach 4

2.2 Evolutionary 5

2.3 Mixed-Integer Linear Programming (MILP) 6

2.4 Grid-based approaches 7

2.5 Artificial Potential Fields 8

2.6 Conclusion 9

3 Mixed Integer Linear Programming 10

3.1 Mixed-Integer Linear Programming (MILP) Basics 10

3.2 Constraints on the Function Solver 10

3.3 Problem Constraints 12

3.4 Handling Multiple Aircraft 13

3.5 Receding Horizon Control 13

4 A* Grid-Based Algorithm 16

4.1 A* Basics 16

4.2 Dynamic Sparse A* 17

4.3 Creating a Heuristic 18

4.3.1 Background Concepts 18

4.3.2 Predicting Plane Locations 19

4.3.3 Estimation of Best Cost to Goal 22

4.4 Search with Dynamic Sparse A* 23

5 Artificial Potential Fields 25

5.1 Simple Artificial Potential Field (APF) Implementation 25

5.2 Calculating Force Vectors 26

5.3 Special Cases 30

5.3.1 Handling Deadlocks 30

5.3.2 Right Hand Rule 31

5.3.3 Aircraft Priorities 32

5.3.4 Aircraft Looping 33

6 Test-bed Design 35

6.1 Test-bed Requirements 35

6.2 Pre-existing Hardware Configuration 36

6.2.1 Base ArduPilot Configuration 36

6.2.2 AU Proteus modifications 36

6.3 Pre-existing Software Configuration 37

6.3.1 ArduPilot Modifications 37

6.3.2 AU Proteus JAVA Controller 37

6.4 Robot Operating System 37

6.4.1 Nodes 38

6.4.2 Messages and Topics 38

6.4.3 Services 39

6.5 Core Control Nodes 39
6.5.1 X-Bee IO 39
6.5.2 Coordinator 41
6.5.3 Control Menu 42
6.6 Core Message Types 42
6.6.1 Telemetry Updates 43
6.6.2 Commands 43
6.7 Research Nodes 43
6.7.1 Simulator 43
6.7.2 Collision Avoidance 44
6.7.3 Visualization 45
7 Results 46
7.1 Number of Conflicts 47
7.2 Collision Reduction and Survival Rate 48
7.3 Aircraft Life Expectancy 52
7.4 Efficiencies of the Algorithms 53
7.5 Algorithm Performance 55
7.5.1 MILP performance 55
7.5.2 A* performance 56
7.5.3 Artificial Potential Fields performance 57
7.5.4 Conclusion 58
Bibliography 61
A Links to source code 63

List of Figures

2.1 Relative motion of two UAVs 5

3.1 Turning Constraint 12

3.2 RHC versus Full Path Planning 14

4.1 Turning angle and constant speed constraints on a grid 20

4.2 Danger grids through time 20

4.3 A plane's predicted path. The yellow path is a predicted turn. The purple represents the path to an intermediate waypoint and the green is the final path taken to the planes goal. 21

4.4 Plane buffer zones 22

5.1 Determining the maximum distance of the repulsive force field 27

5.2 Geometry of calculating the repulsive force acting on U_i 28

5.3 Calculating the total force acting on a UAV 29

5.4 Adjusting the repulsive force to handle head-on collisions 30

5.5 Situation in which the right hand turn rule should not be applied. U_i should continue towards its destination instead of traveling behind U_k. 31

5.6 Finding the new repulsive force vector in order to travel behind U_k 32

5.7 Geometry involved to detect a looping condition. 34

6.1 The layout and interactions of ROS nodes in this system. Nodes are ovals with services represented by dashed lines and messages by solid lines [5]. 40

7.1 Average conflict reduction on a 500x500 meter field 47

7.2 Average conflict reduction on a 1000x1000 meter field 48

7.3 Average collision reduction on a 500x500 meter field 49

7.4 Average collision reduction on a 1000x1000 meter field 50

7.5 Average survival rate on a 500x500 meter field 50

7.6 Average survival rate on a 1000x1000 meter field 51

7.7 Average increase in life expectancy on a 500x500 meter field 52

7.8 Average increase in life expectancy on a 1000x1000 meter field 53

7.9 Average increase in the number of waypoints achieved on a 500x500 meter field 54

7.10 Average increase in the number of waypoints achieved on a 1000x1000 meter field 54

List of Tables

7.1 Table showing the best performing algorithm for each category 59

List of Abbreviations

APF Artificial Potential Fields

DSAS Dynamic Sparse A* Search

GPS Global Positioning System

MILP Mixed-Integer Linear Programming

PCA Point of Closest Approach

ROS Robot Operating System

SAS Sparse A* Search

UAV Unmanned Aerial Vehicle

Chapter 1

Introduction

One area of technological research that has been growing increasingly important over the last decade is the field of unmanned aerial vehicles (UAVs). In particular, UAVs are being used for a large number of surveying applications. This includes surveying an area of land, points of interests, or other mobile vehicles. However, in order to protect these UAVs, algorithms are needed to route them safely through the airspace. In particular, missions involving more than one UAV need these algorithms for safety.

Many algorithms have been proposed and even implemented to help manage this UAV safety problem. Some algorithms work by generating maneuvers on a per second basis whereas others work by planning a path far into the future. Despite all the variations in collision avoidance, there is almost no work comparing them on a singular system. The goal of this research is to take some of these prominent algorithms and evaluate their effectiveness.

1.1 General Problem

Before any algorithms were chosen for experimentation, the problem first had to be defined along with some goals. The base problem is to test and compare collision avoidance algorithms controlling multiple aircraft to determine which algorithms are the best. This meant that there needed to be a standard test bed to perform all these calculations on and a standard set of rules for the simulations. Since this problem will eventually be tested in a real-world setting, some of these standards were chosen based on real model aircraft while others were made to help frame the problem better.

One of these framing standards was to limit the collision avoidance algorithms to a two dimensional airspace with the idea of eventually expanding the simulations to three dimensions. The basic idea being this limitation was that if an algorithm can safely navigate aircraft through a two dimensional space, it should be able to do it through a three dimensional space after some modifications. Additionally, this would save time during algorithm development by limiting the airspace. This would also allow for potential adaptation to ground based vehicles which typically navigate using two dimensional space.

In this paper, environmental factors are also excluded from the problem. Typically, real aircraft will experience a large number of environmental influences including air resistance, wind, violent weather, and unresponsive obstacles such as unknown aircraft. Furthermore, it's assumed all aircraft in the simulation can be tracked and that their positions are known. This means the algorithms assume all data received from the aircraft is accurate and that all obstacles are known by the system.

1.2 Physical Constraints

There are also constraints due to the physics of a real aircraft. Specifically, these algorithms are limited to aircraft with a maximum turn radius and constant speed. For this problem, the maximum turning angle was limited to 22.5°/second. This means for an aircraft to reverse its direction, it would have to spend eight seconds of flight time changing its heading. The aircraft was also limited to an airspeed of 25 miles/hour which is approximately 11.176 meters/second.

This problem is also considered a real-time problem. While the algorithms can choose how long they wish to perform a particular calculation, the simulation will continue running during this calculation time. In particular, the simulation would generate a new update from each aircraft at a rate of one update per second and it will continue generating those updates and following the preset course unless the collision avoidance algorithm alerts the aircraft to a new path or until the aircraft is determined to have a collision.

1.3 Analyzing Results

With the problem limitations defined, the methods for analyzing the algorithms must be defined. First, collision avoidance algorithms are meant to reduce both the number of collisions and the number of conflicts of the aircraft. A collision can be defined as two aircraft occupying the same space. In this real world, this would typically imply destruction of both aircraft. Because of the granularity of the simulations, a collision is defined as two aircraft being approximately one second of flight time away from each other. Since the airspeed of the aircraft is a constant 11.176 m/s, a circle of radius 12 meters is chosen as the collision zone for these aircraft. Similarly, a conflict is meant to represent an imminent collision between aircraft. For this conflict zone, the radius is double to a 24 meter circle representing the conflict zone. For these simulations, each conflict and collision is tracked but a collision will remove any aircraft involved from future time steps to simulate the destruction of those aircraft.

While these two definitions describe the effectiveness of the collision avoidance, there needs to also be a way to describe how efficient the algorithms are at reaching the goal waypoints. In order to do this, the increase in waypoints achieved was used. This metric was used because it accounts for both the effectiveness of collision avoidance and the efficiency of the maneuvers. For example, an algorithm that loses several aircraft will have a lower waypoints achieved, but an algorithm that takes overly cautious maneuvers will also have a very lower number of waypoints achieved.

In the next chapter, several proposed algorithms are discussed along with some of the known strengths and weaknesses of those algorithms. Then, three of those algorithms are discussed further, implemented, and modified to improve on some of the known or discovered weaknesses. Despite all the differences in these equations, these algorithms are then implemented on a single common testbed for analysis. Finally, the results from the test bed simulations are presented along with some analysis of the different algorithms.

Chapter 2

Survey of Aerial Collision Avoidance Algorithms

2.1 Geometric Approach

Geometric methods for collision detection and maneuvering are based on a couple of simple geometric principles. For this particular approach, the point of closest approach (PCA) algorithm was examined. In this algorithm, the aircraft involved are typically considered a single point mass with a constant velocity vector denoting speed and direction [8]. By extending these velocity vectors out from the point mass, it's possible to determine how close two aircraft will come to each other, a distance referred to as $\vec{r}_m$, the "miss distance vector" [8]. Referring to figure 2.1, if $\hat{c}$ is the unit vector representing the difference in the velocities of the two aircraft and $\vec{r}$ is the relative distance between the point masses, then $\vec{r}_m$ is defined:

$$\vec{r}_m = \hat{c} \times (\vec{r} \times \hat{c}) \tag{2.1}$$

If this miss distance vector is lower than a pre-defined safe threshold, typically the conflict distance, then the algorithm knows that it needs to re-route one or both of the aircraft. Further vector manipulation is done on the miss distance vector in order to determine the maneuvers that need to take place to resolve the conflict [8].

One of the benefits of geometric approaches is the fact that typically, they can be implemented in both 2D and 3D spaces by modifying the point mass and velocities to include a z component. In addition, it's already assumed that the aircraft maintain a constant speed and ignores environmental components [8]. This algorithm is also a very efficient algorithm in terms of time. With only two aircraft, it's fairly trivial to determine if their paths are in

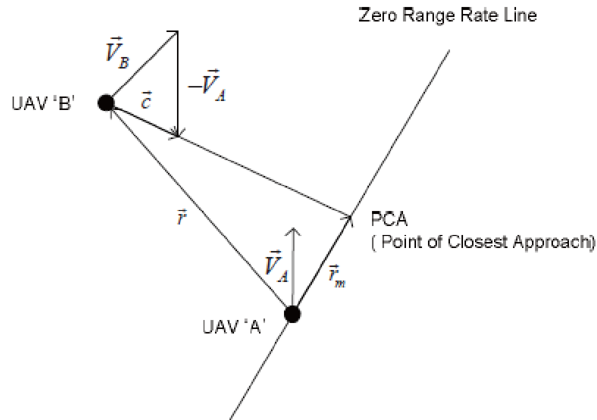


Figure 2.1: Relative motion of two UAVs

conflict and then how to re-route those aircraft. However, this particular algorithm is only shown to work for two aircraft. As the number of aircraft increase, the problem of checking for conflict and re-routing becomes much more complex because you have to factor in each point mass and vector.

2.2 Evolutionary

The evolutionary approach is a method of handling collision avoidance through adaptation of a known path. To start, the algorithm creates a random set of solutions from the start point to the end [10]. From this point, the algorithm enters a cyclical series of steps. First, each path is mutated based on one of the following four techniques and that new mutated path is added to the pool [10]:

1. Mutate and propagate - change one segment of the path and modified the rest of the path to reach the endpoint
2. Crossover - uses the start of one path and the end of another path and combines them using a “point-to-point-join function”
3. Go to goal - takes a point near the end of a path and performs the point-to-point-join function from that point to the end

4. Mutate and match - mutate one or more of the segments in the path, then attempt to connect the end of the mutated segments to the end portion of the path

This effectively doubles the pool size by creating a new, randomly-mutated path for each old path. Then, this pool goes through an evaluation based on a variety of techniques such as feasibility, constraints, goals, etc [10]. The most fit segments are kept within the pool and the others are pruned away until the pool is back to its original size [10]. This is one generation of the evolutionary approach, but in most scenarios, multiple generation would be calculated before returning anything to the aircraft.

This approach has a couple of benefits because of its system of execution. First, it's easily adaptable to a real-time environment. WIth the evolutionary approach, the algorithm can continuously create new random generations and can return a path (not necessarily the best path) at any point. Additionally, the iterative nature allows it to receive updated GPS data and modify its paths based on this new data [10]. This makes the algorithm extremely easy to adapt to a real-time system. One of the downfalls of this process is that the evolutionary algorithm is not guaranteed to ever find the optimal solution to a problem. However, it is still very effective at finding a solution and continuously adapting that solution as time passes.

2.3 Mixed-Integer Linear Programming (MILP)

While some of these approaches directly address the problem of collision avoidance, the MILP method is actually an adaptation of this problem into a set of constraints [11]. These constraints are then used as inputs into an MILP solver, a program that takes a set of constraints and attempts to find an optimal solution [11]. Today, there are many commercially available MILP problem solvers available for purchase.

With one of these solvers, the only things left to create are the input constraints. In order to adapt the problem for MILP, a few basic constraints need to be considered. First, each UAV is mapped into the program with it's mechanical state: position, velocity, and

acceleration [11]. This is necessary because the solver needs to know both where the UAVs are positioned and it needs to insure that the UAVs maintain a safe distance while solving. In turn, the constraints on safe distance must also be constraints to the problem solver [11]. Finally, the goal to minimize traveling time must be entered as a constraint on the system[11]. This gives the problem a goal and a method of evaluating success of each of the solutions calculated by the solver.

One of the problems with the MILP method is that it can be extremely time-consuming to calculate an optimal solution due to the infinite number of possibilities. Even with the relatively quick MILP solvers, these problems are grow exponentially with each UAV added to the equation. This means that for MILP to be a valid solution to collision avoidance, the problem needs to be adapted for real-time and the problem needs to be heavily constrained so that the time complexity goes down.

2.4 Grid-based approaches

Previous algorithms have used the aircraft as the focal points for collision avoidance. However, there are several algorithms that instead focus on a grid representing the airspace. These algorithms work by first taking the airspace and turning it into a discrete grid of squares (or cubes in a three dimensional environment). After generating this grid, each square is assigned values based on the algorithm. Typically, these values represent occupied grid-space either through static or dynamic obstacles.

After generating this grid, the algorithm uses the grid to generate a path for the aircraft. One of these specific algorithms is known as A*(spoken as "A-star"). In this algorithm, the grid is used as a graph that starts at the aircraft's initial position. From there, the idea is to consider nodes that the aircraft could travel to. Then, each of these nodes are assigned ratings that represent the best possible path to the destination. As each node is considered, it's added to a heap with the lowest cost element at the top of the heap. The process of rating these nodes is called "branching" [17]. After branching once, it repeats the process by

looking at the least cost node in the heap and considering where it can branch to from there. This recursive process is known as “bounding” the future search to only best-cost estimates [17]. Because of this, A* is known as a branch-and-bound method of handling these complex problems.

Grid-based problems do have some drawbacks caused by the grid. One of these is converting the three dimensional space into a three dimensional grid. For the planet, this means that the curvature of the earth has to be specially handled by the grid. Additionally, whenever a space is discretized, it can sometimes be difficult to determine what spaces are occupied by objects. For example, two aircraft in adjacent grids could be almost directly beside each other or on the far sides of their grid space. This can be accounted for by discretizing the airspace further, but that can leads to problems of complexity due to the grid space getting larger.

2.5 Artificial Potential Fields

Artificial potential fields offers a relatively simple method of handling collision avoidance. In physics, particles have both positive and negative charges where particles of similar charge repel each other and particles of opposite charge attract. Researchers discovered a way to apply this idea to collision avoidance.

In this approach, each aircraft is considered a negative charge in order to make them feel an artificial repulsive force from each other [6]. Additionally, each aircraft has a goal waypoint associated with a positive charge [6]. In order to determine where the aircraft should travel, the repulsive forces and attractive forces are added together into one vector. Then, this vector is used to determine what direction the aircraft should travel to remain as safe as possible [6]. This process is repeated for each aircraft in the scenario until each has a direction vector.

Artificial potential fields is generally a significantly faster method of handling collision avoidance compared to other algorithms. This is primarily due to the simplicity of the

calculation. However, the simple design makes this algorithm purely reactive in nature. This means that each time the algorithm is executed, it makes a greedy choice based purely on the current state of the aircraft. Because of this greedy choice, there isn't any long-term planning for future time steps which may result in some special cases that can't be handled by only the artificial potential fields.

2.6 Conclusion

These five algorithms were the original algorithms that were looked at for use in this collision avoidance test. Of those five, three algorithms were chosen to participate in this comparison: MILP, A*, and artificial potential fields. The geometric approach was found to have problems with working with a large number of aircraft, one of the problem requirements. While the evolutionary approach doesn't have that problem, it did suffer from problems with random mutations and so more predictable algorithms were chosen for testing. MILP and A* were both algorithms that could be potentially time consuming but were found to be adaptable to real-time situations with the proper problem constraints. Finally, artificial potential fields is a well-known solution to collision avoidance that has some special situations that can be handled with proper planning. The following chapters address these algorithms in greater detail along with some of the changes that were made to fit this specific problem.

Chapter 3

Mixed Integer Linear Programming

3.1 Mixed-Integer Linear Programming (MILP) Basics

MILP is one way to adapt the collision avoidance problem to a known problem solver. MILP is useful because it allows researchers to take a set of constraints and place them on the input values in order to achieve a goal [11]. In the case of collision avoidance, you want to minimize time traveling to the various waypoints while also minimizing collisions and conflicts [11]. Unfortunately, MILP solvers suffer from the infinite number of possibilities for these UAVs. With each aircraft added to the problem, the time to obtain a solution grows exponentially. Fortunately, since the UAVs reside in the physical world, there are several constraints that we can intuitively put on the solver to help reduce the problem. Additionally, the computation time can be further reduced using some techniques to reduce the problem size.

3.2 Constraints on the Function Solver

As mentioned earlier, the overall goal is to minimize the time needed to safely traverse several waypoints. Since the speed of the aircraft is constant, this is the same as minimizing the distance travelled to reach all of the waypoints. In order to do this, the objective function must be defined to reflect this goal. For this scenario, the objective function J is defined in Equation 3.1 where N is the number of aircraft in the problem and T_{F_v} is the time it takes for each aircraft to reach its final waypoint.

$$\min\ J = \sum_{v=1}^{N} (T_{F_v}) \tag{3.1}$$

With this goal defined, the solver now needs constraints on the problem. To begin, the state of each aircraft is defined in Equation 3.2 based on the three primary elements of motion: position, velocity, and force (from which acceleration can be calculated).

$$state = \begin{bmatrix} x \\ y \\ v_x \\ v_y \end{bmatrix} \qquad force = \begin{bmatrix} f_x \\ f_y \end{bmatrix} \tag{3.2}$$

Each aircraft will have a starting state based on it's latitude, longitude, and original bearing which will have to be converted into a Cartesian coordinate x, y, and velocity components. For an aircraft flying straight, the force component of the state would be zero, denoting that the aircraft is not feeling force from a turn. With this starting state defined, it's now necessary to relate each state to the previous state. Using well known physics equations, this relationship is easily defined using Equations 3.3, 3.4, and 3.5.

$$state_{t+1} = A \times state_t + B \times acceleration \tag{3.3}$$

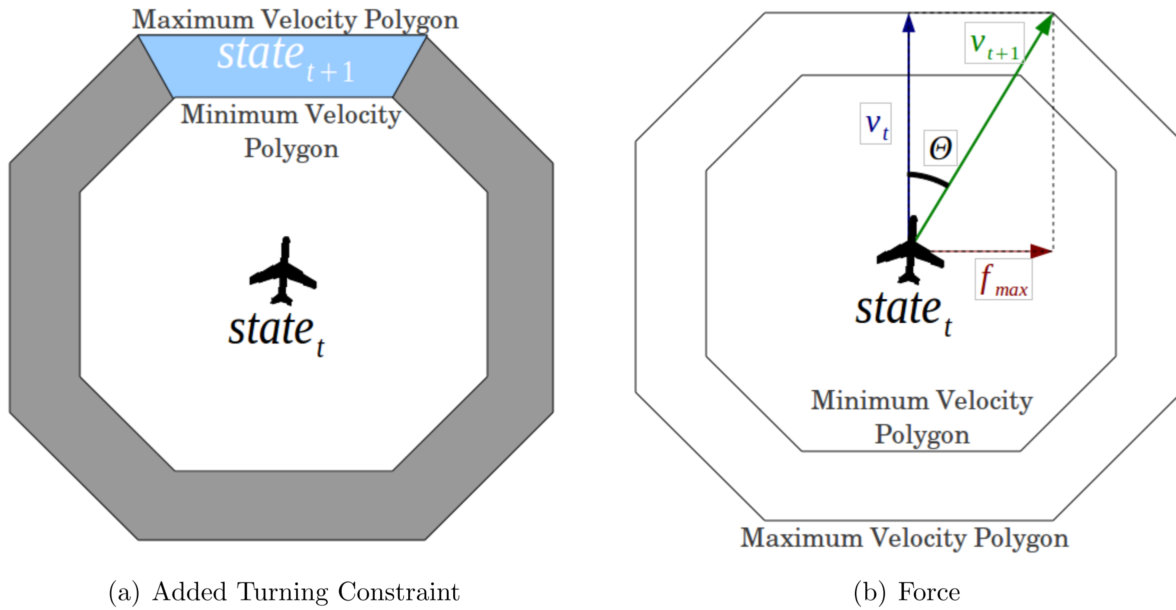
$$A = \begin{bmatrix} 1 & 0 & dt & 0 \\ 0 & 1 & 0 & dt \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad B = \begin{bmatrix} \frac{1}{2}(dt)^2 & 0 \\ 0 & \frac{1}{2}(dt)^2 \\ (dt) & 0 \\ 0 & (dt) \end{bmatrix} \tag{3.4}$$

$$state_{t+1} = \begin{bmatrix} x_t + v_x dt + \frac{1}{2}a_x(dt)^2 \\ y_t + v_y dt + \frac{1}{2}a_y(dt)^2 \\ v_x + a_x dt \\ v_y + a_y dt \end{bmatrix} \tag{3.5}$$

With this relationship in place, the fundamental components of solving the problem are in place. However, there are additional physical constraints that need to be added to the system in order to maximize the accuracy of the solution.

3.3 Problem Constraints

The problem definition included both a constant speed and a maximum turning angle for the aircraft, therefor the MILP solver must also be capable of accounting for both this speed constraint and maximum turning angle or infeasible solutions might be generated [11]. In order to account for these constraints, two things must happen. First, the distance of movement is converted into two polygons, a minimum and maximum velocity polygon. The reason for this is because polygons are easily understood by the MILP solver and can be used as an approximation for a circle around the aircraft. These two polygons are visually represented by Figure 3.1.



(a) Added Turning Constraint (b) Force

Figure 3.1: Turning Constraint

In addition to this distance constraint, the turn constraint can be represented by this polygon. Given the maximum turning angle, Θ, and the constant speed from the problem

statement, we can calculate the maximum force, f_{max} that can safely be exerted on the aircraft at any given moment. At this point, the solver has everything it needs to calculate the path of one single aircraft. By modifying the constrained force exerted at each state, the MILP solver changed the turn angle which dictates the next state's location and velocity. By doing this repeatedly, the solver can calculate several steps to form a complete path.

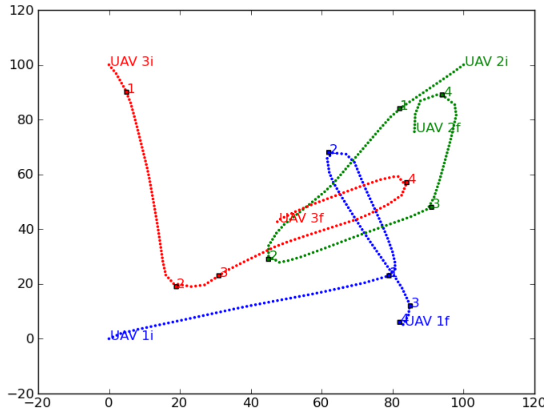
3.4 Handling Multiple Aircraft

In previous work, MILP solvers have been used for calculating paths for terrestrial vehicles performing various tasks [11]. In order to perform these tasks, the vehicles often had to avoid obstacles such as walls, boulders, or unsafe terrain. In order to do this, each obstacle was mapped into the coordinate space as polygons that cannot be entered by the vehicle [11]. With these additional constraints taken into account, the solution size gets reduced and the problem of avoiding these obstacles is handled.

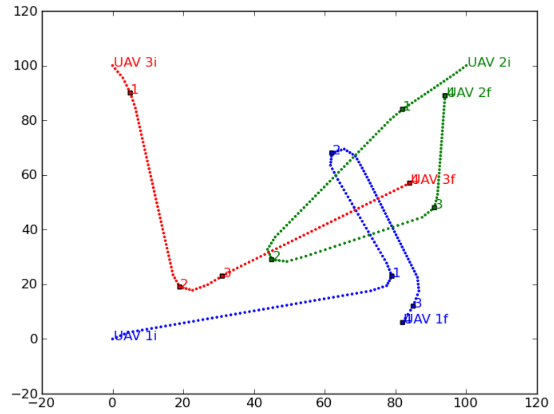
This previously implementation was used for static obstacles, but it can be easily adapted for dynamic use. In order to implement dynamic avoidance, Richards and How suggested treating the vehicles as rectangles and constraining the system to insure that no vehicle entered another vehicles rectangle [11]. While the rectangle idea wasn't used in this implementation, the idea of a safety distance and having dynamic constraints using this safety distance was borrowed. Instead of creating rectangles around each aircraft, the solver uses the distance formula to calculate the distance between vehicles at each state. Each generated state must enforce that the distance between each aircraft is greater than the safety distance. This constraint is the main component of MILP collision avoidance.

3.5 Receding Horizon Control

One of the major problems associated with MILP is the problem of size. Typically, MILP is used to generate a one-time solution for the entire problem. Unfortunately, this type of solution doesn't work well in a real-time environment such as collision avoidance for



(a) RHC Flight Map



(b) MILP Flight Map

Using RHC to compute paths:
Vehicle 1 Completion Time: 160
Vehicle 2 Completion Time: 143
Vehicle 3 Completion Time: 125
Number of Infeasible Solutions Returned: 20
Number of Feasible Solutions Returned: 139
Mean Iteration Computation Time:
0.0676489431153
Total Computation Time: 10.7561819553

Using MILP to compute paths:
Computation Time: 865.9s
Vehicle 1 Completion Time: 159
Vehicle 2 Completion Time: 138
Vehicle 3 Completion Time: 118

Figure 3.2: RHC versus Full Path Planning

several reasons. The time to compute one master solution is infeasible for real-time problems, and that time only grows exponentially with each aircraft added to the equation [14].

Fortunately, receding horizon control is one of the solutions to this problem. Receding horizon works by limiting the number time steps into the future that the algorithm is calculating [14]. For example, instead of calculating one master solution over 10 minutes of time, the algorithm may calculate 60 solutions that each span only 10 seconds. One of the downsides of the solution is a loss of optimality. Because each solution covers only a small portion of time, each solution is only optimal for that period of time which may lead to suboptimal solutions overall [13]. However, this suboptimal solution is still significantly better computationally and doesn't vary too far from the overall optimal solution. Referring to Figure 3.2, the receding horizon computation time total was a little under 11 seconds, whereas the total computation was 865.9 seconds. Additionally, the receding horizon completion time was a

total of 13 seconds slower than the three aircraft. The most important thing to note is the mean computation time of .0676 seconds which is fast enough to function for this real-time system. Because of these positive results, receding horizon control is implemented in the MILP algorithm used for testing.

Chapter 4

A* Grid-Based Algorithm

4.1 A* Basics

As described earlier, A* is considered a grid-based algorithm. This means that the airspace must first be converted to a grid and then that grid is used as a basis for the collision avoidance algorithm. A* specifically works through the "branch-and-bound" method of handling the problem. The algorithm starts at the node containing the aircraft and "branches" out to all nodes that the aircraft can possibly move. Then, these nodes are organized into a heap structure with the least cost node at the top of the heap. Next, the algorithm "bounds" the search by only considering the best possible known option which is as the top of the heap. The algorithm will do this recursively until the goal is reached.

This recursive algorithm means that the method of estimating cost is extremely important for the algorithm. This estimation, known as the algorithm's heuristic, is intended to calculate the best path from the initial node to the goal node. This cost is calculated based on two values: $g(n)$, the known cost from the start node to node n, and $h(n)$, the estimated cost of the best path from n to the goal node. The estimation, $f(n)$, can be calculated using Equation 4.1 [17].

$$f(n) = g(n) + h(n) \tag{4.1}$$

One of the benefits of this algorithm is that A* guarantees finding the optimum path as long as two requirements are met [17]. The first requirement is that A* should not overestimate the cost of $h(n)$, the cost from n to the goal. Second, $h(n)$ needs to be estimated

consistently. As long as these conditions hold, A* has the advantage of generating optimum solutions. Unfortunately, the complexity of A* leads to high computation times that grows exponentially with the input [17]. Fortunately, good heuristics can help reduce this complexity while still maintaining a high level of optimality.

4.2 Dynamic Sparse A*

The algorithm specifically used in this paper is called Dynamic Sparse A*, a derivative of Sparse A* Search (SAS). SAS was originally designed for military flight planning [16]. This algorithm helped reduce the complexity issues described earlier through some assumption about flight. One such restriction is the turning angle constraint which limits the grids considered to only grids in front of the aircraft [16]. Second, the minimum travel distance limits the grid space even further to only those a certain distance in front of the aircraft [16]. Both of these constraints can be shown by Figure 4.1.

SAS alone provides a large numbers of benefits to the collision avoidance problem. First, it provides globally optimal paths based the A* algorithm. Second, time complexity only scales linearly with the number of aircraft used in this kind of algorithm. Additionally, the algorithm can be parallelized such that each aircraft can use it's own processor allowing for faster calculations. Finally, with a strong heuristic, the A* algorithm can plan paths for multiple aircraft in a very short amount of time.

In order to improve further on this algorithm, Dynamic Sparse A* Search (DSAS) was created to strengthen the heuristic used by the algorithm. One of these improvements is to first check whether danger is present between an aircraft and its goal. In the case of no danger, A* searches are unnecessary and the aircraft can simply fly a normal path to the goal. This helps reduce unnecessary computations on the system. The major improvement and where DSAS gets its name from is the way it can dynamically plan around moving obstacles such as other aircraft. In order to handle this, DSAS uses grids created through

time. Finally, the algorithm is capable of handling changes to the conditions of the airspace and is capable of handling a high number of aircraft path planning.

These improvements lead to two main components of the algorithm: heuristic generation and the search. The heuristic calculation is created by rating the danger in each grid space and through several time steps into the future. This danger grid combines all the aircraft danger zones into one grid which is then used to generate a path from the aircraft's location to its goal.

4.3 Creating a Heuristic

The main way to reduce complexity in A* is by modifying the heuristic function described by Equation 4.1. As mentioned before, $g(n)$ is the known component of the calculation generated by the search whereas $h(n)$ is only an estimation. Since $h(n)$ is the only unknown, a good heuristic is defined by this estimation. In order to accurately and efficiently perform this algorithm, $h(n)$ must not overestimate this cost while maintaining a close estimate [17].

4.3.1 Background Concepts

The first step in performing this algorithm is to create a grid representing the real-world airspace. Originally, this danger grid was used by Szczerba *et al.* as a best cost grid [16]. This grid was designed to represent occupied airspace as having an extremely high cost such that the algorithm wouldn't attempt to traverse it. In addition to creating this $x*y$ best cost grid, DSAS adds the time component, t, which creates an $x*y*t$ matrix of best cost values throughout time. Note that for this problem set, the altitude (third physical dimension) is not included, but could be included to create an $x*y*z*t$ matrix.

One of the problems described earlier was defining this grid space dimensionally. In order to do this, the actual attributes of the physical aircraft came into play. First, the tests would be conducted on both 500 and 1000 meter square fields. Because of this, the

resolution chosen was 10 meters per grid square. Second, the aircraft sends GPS updates once per second, so the time step chosen was one second with a prediction of 20 seconds (total of 21 including the present state $t = 0$) into the future. This means that the total resolution is 100 by 100 by 21 grid squares. This is best shown by looking at Figure 4.2 which shows the grid from time 0 to 5.

Inside each square is the danger rating which represents the likelihood of encountering an obstacle in that square. Note that for this experiment, the only obstacles included are other aircraft but static obstacles could be included by simply making the squares occupied by that static obstacle have a probability of 100%. This danger rating is related to the best cost values described earlier and used by Szczerba [16]. In order to calculate this danger rating, the algorithm predicts the location of each aircraft throughout time and fills in the danger grid with the appropriate values. However, you don't want an aircraft to try avoiding itself so each aircraft is said to "own" a danger grid which doesn't contain its own danger ratings.

4.3.2 Predicting Plane Locations

With the grid structure defined, the next step is determining how to accurately predict plane locations and how to use that prediction to fill in the danger grid. Predicting the linear path from one point to another is considered a fairly simple thing to do. As mentioned earlier though, using a discretized airspace to leads to problems in determining the aircraft's path. A straight line in grid space will typically bisect several grid squares in an uneven manner. Fortunately, this grid bisecting line can be used as a method of calculating the probability that an aircraft will occupy a grid square in the future. The aircraft's bearing to the goal is compared to the closest angle that will perfectly bisect a neighboring square. The offset between these two allows for predictions of how much time the neighboring square will be occupied by the aircraft. The remaining percentage is used as a prediction for the next closest square. These two probabilities are then inserted into the danger grid as shown by

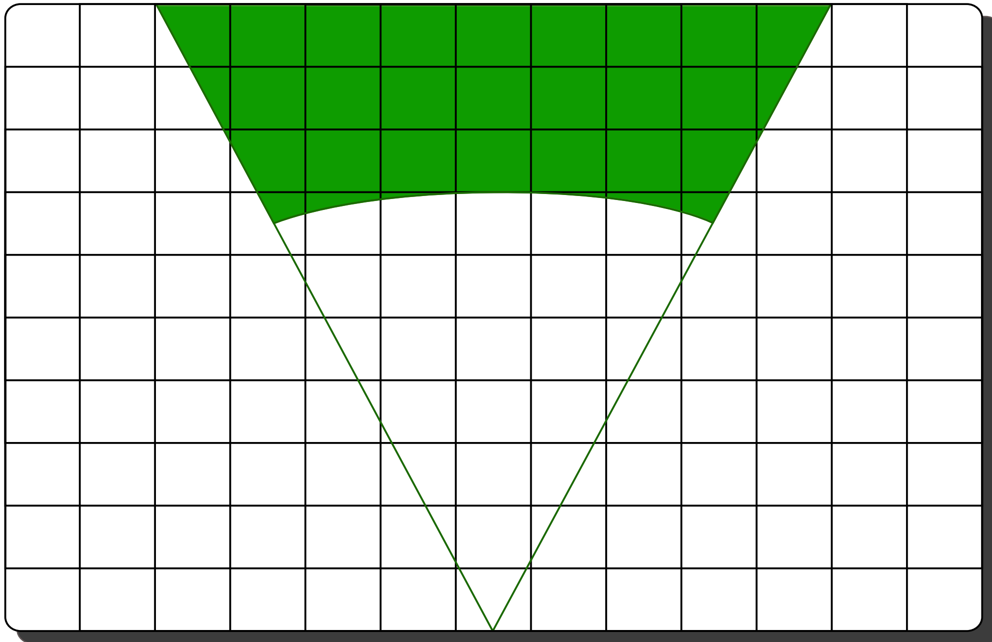


Figure 4.1: Turning angle and constant speed constraints on a grid

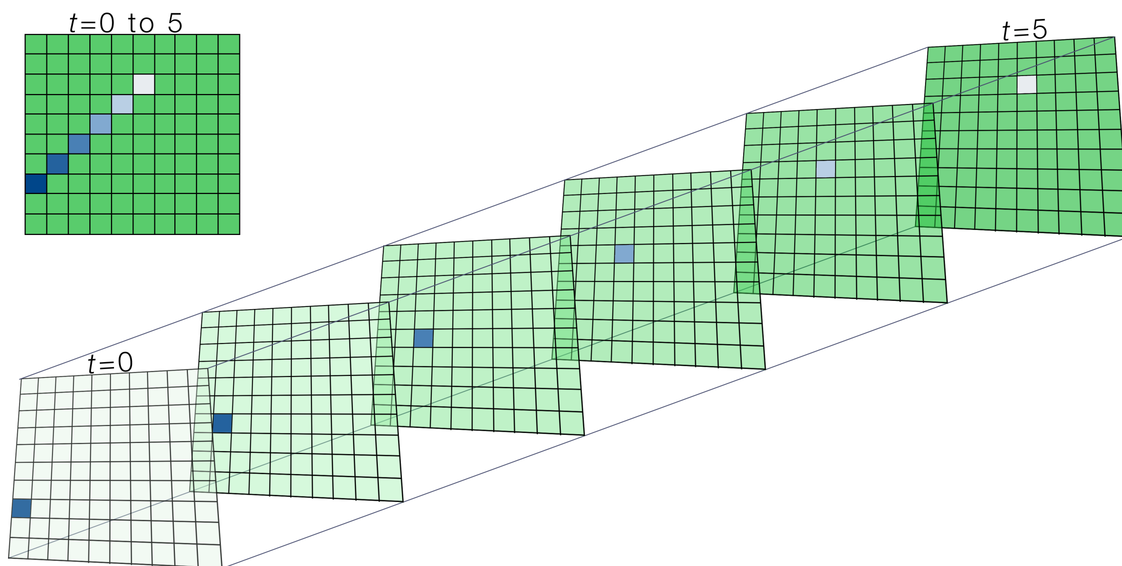


Figure 4.2: Danger grids through time

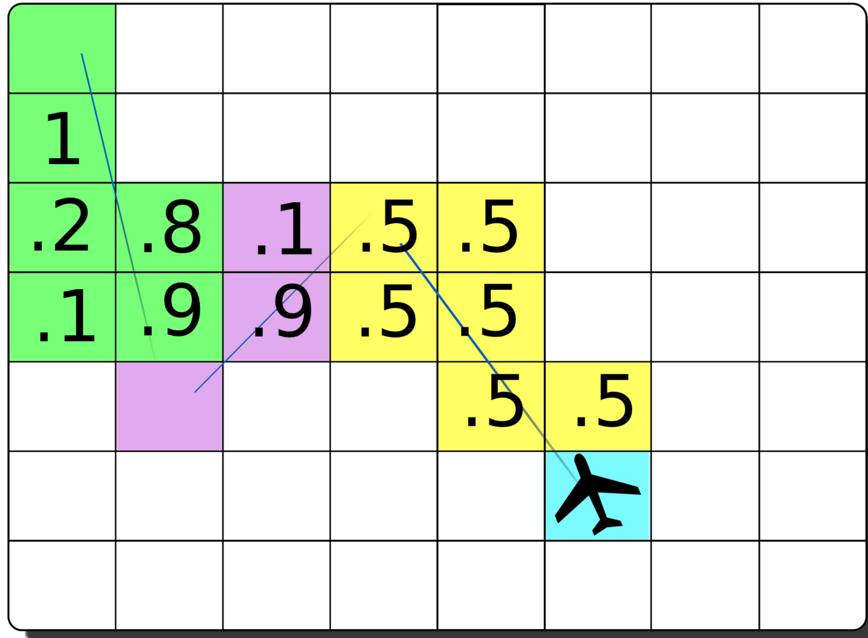


Figure 4.3: A plane's predicted path. The yellow path is a predicted turn. The purple represents the path to an intermediate waypoint and the green is the final path taken to the planes goal.

Figure 4.3. This process is repeated until the aircraft is within the grid square containing the goal.

This system works well for straight lines but not for curves or turns in the aircraft's path. In order to compensate for this, the straight line method isn't used until the goal is within the maximum turning angle of the aircraft. Once this condition is met, the straight line algorithm is used to predict the future occupied airspace.

The final step for filling the danger grid is to create some buffer zones around the aircraft. First, the danger rating of occupied squares would be the highest value allowed by the algorithm to represent that under no circumstance should another aircraft attempt to fly through that space. In addition to this central location, buffer zones are added around the path with lower danger ratings. In front of the aircraft, the danger rating would be higher because there is a higher likelihood that the aircraft may end up in that grid square, but squares on the edge of the maximum turning angle may have a lower danger rating. Referring

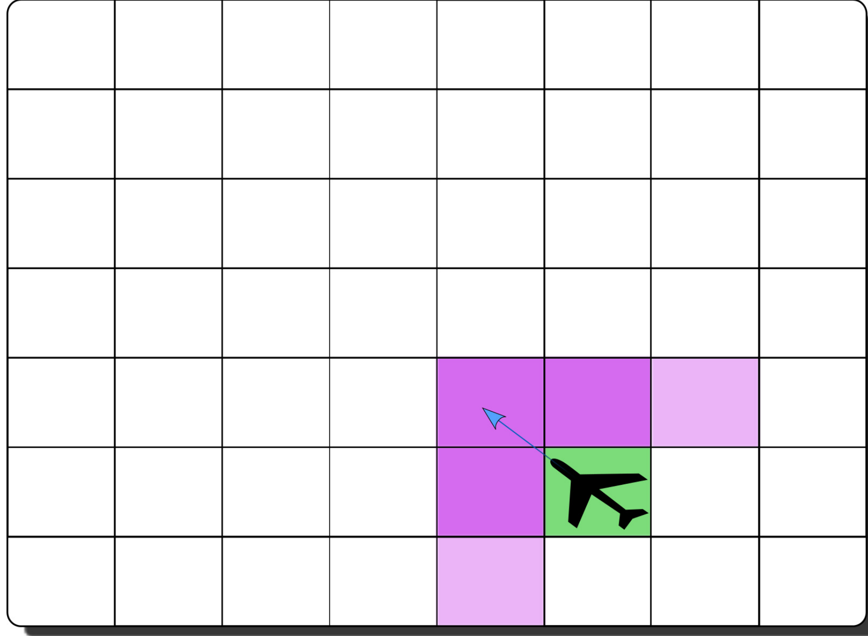


Figure 4.4: Plane buffer zones

to Figure 4.4, the green square is the predicted location of an aircraft in the danger grid at time t and therefor has the maximum danger rating. Additionally, the purple squares have danger ratings with the darker purple representing a higher rating because the aircraft is likely to move into those squares at time $t+1$ or even occupy those squares at time t.

4.3.3 Estimation of Best Cost to Goal

With the danger grid defined, the last step before the A* search is to define the best cost grid that's used in the A* search algorithms. Typically, the estimate $h(n)$ is determined based on the distance to the goal. However, in this case, this estimate will actually be a function of the danger rating and distance to the goal. Since any airspace using this algorithm would be reasonably sparse and there are no static obstacles, the entire calculation of $h(n)$ is done by Equation 4.2. Within this equation, both the distance from the goal and a scaled danger rating are added together to create the final best cost estimation.

$$h(n) = (\text{distance from } n \text{ to the goal}) + (\text{danger rating of } n) * (\text{scaling factor}) \quad (4.2)$$

While this heuristic may seem fairly simple, it works well for a couple of reasons. First, in a relatively sparse airspace without obstacles, it will always be preferable to perform the simple navigation around another aircraft rather than to risk a conflict or collision. Second, by using this method, straight line paths will be preferred as well. Note that if there are no other aircraft in the airspace, the straight line solution will always be chosen by DSAS because it is the shortest physical distance. Using this solution also helps handle the problem of overestimation by making it impossible to overestimate the distance portion of the heuristic. This helps the algorithm adhere to the A* rules of not overestimating on the best cost grid. The final reason to use this heuristic is time complexity. By using this heuristic, the algorithm complexity is $O(xytp)$ where x and y are the length and width of the airspace, t is the number of time steps, and p is the number of aircraft in the airspace. Since x, y, and t are typically chosen beforehand, this means that the best cost grid calculation is linear with respect to the number of aircraft in the airspace.

4.4 Search with Dynamic Sparse A*

Once the danger grid has been created, DSAS creates a best cost grid that can finally be used by the A* algorithm to create the path to the goal. In order to create the path, A* will use the branch-and-bound methodology described earlier [17]. First, the algorithm will branch out to all adjacent node from the start node and order them into a best cost heap with the minimum cost at the top. Then, the algorithm bounds the search by looking at all nodes branching off of the minimum cost node. The algorithm will recursively do this branch-and-bound technique until it reaches the goal. Note that if the algorithm detects a

better solution, it will be found at the top of the heap so at any point during this recursion, the bounding may determine that a different path should be pursued [17].

Unfortunately, this basic A* implementation can create infeasible solutions in certain scenarios. Consider a situation where is aircraft's current bearing is due north but the goal is located south of the aircraft. Given an obstacle free airspace, the traditional A* would choose a straight line path south which is an impossible maneuver for the aircraft. The aircraft would need to spend time to turn around and then head south to the goal.

In order to handle this type of situation, the branching in DSAS is limited to nodes within the maximum turning angle and within the distance that can be travelled in one time step. This constraint was previously shown in Figure 4.1. Having these two constraints actually improves the performance of the algorithm by limiting the number of options that DSAS has to search through when finding the best cost path [16]. The algorithm is further optimized by checking for the base case of a danger free path to the goal. If this is the case, the aircraft is simply allowed to fly to its goal with interruption from DSAS.

In short, DSAS works by first creating a danger grid representing the existence of hazards in the real world. This danger grid is then converted into a best cost grid for each aircraft that's composed of both the danger rating and distance to goal of each node. Finally, DSAS will use the limited branch-and-bound technique until the minimum node on the heap is the goal or until the maximum time step value is reached. Once the search is complete, the waypoints are transmitted to keep the aircraft both safe and on track to the destination.

Chapter 5

Artificial Potential Fields

5.1 Simple Artificial Potential Field (APF) Implementation

The basic idea behind this algorithm is to create artificial fields of positive and negative energy around objects in the airspace. Specifically, all aircraft in the airspace will have a negative charge associated with their locations and the goal of each aircraft will have a positive charge on that aircraft [6]. By following the same rules as magnets, the aircraft (negative charges) will feel repulsive forces from each other and an attractive force from their respective goals [6]. Each of these forces carries a weight on the final calculation of the force vector for that aircraft. Then that vector is used to model the direction the aircraft should go to travel safely.

These weights were tested in order to define the attraction constant. This attraction constant, γ, was meant to serve as a method of determining the best weights for the attractive and repulsive forces [15]. In practice, γ is used as the weight of the attractive force felt by the aircraft and is a value such that $0 < \gamma < 1$. For the full force equation, please refer to Equation 5.1 [15].

$$\vec{F_{tot}} = \gamma * \vec{F_{attr}} + (1 - \gamma) * \vec{F_{rep}} \tag{5.1}$$

In previous research, Sigurd and How used this equation to test this collision avoidance algorithm as well [15]. They found that the best results were achieved when using $\gamma = .66$ for the attractive weight and .34 for the repulsive weight [15]. This specific implementation follows their research in using both the force equation and those numerical weight values. This formula forms the basis of this artificial potential field implementation.

5.2 Calculating Force Vectors

With the basic equation for combining attractive and repulsive forces defined, the algorithm now needs a way to calculate what those force vectors are. First, in the real world, magnetic forces are based primarily on the distance between the two objects. This force can be felt at any distance but gets weaker as the distance grows. However, at a certain point, this force becomes negligible, so this algorithm needs to define d_{fmax}, the maximum distance at which one aircraft's force can be felt.

In order to define this, the constant d_{onesec} is used to represent the distance that the aircraft can travel in one second. In addition to this constant, a scale factor α is used to define the ratio of the collision zone to the size of the potential field. Through experimentation, this implementation chose to use $\alpha = 5$ in the calculation. Finally, the field is modified to be an elliptical shape with the force field extending farther in front of the aircraft to represent where that aircraft is traveling [7]. To define this elliptical shape, two constants, λ_{scalef} and λ_{scaleb}, are used to determine the amount of the elliptical extending in front and behind the aircraft. These values were experimentally chosen to be $\lambda_{scalef} = 2$ and $\lambda_{scaleb} = 1.25$. Finally, θ represents the angle from the bearing of one aircraft, U_k, to the position of the current aircraft, U_i. These values are then used in Equation 5.2 to solve for the maximum distance that would allow an aircraft to effect the current aircraft's repulsive force calculation. Figure 5.1 represents the force field around an aircraft that may be included in this calculation.

$$d_{fmax} = \alpha * d_{onesec}[(\lambda_{scalef} - (\lambda_{scalef} - \lambda_{scaleb})/2)] + ((\lambda_{scalef} - \lambda_{scaleb})/2) * cos(\theta)) \quad (5.2)$$

Similar to the maximum distance described above, there also needs to be a minimum distance, $d_{failsafe}$, that will modify our force value to represent an impending collision [7]. If the distance between two aircraft becomes less than this minimum distance, both will feel a strong force, $F_{failsafe}$, that overpowers all other forces in the calculation.

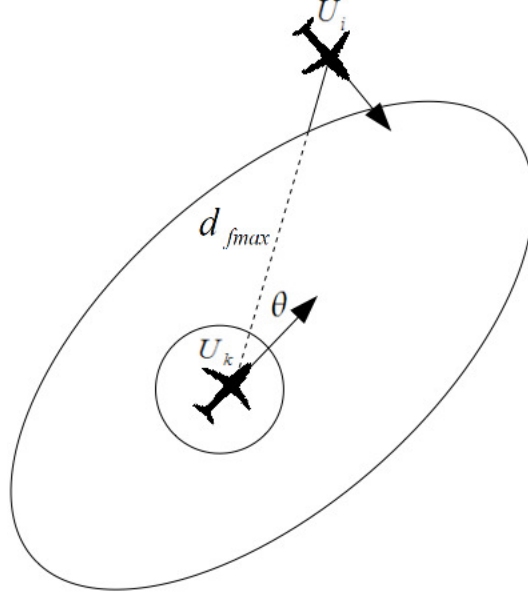


Figure 5.1: Determining the maximum distance of the repulsive force field

Finally, the last range to be covered is when $d_{failsafe} < d \leq d_{fmax}$. In this range, there are two primary factors that will effect the repulsive force. First, the distance between the two aircraft is a major factor as it was for determining the minimum and maximum forces [7]. Second, the position and direction of the aircraft is taken into account such that if one aircraft is in front of another, it will feel a stronger force trying to push it out of the way. In contrast, if an aircraft is behind another, it won't feel as strong a force because the aircraft is moving away. This final calculation uses two constants: k_{emitf} for the scalar effecting the force in front of U_k and k_{emitb} for the forces behind it. The value q_{UAV} is used as the negative charge from a UAV in this equation. Finally, let λ be a scalar value that helps check the strength of this repulsive force. With all of these conditions and values taken into account, the final repulsion force is calculate with Equation 5.3.

$$r(\theta, d) = \begin{cases} 0 & \text{if } d > d_{fmax} \\ q_{UAV} * [(k_{emitf} - (k_{emitf} - k_{emitb})/2)] + ((k_{emitf} - k_{emitb})/2) * cos(\theta)\frac{d_{fmax} - d}{\gamma * \alpha} & \text{if } d_{failsafe} < d \leq d_{fmax} \\ F_{failsafe} & \text{if } d \leq d_{failsafe} \end{cases} \tag{5.3}$$

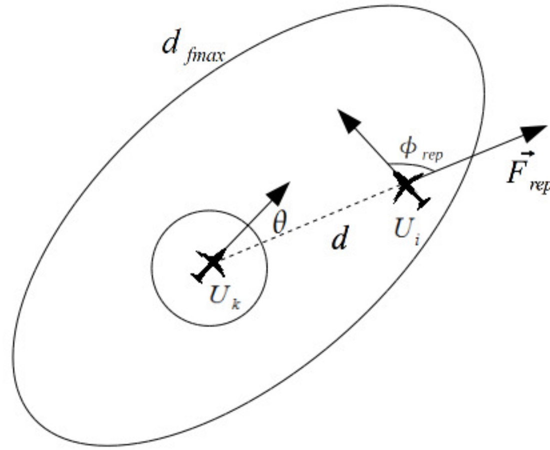


Figure 5.2: Geometry of calculating the repulsive force acting on U_i

Note that through experimentation, this implementation found that $q_{UAV} = 80$, $k_{emitf} = 1.5$, $k_{emitb} = 1$, and $\lambda = 4$ to be the constants with the best results.

With the emitted force calculated, the algorithm determines how much of the emitted force is felt by the aircraft. In order to calculate this, three additional values must be known: the angle between aircraft's bearing and the repulsive force (ϕ_{rep}), a scalar representing the force felt from obstacles in front of the aircraft (β_{feelf}), and a scalar representing the force felt from an obstacle behind the aircraft (β_{feelb}). Using these values, Equation 5.4 is used to increase the force felt from obstacles in the path of the aircraft. For a visual representation of the angle described here, please refer to Figure 5.2. Through experimentation, values of $\beta_{feelf} = 1$ and $\beta_{feelb} = .5$ were used for this implementation.

$$s(\theta, \phi_{rep}, d) = r(\theta, d) * [(\beta_{feelf} - (\beta_{feelf} - \beta_{feelr})/2) - ((\beta_{feelf} - \beta_{feelr})/2) * cos(\phi_{rep})] \quad (5.4)$$

With these two angle based equations defined, a couple observations can be made. First, the most powerful forces will be present when two aircraft are heading directly for each other. In these scenarios, if two aircraft are heading towards each other, the result angles are $\theta = 0$ and $\phi_{rep} = \pi$. This will result in the maximum force emission from the obstacle and the

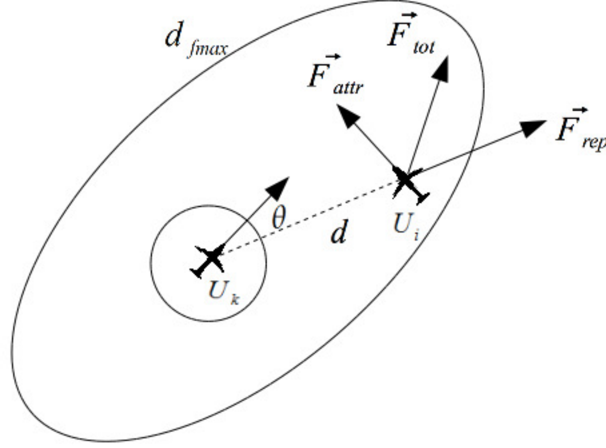


Figure 5.3: Calculating the total force acting on a UAV

maximum force felt by the aircraft. Conversely, if two aircraft are heading directly away from each other, the force emitted and the force felt will be at a minimum.

After calculation of each repulsive force, the values are combined into one value $\vec{F_{rep}}$ using Equation 5.5.

$$\vec{F_{rep}} = \sum_{k=1}^{n} s(\theta_k, \phi_{repk}, d_k) \tag{5.5}$$

This value is finally used with Equation 5.1 and an associated force for the attraction to the goal waypoint. In this implementation, a constant force value is used for the attractive force felt by the aircraft such that $|\vec{F_{attr}}| = 100$. At this point the algorithm calculates the total force, as is shown in Figure 5.3, and uses it to determine where the aircraft should go for the next time step. If the angle of $\vec{F_{tot}}$ is outside the maximum turning angle of the aircraft, it will tell the aircraft to turn as far as it can towards $\vec{F_{tot}}$, but if the angle is within the maximum turning angle of the aircraft it will instead tell that aircraft to turn to that matching angle.

Figure 5.4: Adjusting the repulsive force to handle head-on collisions

5.3 Special Cases

As previously mentioned, artificial potential fields is a relatively easy algorithm for collision avoidance, both in theory and in execution time. However, it does suffer from a few special cases that cause aircraft to follow courses that don't make sense or are longer than necessary. The follow sections detail some of these special cases and how they are addressed in this implementation.

5.3.1 Handling Deadlocks

One of the special cases with this algorithms involves a deadlock scenario where two aircraft are heading towards each other and their destinations are behind the other aircraft [9]. In this situation, the attractive and repulsive forces will be directly opposite of each other and the aircraft will simply fly straight into each other. In order to detect this problem, the attractive and repulsive forces are converted into unit vectors and added together. If the unit vectors cancel each other out, then the deadlock situation is detected. In order to resolve the deadlock situation, the aircraft are given directional changes of 15° to the right so they can safely pass each other [9]. The deadlock situation and simulation of the resolution are shown in Figure 5.4.

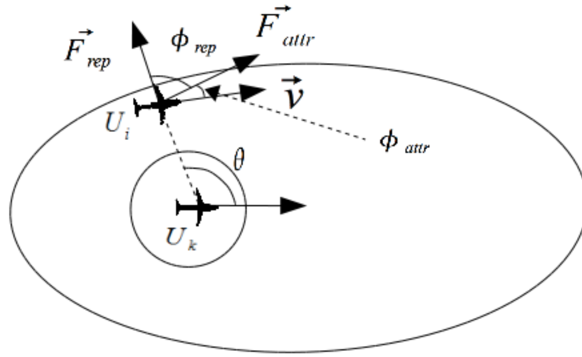


Figure 5.5: Situation in which the right hand turn rule should not be applied. U_i should continue towards its destination instead of traveling behind U_k.

5.3.2 Right Hand Rule

Another tweak to the algorithm involves scenarios where it is faster to cross paths behind another UAV instead of in front of it. In some cases, if this doesn't happen, a deadlock will occur for an extended period of time. In order to fix this problem, additional rules had to be applied to modify the direction of force vectors felt by other aircraft in these situations.

In order to detect these special cases, three values are used. The first value used is θ, the angle between the bearing of another aircraft, U_k, and the location of the current aircraft, U_i. If θ is found to be on the interval [-135, 0], then U_i is to the left of U_k. Next, the algorithm calculates ϕ_{rep}, the angle between the current bearing of U_i and the repulsive force acting on U_i from U_k. If this angle is between [-180, -90], then U_i is attempting to turn left to avoid U_k. However, this could place U_i on the path in front of U_k and potential create a hazardous and/or deadlock scenario. Finally, ϕ_{attr} is defined as the angle between the bearing of U_i and the attractive force from its goal. If $\phi_{attr} \leq 0$ and $\theta < -90$ then the situation is fine because U_i is both to the left of U_k and turning left away from a collision as shown in Figure 5.5.

If θ is on the interval [-135, -25], then a right turn should be used such that U_i passes behind U_k. In this situation, the angle ϕ_{rep} is actually flipped across the length of the aircraft

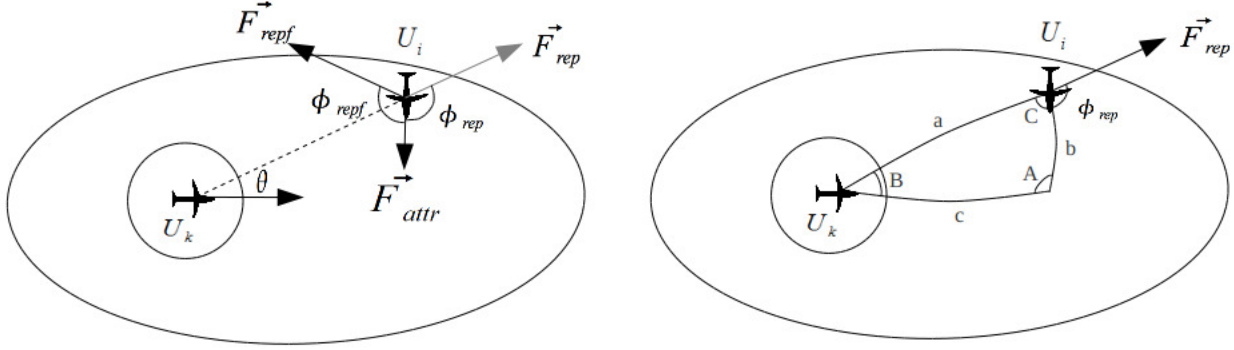


Figure 5.6: Finding the new repulsive force vector in order to travel behind U_k

to ϕ_{repf} which is shown in Figure 5.6. This will guild U_i behind U_k to avoid both a collision and a deadlock scenario.

The last interval is when θ is bound by [-25, 0]. For this situation, U_i is still to the left of U_k, but we need to use a spherical triangle to determine the appropriate course of action. Referring to Figure 5.6, a is defined as the great-circle distance between U_i and U_k. Then, b is the great-circle distance between U_i and the point of intersection with U_k while c is the great-circle distance between U_k and the point of intersection. b and c and calculated using the spherical law of cosines and are used as a basis for determining if the aircraft should turn. If $(c - b) \leq (-\phi_{rep} - 90)$, then the right hand turn is active just like if θ was on the [-135, -25] range. If this doesn't happen, it simply means the distance c is large enough that U_i can cross U_k's path and while maintaining safety. Generally speaking, if θ is in the [-25, 0] range, U_i will simply pass in front of U_k.

5.3.3 Aircraft Priorities

One of the noticeable problems with artificial potential fields is that as an aircraft approaches its goal waypoint, it can be pushed off course by other aircraft that are too close. This can have several negative effects such as unnecessary turns, being thrown way off course, or getting stuck in a loop around the waypoint. Eventually, the interfering aircraft

will fly away from the waypoint but not after potentially causing some serious problems for the main aircraft.

In order to address this issue, the algorithms prioritizes the aircraft based on how far it is from its destination. If it is within $d_{priority}$, defined as $d_{priority} = 4.5 * d_{onesec}$, of its goal, the aircraft will become a prioritized. For an aircraft of priority m, the repulsion forces of only aircraft of higher priority are factored into the summation. This is best represented by Equation 5.6 which takes the priority system into account by only factoring in aircraft of a higher priority when calculating $\vec{F_{rep}}$.

$$\vec{F_{rep}} = \sum_{k=1}^{m-1} s(\theta_k, \phi_{repk}, d_k) \tag{5.6}$$

Unfortunately, this doesn't completely solve the problem because another problem is introduced. Considering any situation where one aircraft is within $d_{priority}$, only one of the two vehicles will attempt to avoid the other due to the priority system which can lead to problems because normally both are trying to avoid a collision. To handle this side effect, aircraft that are prioritized have an expanded potential field that can be defined as $p_{mult} * d_{fmax}$. For this implementation, a value of $p_{mult} = 1.2$ was used when calculating the force fields of prioritized aircraft.

5.3.4 Aircraft Looping

The final problem major problem accounted for with this algorithm is the issue of looping. Consider two sequential waypoints that are close to each other. It's possible for the aircraft to arrive at the first, but then get stuck in a loop because it cannot turn sharp enough to reach the close waypoint. If this happens, the aircraft will continuously loop around the waypoint indefinitely because of the strong attractive force emitted by that waypoint.

In order to solve this problem, a method of detection and a method of correction must both be defined. In order to detect it, some basic geometry is used along with some of our pre-defined aircraft constraints. In this system, the aircraft is consider to have reached

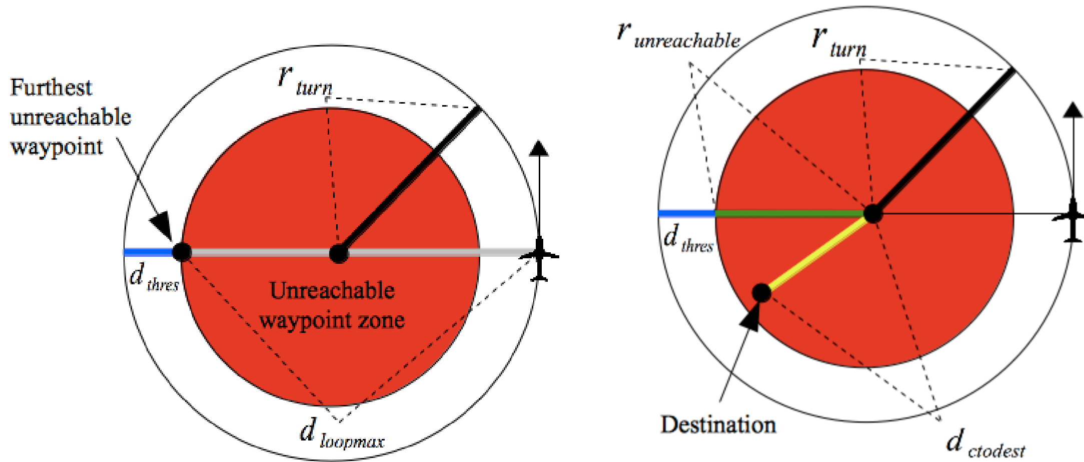


Figure 5.7: Geometry involved to detect a looping condition.

a waypoint if it is within a threshold distance, d_{thres} of the goal. Additionally, there is a maximum turning angle that constrains the aircrafts' physical movements. With these two values combined, you can create an unreachable area as shown in Figure 5.7. In this figure, the white zone is reachable by a turn, but the area in red cannot be reached with a normal turn maneuver. By using the pre-defined values, this circle of unreachable waypoints can be easily defined as a point, c, and a radius, r_{turn}.

First, there is no reason to check for this condition unless the aircraft is close to the destination. In fact, this distance can be defined as $d_{loopmax} = 2 * r_{turn} - d_{thres}$. If this condition is met, the algorithm calculates the value of $d_{ctodest}$ which is the distance between c and the destination. If $d_{ctodest} \leq r_{unreachable}$ then the aircraft needs to correct itself to prevent looping. The simplest way to perform this correction is to change the waypoint's attractive force into a repulsive force until the condition is no longer true.

Chapter 6

Test-bed Design

6.1 Test-bed Requirements

One of the key goals of this research is to create a standardized way to test and compare collision avoidance algorithms based on a set of metrics. To accomplish this goal, there needed to be a system that would be standardized regardless of which collision avoidance algorithm was currently active. Additionally, this system needed to work with a pre-existing, Java-based control system that was capable of sending waypoints to a UAV and receiving GPS data from that UAV. For testing purposes, the control system needed to have the ability to perform simulations. This feature would allow a researcher to perform basic tests and error checking in a laboratory setting before risking the hardware components of a real UAV. There also needed to be methods to both store and potentially visualize the data collected from these flights such that analysis could be performed on the collected GPS data. In summation, this system would need to meet the following requirements:

1. Standardization of the control system
2. Ability to easily switch collision avoidance algorithms
3. Integrate into pre-existing control software
4. Ability to perform laboratory simulation
5. Collect and visualize GPS data
6. Provide an interface for loading waypoints, paths, or courses into the system

6.2 Pre-existing Hardware Configuration

6.2.1 Base ArduPilot Configuration

This system is actually a continuation of the AU Proteus Project [12], a project aimed at research in the field of controlling UAVs from the ground. Because of this, many of the hardware design decisions were carried over from the previous research. For hardware, the UAV used is the Multiplex EasyStar system in combination with an Arduino based autopilot known as ArduPilot. The EasyStar is composed of several aerial components: the UAV body, servos for control, electronic speed controller, ArduPilot board, ArduIMU board (used for stabilization), an XBee antenna, the manual receiver, and a power source. Fortunately, there are step-by-step instructions readily available online for assembling the EasyStar along with all of the Arduino components [1, 2]. By simply assembling the unit with the online instructions and using the default software provided, these hardware components will allow you to manually fly a UAV and autonomously fly that UAV on a pre-determined course while maintaining stability.

6.2.2 AU Proteus modifications

The major change to the core system was the addition of a XBee antenna in order to relay messages back and forth between the UAVs and the ground station. The XBee was attached to the ArduPilot using extra pins located on it's shield such that any messages to and from the ground station could be relayed [12]. We were able to use pin 4 on the shield for our data output and read incoming packets by using the extra A4 pins that also provided power to the XBee. Each UAV is equipped with its own antenna and each antenna has a unique identifier for recognition by the whole network.

6.3 Pre-existing Software Configuration

6.3.1 ArduPilot Modifications

Because of the addition of the XBee antenna, some modification had to be made to the actual ArduPilot code. Mainly, there had to be functionality in the code to pass on any new GPS data along with the ability to receive waypoints while flying and switch the autopilot to go to the new waypoint [12]. This modification serves as the core communication unit to the UAVs during flight and provides our ground station with any relevant data regarding position and speed to perform control functions on the UAVs. Another thing to note about these modifications is that they're fairly general modifications for all ArduPilots. As long as you have the proper hardware configuration and the correct code additions, this wouldn't be limited to the EasyStar setup that was used in these specific tests.

6.3.2 AU Proteus JAVA Controller

With the autopilot sending relevant messages to the XBee, the system needed a way to receive those messages at a ground station for processing. For this, the pre-existing system had a JAVA program running that allowed users to monitor incoming data and send single waypoints to the program [12]. On the ground station, there is another XBee antenna plugged in via USB that will monitor the data being sent from any UAVs. The JAVA controller is polling this XBee for any packets from the UAV and processes them for the user upon receiving one. This program is one of the core programs used in the new system because it was a standalone component providing the necessary communication channels to the UAVs. For specifics on this original controller, please review the references provided [12].

6.4 Robot Operating System

One of the major design decisions for this system revolved around the fact that there was already a pre-existing program for ground-to-air communication. In order to avoid

reinventing this software, the new system needed to either build on that software or find a method to compartmentalize it so it could easily communicate with it. Fortunately, the Robot Operating System (ROS) framework provided a simple solution to this problem. ROS is commonly used on robots in order to divide up tasks amongst the various components of the device such as sensors, motors, controllers, and input/output [4]. In fact, many components commonly used for research projects (such as the roomba or the kinect) have standard communication protocols in ROS. This allows for developers or researchers to easily create methods for controlling each part of a complete robot while simultaneously isolating those parts into smaller subprograms.

6.4.1 Nodes

In ROS, the three main things to know about are nodes, messages, and services. A node is basically any subprogram or process within an entire system [4]. Usually, each node is responsible for one particular task and can share information regarding that task through messages and/or services. Because of this partitioning of tasks, many ROS based programs typically have several nodes to cover each aspect of a given program. The node structure also allows for easy portability or swapping of nodes. For example, if I have two different nodes for steering a robot, as long as both adhere to the same communication protocol, they can steer completely differently but still fill the same node slot in the overarching system.

6.4.2 Messages and Topics

A message within ROS is effectively a broadcast to any nodes listening and is considered a many-to-many form of communication [4]. These messages get posted to a topic and anyone subscribed to that topic will receive the message when it gets posted [4]. It's important to note that there can be multiple publishers and subscribers for each topic and that each node has it's own way to handle a message when it is received on that topic. Typically, messages are used for components of the system that need to operate periodically such as a sensor

reading or status messages [4]. When setting up a system based on ROS, the developer must carefully plan the data components of the messages such that the information within that topic makes logical sense. Then, each node using the topic must be aware of the format in order to pass the proper messages to each topic.

6.4.3 Services

A service is instead a one-to-one communication system between two nodes where there is both a request and response component to the communication [4]. Like with messages, you can specify the data in both a request and response such that any developers know what needs to be filled out when invoking a service. In contrast to messages, most services tend to run at irregular intervals and tend to be specific to a particular node. The request/response format also allows for bi-directional communication between two nodes instead of a global broadcast with no direct response. Finally, two nodes can't advertise the same service, but any node can invoke a service call to another node [4].

6.5 Core Control Nodes

In Figure 6.1, the overall architecture of the system is represented in a visual manner. Within this system, there are three core nodes that are necessary for control of the UAVs in flight: X-Bee IO, Coordinator, and Control Menu [5].

6.5.1 X-Bee IO

With the movement of this system from a standalone program into ROS, the pre-existing Java controller needed to be ported into ROS. This mostly involved adding callbacks to forward messages received from the UAVs into the position updates topic and forwarding commands from inside the ROS system out to the UAVs. Because of this, the X-Bee IO node acts like a translator from the message format that the UAVs use into a internal format understood by ROS. For example, latitude and longitude on the UAVs are multiplied by one

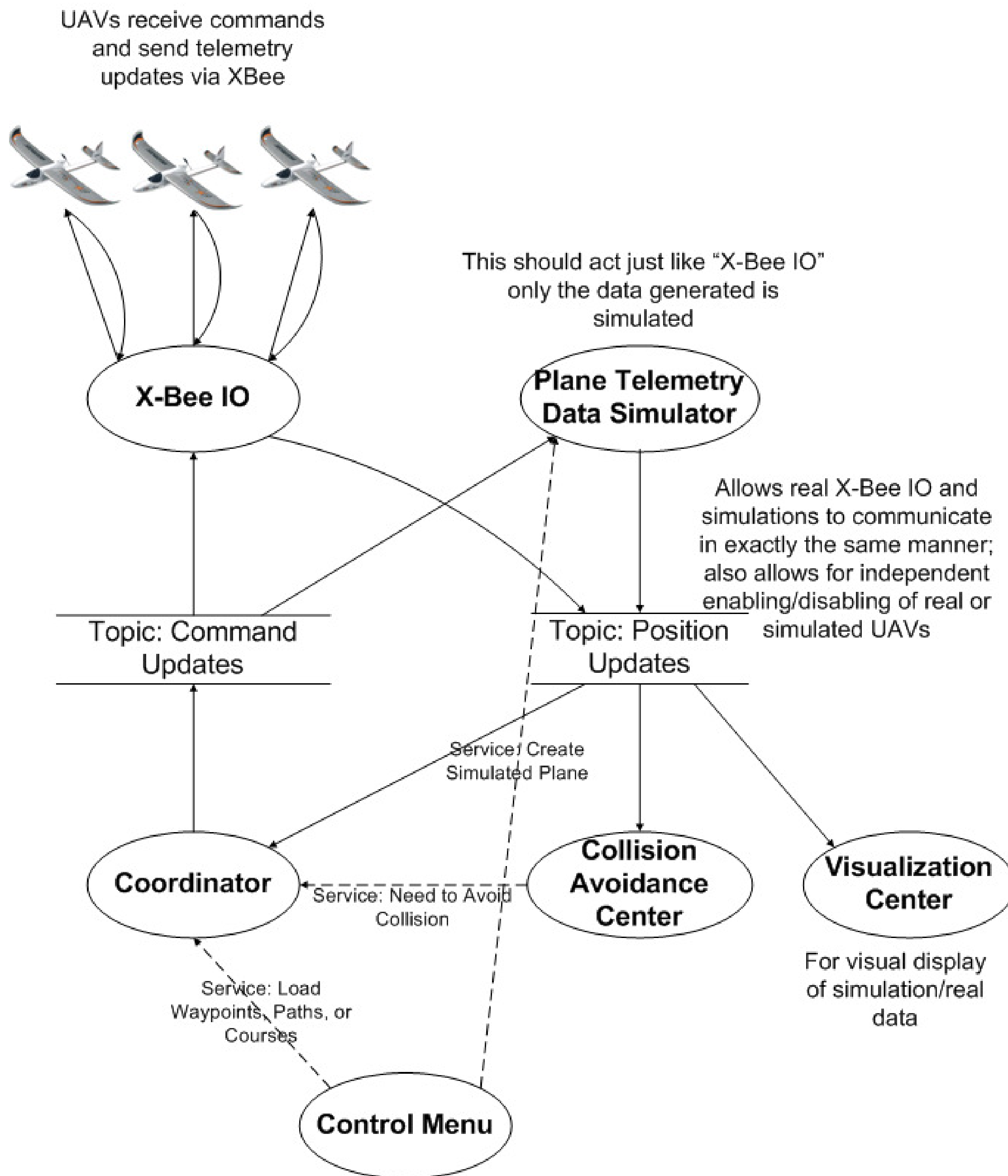


Figure 6.1: The layout and interactions of ROS nodes in this system. Nodes are ovals with services represented by dashed lines and messages by solid lines [5].

million and stored as integers [2]. When that information gets passed through X-Bee IO, it is divided by one million and stored as a double such that any internal operations are performed on the real latitude and longitude values. Because of this translation functionality, this node is basically a bridge from internal control messages to the real world UAVs.

6.5.2 Coordinator

The second major control node is the Coordinator, a node that, in many ways, can be thought of as the core control node within this system. When a user has provided a waypoint, path, or course for the UAV(s) to go to, the coordinator is the part of the system storing this information for later use. Whenever it detects that the UAV has reached its current destination (based on a new telemetry message), it will remove that waypoint from its destination queue and send a command to go to the next waypoint in the queue. In does this by monitoring the telemetry topic and checking messages from that topic against the destination queue for that UAV. This automates a lot of the control functionality for the user. When managing multiple UAVs, the coordinator keeps independent queues for each UAV. In addition to the normal destination queue, each UAV has a second prioritized queue referred to as the collision avoidance queue. If this queue is empty, then the UAV will act like it normally would in going from destination to destination. However, when a waypoint is placed in the collision avoidance queue, the Coordinator will immediately send that waypoint to the UAV because it takes priority over a normal destination waypoint. This prioritized queue system is the way collision avoidance waypoints are quickly conveyed to the UAVs. Since this is a priority queue, it also allows some collision avoidance algorithm to plan a list of waypoints in between destinations in order to insure UAV safety.

In addition to the monitoring aspect of the coordinator, there is also a wide range of services in place to keep the system running smoothly. In order for the user to load a single path or (for multiple UAVs) a course, there are callbacks in place that simply hand the coordinator a filename and it extracts all of the necessary waypoint information from

that file. This allows users to pre-plan a path or course and simply load it with ease at runtime. There are also callbacks for nodes to add waypoints to either queue or to even empty a queue. In short, the coordinator handles all the syntax with sending commands while providing services to other nodes so the coordinator can be updated on what waypoints need to be sent in those commands.

6.5.3 Control Menu

In order to manage this system, there needed to be a node running that the user could interact with. This is where the Control Menu comes in by providing a basic user interface for all of the services provided by the other nodes. It's a fairly simply terminal based menu system for a user of the system to interact with. The first functionality it provides is just control over the waypoints being stored in the coordinator. There are menu options for loading a path, loading a course, or simply sending a waypoint to a specific UAV. In addition to the services associated with the Coordinator, there are also services associated with the Simulator such as the creation and deletion of simulated planes within the environment. One of the nicer features of the control menu is being able to automatically create simulated planes to match a course file. For example, if a user decides to load a course file containing points for UAVs 1, 3, and 4, the user can also specify that simulated UAVs with those IDs should be created if they don't already exist.

6.6 Core Message Types

The messages compose the communication backbone of this system. When UAVs are flying, these topics are typically very active due to the need to receive real-time UAV data and send back responses in an expedient manner.

6.6.1 Telemetry Updates

Within this system there are two major message structures with the first being an actual telemetry update. The telemetry update message is meant to be a simple message containing any data acquired by the UAV that can be used for coordination, path planning, or visualization. As of now, this core data includes longitude, latitude, altitude, plane ID, destination waypoint, ground speed, bearing to target (current destination), and distance to target. The only nodes planned to publish to this topic are X-Bee IO which has real telemetry messages and the Simulator which generates fake messages for testing purposes.

6.6.2 Commands

The second major message type is the command. Command messages are generated in order to tell UAVs where their current destination is (regardless of being a normal destination or collision avoidance destination). Within this system, we want to limit the publisher of this topic to the Coordinator only. The reason for this is to insure that the Coordinator is aware of any current commands to the UAVs. In fact, if the Coordinator detects that the current destination of the UAV doesn't match what's in it memory, it will immediately send a corrective command. Only X-Bee IO and the Simulator currently subscribe to this topic.

6.7 Research Nodes

With the core nodes and communication channels in place, development of nodes tailored towards research can be added to the system. These nodes are meant to be modified by researchers in order to test a wide variety of components such as simulation, collision avoidance algorithms, and data representation.

6.7.1 Simulator

Since one of the major functions of this system is to test and perform collision avoidance algorithms, there needed to be a cheap (in terms of hardware cost) method to test algorithms

without running the risk of destroying equipment. For this reason, a simulator was created to work within the system. In order to maximize the simulation, the Simulator was written to act just like the X-Bee IO would in terms of how it communicates with the coordinator. This means that the Simulator subscribes to the same command topic and publishes to the same telemetry topic. One of the interesting results from this setup is the fact that to the coordinator, there is nothing distinguishing a real UAV from a simulated one so you could have both real and simulated UAVs flying simultaneously. As mentioned before, there are services within the Simulator that allow a user to create UAVs at specified waypoints and then delete them later as well.

As far the simulation functionality goes, the current simulator was designed to be very simple representation of UAV activity. The simulator takes in an estimated cruise speed and will calculate a telemetry update based on that speed alone. However, it does allow for researchers to set a maximum turning angle in order to impose some physical limitation on the simulated UAVs. For the testing results presented later, a turning angle of 22.5°. This means over the course of 4 seconds, the UAV can make a 90° turn. It's important to note that this basic simulator doesn't factor in other conditions such as weather, wind resistance, or other environmental elements.

6.7.2 Collision Avoidance

One of the key reasons for this project was to provide a way to test some of the collision avoidance algorithms proposed by various researchers. Because of this, a shell node was created to demonstrate how collision avoidance can be obtained within this system. The idea behind collision avoidance within this system is to monitor the telemetry data received and correct the UAV's path as necessary in order to avoid collision or conflicts in the air. The shell houses no algorithm for collision avoidance but serves as a model for how to implement a collision avoidance algorithm. The three implemented algorithms were built off this shell by taking the callback and filling those callbacks with algorithmic processes.

6.7.3 Visualization

Another area for expansion of this project lies in visualizing the data received from the UAVs. As of now, there is no functionality for viewing this data real-time within the system. However, there is a node provided with the system known as the KMLCreator. The KMLCreator will monitor any telemetry data received and will save all of that data in a kml format which is support by Google Maps [3]. This is useful for visually seeing the paths of UAVs after a course (real or simulated) has been run.

Chapter 7

Results

In order to compare the different algorithms, five metrics were used as the primary methods of analysis. The first four metrics were used to determine the effectiveness of the algorithms. These metrics were number of conflicts, number of collisions, aircraft survival rate, and average time of death of the aircraft in the test. The final metric is the increase in the number of waypoints achieved. This metric was created to determine the efficiencies of the algorithms.

In order to perform a variety of stress tests, two major variables were modified: number of aircraft on the field and the size of the field. For the number of aircraft on the field, the base value was four aircraft with each subsequent test doubling that number until the maximum value of thirty-two aircraft was met. For the field, all tests were performed on square fields of both 500x500 meters and 1000x1000 meters. Finally, for each combination of number of aircraft and field size, three randomly generated courses were used for telling the aircraft where to go. This led to a grand total of twenty-four simulations per algorithm. Additionally, each simulation ran a total of 10 minutes (600 seconds) which meant four hours of simulation per algorithm. The base case of a system with no collision avoidance was also executed with these same simulations and is included in the results below.

These tests present a wide range of stressful situation to these algorithms. First, on a small 500m field, it's fairly reasonable to fly four aircraft, but becomes difficult to fly any more than that in such a constrained airspace. The 1000m field offers more flexibility, but even that airspace becomes stressed when 32 aircraft are flying around it. Second, the variable number of aircraft helps to show how efficiently the algorithms can calculate viable solutions for a large number of aircraft. Without efficiency, the solutions will not reach

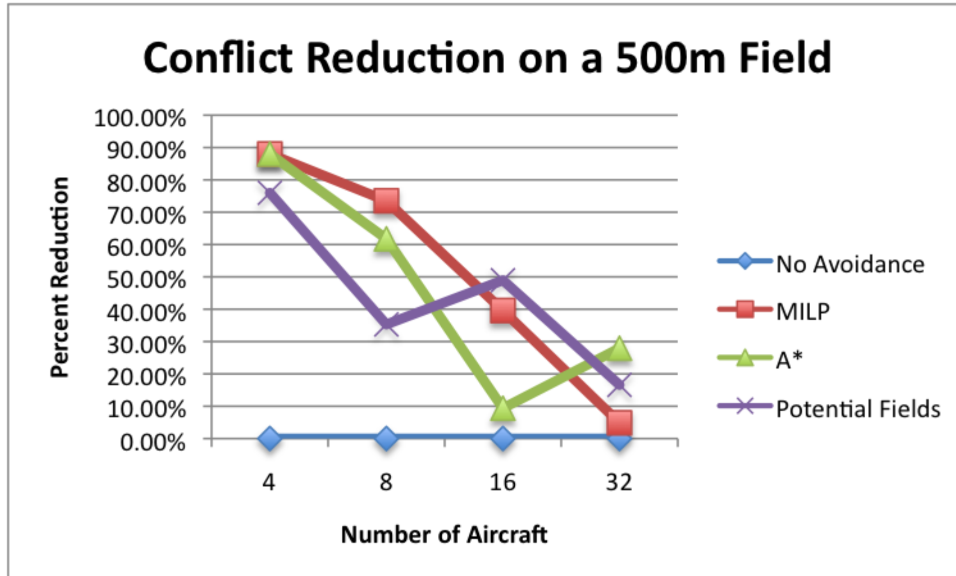


Figure 7.1: Average conflict reduction on a 500x500 meter field

the aircraft in time which may result in conflicts and/or collisions. Finally, the randomly generated courses present the challenge of flightpaths that may not make sense. Typically, the aircraft would have goals such as mapping a field, scanning an area for personnel, or even just flying to specific points on the field. However, in these tests, there is no logical organization to the waypoints which can create unique problems for these algorithms.

7.1 Number of Conflicts

As described in the problem statement, a conflict is detected when two aircraft are within two time steps of each other. In these situations, imminent danger is present because while a collision has not occurred, there is a high probability of one occurring within the next time step. Since each time step in these tests are one second and the flight speed is constant, a conflict zone can be safely defined as a circle with a radius of 24 meters.

Figures 7.1 and 7.2 show the average percentage of conflict reduction that occurred across the tests for each algorithm. For the 500m tests, each algorithm showed over a 75% improvement over having no collision avoidance. However, all three algorithms struggled to keep conflicts low as the number of aircraft increased. The maximum reduction with

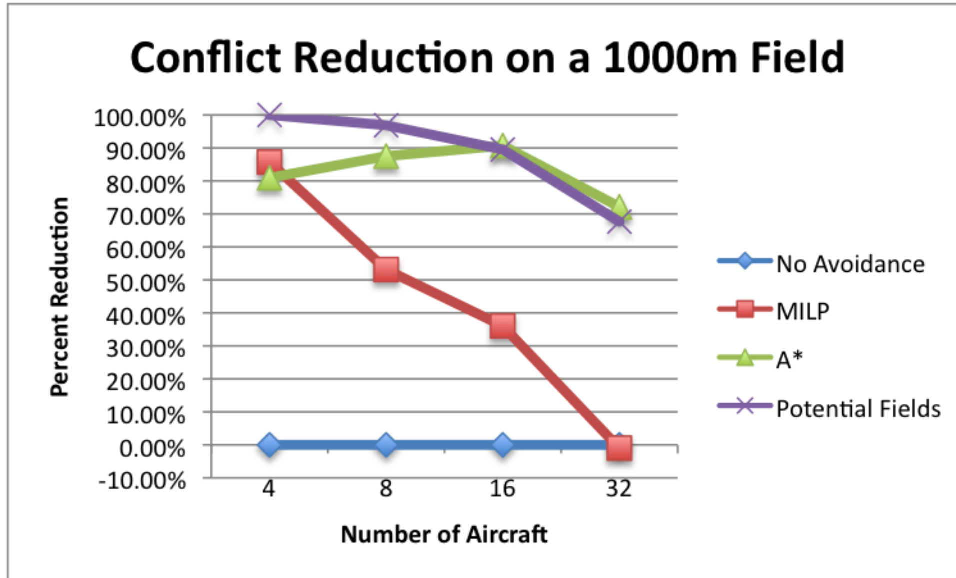


Figure 7.2: Average conflict reduction on a 1000x1000 meter field

thirty-two aircraft was under 30% and MILP showed almost no decrease in the number of conflicts on the field.

On the 1000m field, you can see definite improvement in the number of conflicts for all three algorithms. Specifically, both A* and artificial potential fields kept conflicts to a minimum even at sixteen aircraft on the field with potential fields having no conflicts with only four aircraft on the field. Even at thirty-two aircraft, both algorithms showed significant improvement by reducing the number of conflicts by approximately 70%. MILP performed well with only four aircraft, but only showed minor improvements at eight and sixteen. In addition, there was no improvement in MILP compared to the base case at thirty-two aircraft.

7.2 Collision Reduction and Survival Rate

The percentage reduction of collisions and the survival rate of the aircraft are almost completely related due to the fact that if a collision is detected, at least two aircraft are eliminated. However, survival rate was included as a statistic because there are some situations where more than two aircraft can be eliminated. Because of these situations, survival rate

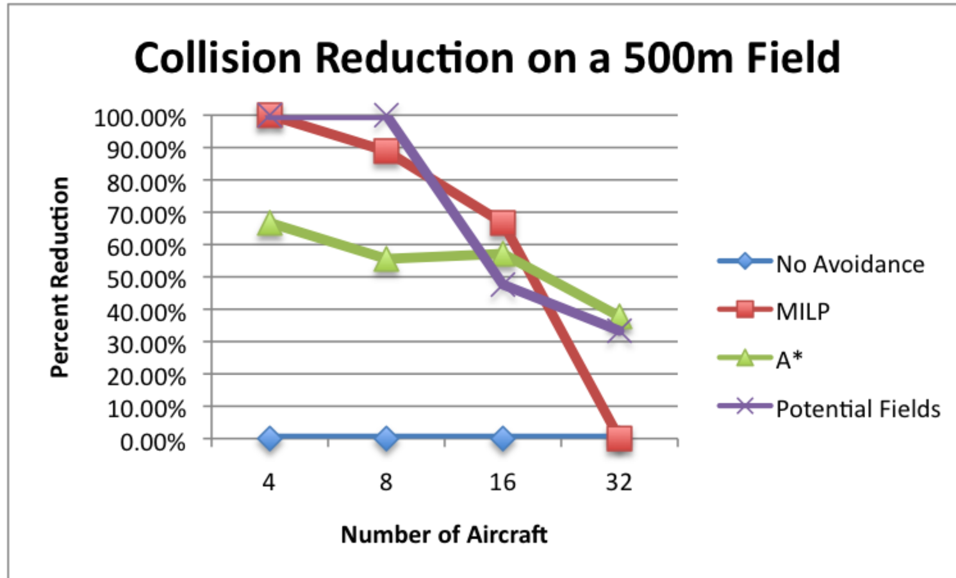


Figure 7.3: Average collision reduction on a 500x500 meter field

is generally a more accurate representation of how well the collision avoidance algorithms performed.

In Figures 7.3 and 7.4, there are some trends in each algorithms' performance and across the board. First, as the number of aircraft on the field increases, the reduction in collision avoidance decreases. This is primarily due to the increasingly crowded airspace. This creates increasingly stressful problems for the algorithms that don't always have an easily generated solution. Additionally, it's important to not that on a 500m field, at least one algorithm was capable of safely managing eight aircraft, and on a 1000m field, at least one algorithm could safely manage sixteen aircraft.

The results from the survival rate calculations are shown in Figure 7.5 and 7.6. Note that a "perfect" algorithm would have 100% survival rate for each feasible simulation. First, consider the base case simulations. In almost every simulation on a 500m field, the base case would have collisions until approximately two aircraft were left alive. At this point, the chances of the two aircraft flying into each other became minimal and they typically survived the simulation. The same trend can be seen on a 1000m field, but there were more

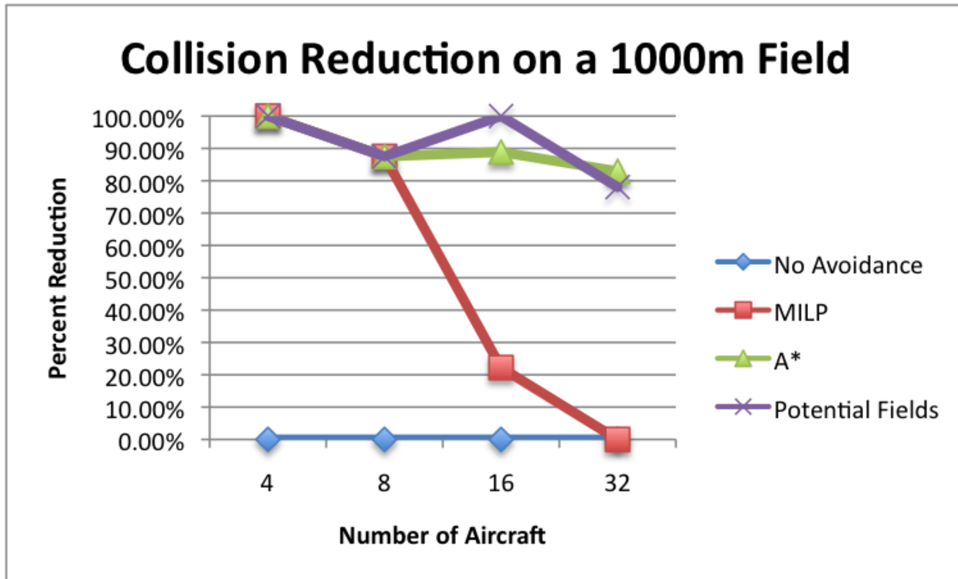


Figure 7.4: Average collision reduction on a 1000x1000 meter field

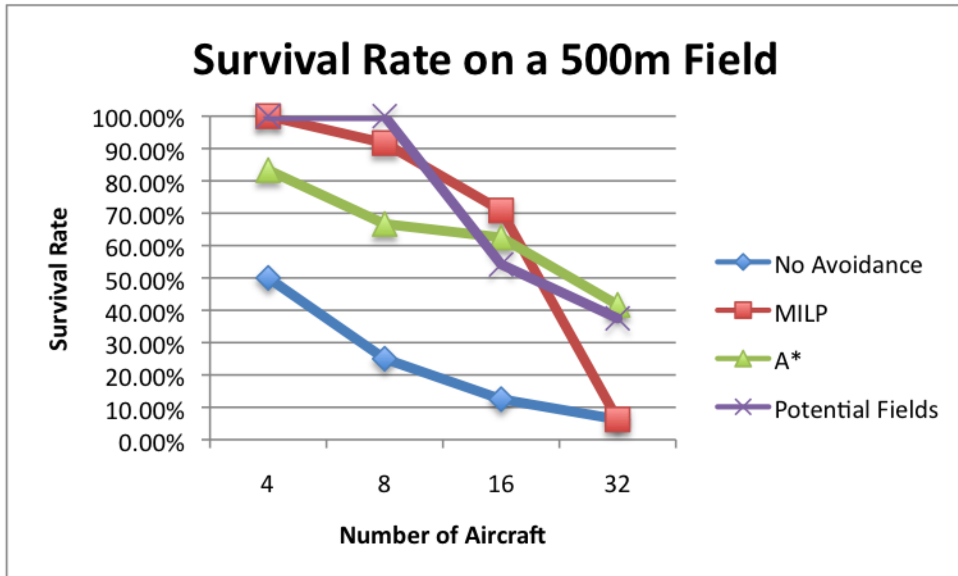


Figure 7.5: Average survival rate on a 500x500 meter field

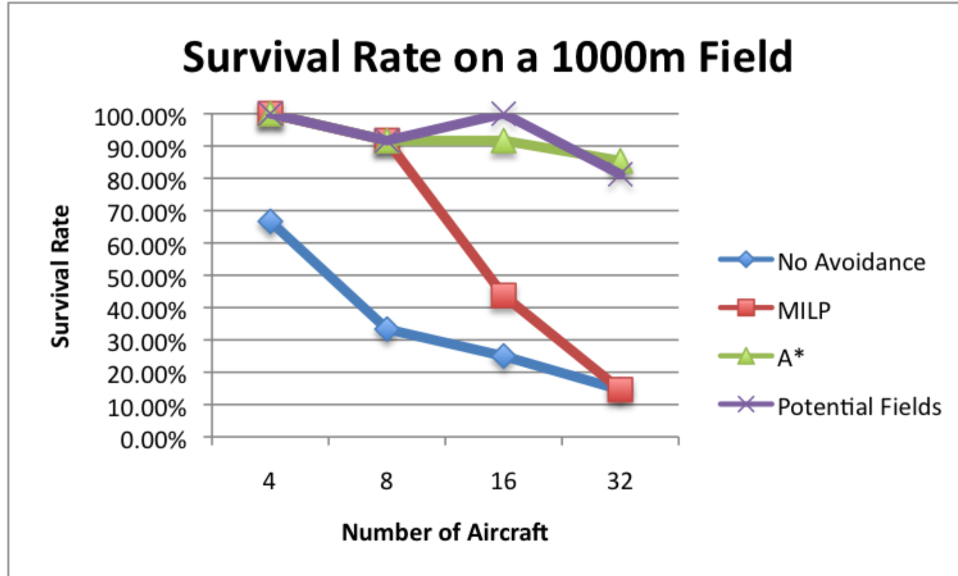


Figure 7.6: Average survival rate on a 1000x1000 meter field

cases where up to four aircraft were left alive at the end of the simulation due to the larger airspace.

Looking at the survival rates, it's apparent that the most stressful test was, as expected, thirty-two aircraft on a 500m field. For this scenario, the best algorithm averaged 41.67% survival rate or 13.3 aircraft. Additionally, at the sixteen aircraft simulations, these algorithms had up to a 70.83% survival rate or 11.3 aircraft.

The simulations involving four and eight aircraft on a 500m field were more realistic. Both MILP and artificial potential fields had 100% survival rate for four aircraft on a 500m field. A* had only one collision one of these simulations which dropped its survival rate. At eight aircraft, artificial potential fields still maintained 100% survival but both MILP and A* had a slight drop in survival rate. From here, all algorithms deteriorated in survival with MILP having almost no improvement in survival over the base case in the thirty-two aircraft scenarios.

On the 1000m fields, many of the same trends are reflected. However, because of the significant increase in airspace, many of these trends are less drastic. The vast airspace allowed all three algorithms to perform perfectly with only four aircraft on the field. At

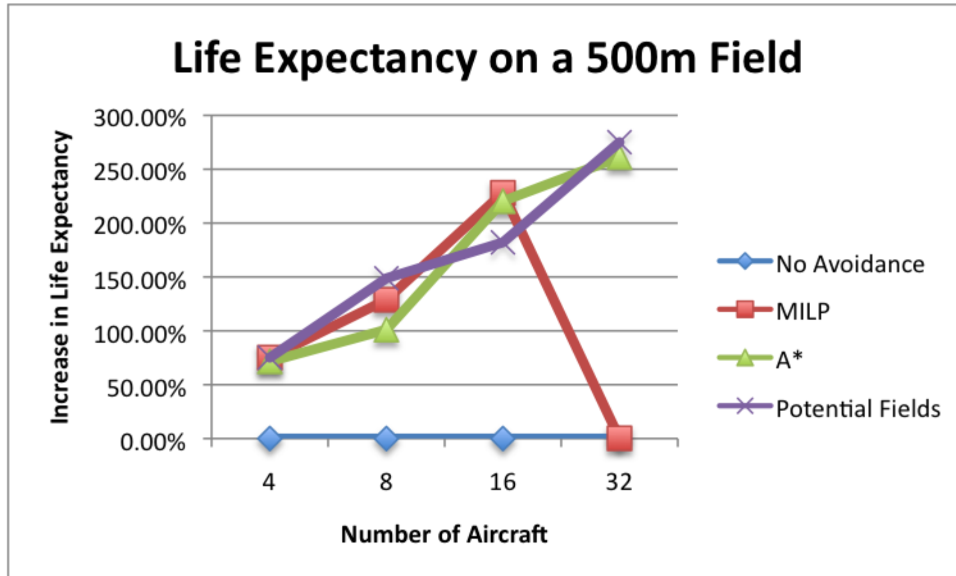


Figure 7.7: Average increase in life expectancy on a 500x500 meter field

eight aircraft, each algorithm lost a total of two aircraft across all three simulations. A* and artificial potential fields proceeded to work well with even 16 and 32 aircraft on the field with both algorithm staying above 80% survival rate. Unfortunately, the same cannot be said for MILP which had drastic drops in survival rate at both sixteen and thirty-two aircraft.

7.3 Aircraft Life Expectancy

The life expectancy statistic was meant to serve as a statistic demonstrating the average flight time of each aircraft in these simulations. For life expectancy, the simulation with no avoidance was used as the base case with Figures 7.7 and 7.8 showing the actual increase in the aircrafts' flight time.

The charts show that the algorithms were almost all effective at all levels in increasing the life of the aircraft in the simulations. Even on a small 500m field, the aircrafts' flight time was increased anywhere from 50-275%. The growth in life expectancy can be partially attributed to the fact that as the number of aircraft go up, the flight time of aircraft in the base case goes down. Both A* and artificial potential fields show significant and comparable

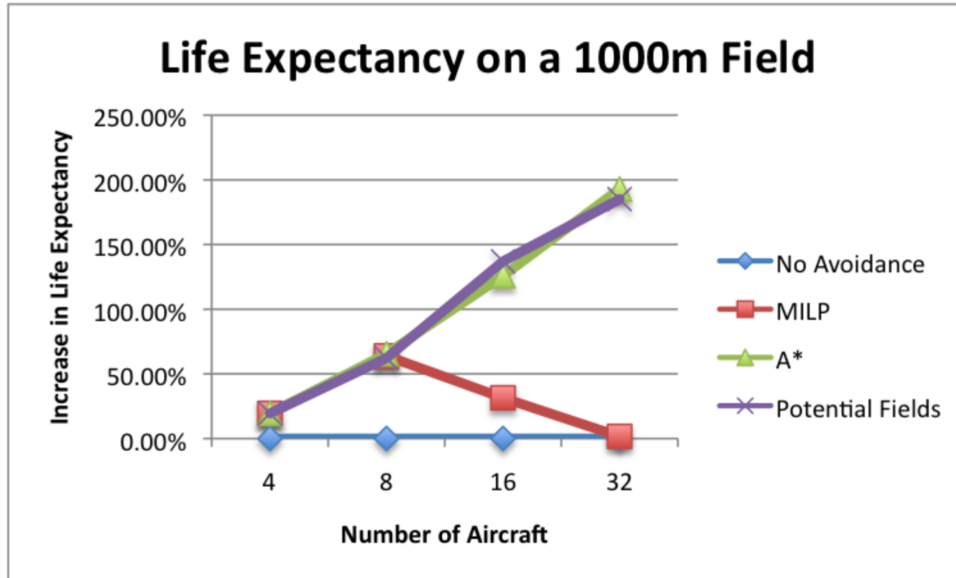


Figure 7.8: Average increase in life expectancy on a 1000x1000 meter field

improvements in life expectancy at all levels, but MILP struggles with a large number of aircraft in the simulations.

7.4 Efficiencies of the Algorithms

The primary metric for measuring efficiency was the increase in the number of waypoints achieved during the simulation. For the base cases, the raw value was typically lower due to the fact that a large number of aircraft would collide during the simulation. This would prevent those aircraft from reaching future waypoints, so in many ways, this metric is also related to the effectiveness of the algorithms. Additionally, the raw numbers are typically smaller on a large field due to a larger distance between waypoints. Because of this, the percent increase in the waypoints achieved is the metric used. In theory, the algorithms should increase the waypoints achieved by keeping the aircraft from colliding which would allow each individual aircraft to reach more waypoints.

Figures 7.9 and 7.10 show the increases for each algorithm. Note that the base case is also represented as a straight line across 0%. Generally speaking, the increases are larger on the smaller field. The reason for this lies in the fact that on a smaller field, the aircraft are

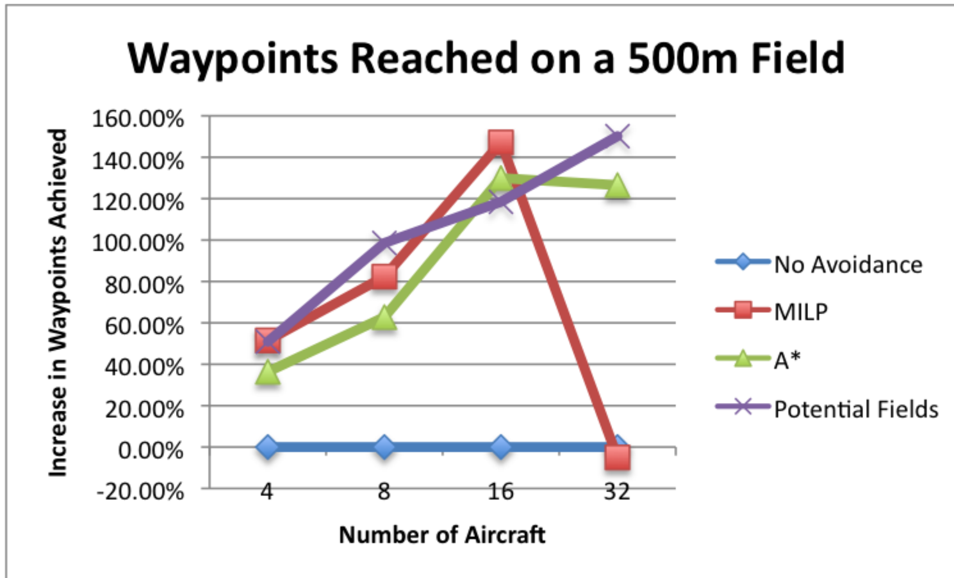


Figure 7.9: Average increase in the number of waypoints achieved on a 500x500 meter field

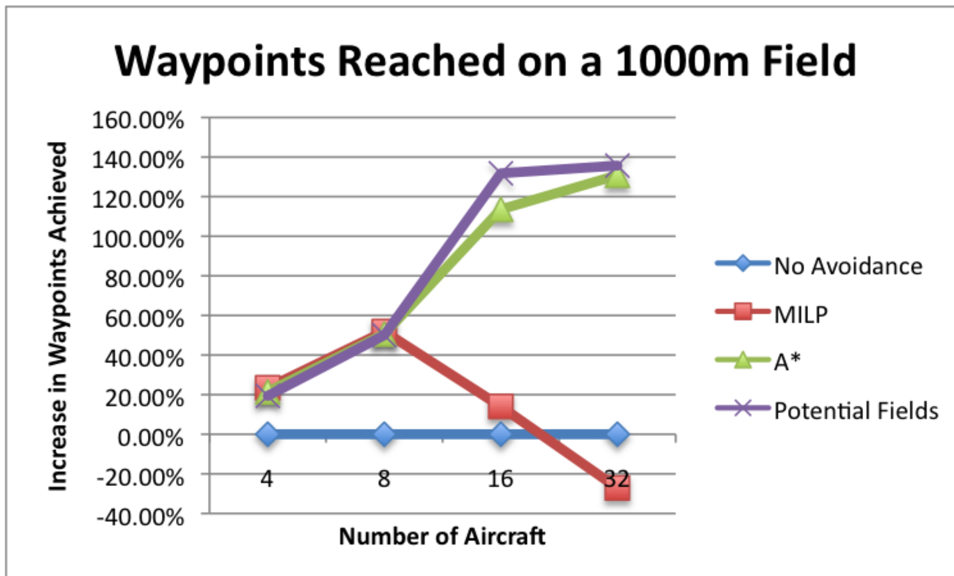


Figure 7.10: Average increase in the number of waypoints achieved on a 1000x1000 meter field

more likely to collide. Without any collision avoidance, the aircraft tended to collide earlier on the 500m field which allowed for greater improvements across all algorithms. Generally speaking, all three algorithms showed improvements with a small number of aircraft on the field. However, MILP began to suffer with a larger number of aircraft and actually achieved less waypoints than the base case with thirty-two aircraft on the field.

7.5 Algorithm Performance

7.5.1 MILP performance

While MILP wasn't the best performing algorithm overall, it still proved that it's capable of working under the more reasonable simulations. In particular, MILP excelled at solving problems with only four aircraft. It had the lowest number of conflicts, perfect survival rate, and the best increase in waypoints achieved on both a 500m and a 1000m field. For the eight aircraft simulations, it had a near perfect survival rate in combination with a large time of death implying that the aircraft that didn't survive were in flight for a relatively lengthy amount of time.

Unfortunately, this implementation of MILP struggled to handle the stress test simulations. As described earlier, MILP becomes increasing complex with each aircraft added to the algorithm. In a tight airspace, the use of receding horizon to reduce the problem becomes increasingly inefficient. This implementation struggled to safely navigate sixteen and thirty-two aircraft simulations. This is most noticeable in the 1000m simulations where A* and potential fields maintain a high survival rate whereas MILP's plummets. Additionally, the conflict reduction on a 1000m field helps emphasize the problems that MILP has with keeping the aircraft at a safe distance. While the survival rate and life expectancy are roughly the same as other algorithms at four and eight aircraft, the reduction in conflicts tended to be fairly low when compared to the other algorithms.

Typically, the MILP simulations would fall behind in the calculations until the problem was reduced. In the thirty-two aircraft simulations, the algorithm would be stuck calculating

solutions for previous time steps. Eventually, a solution would be calculated, but it would be received far too late to matter anymore. In fact, the solution calculated at a previous time step may actually cause more collisions due to the mismatch between time steps. During these simulations, this would continue to happen until enough aircraft perished that the algorithm could handle the complex calculations. Unfortunately, this usually happened after a large number of collisions. This problem is reflected in both the survival rate and the waypoints achieved for the simulations. The survival rate obviously plummets at 16 and 32 aircraft. Additionally, the waypoints of the thirty-two aircraft, 500m simulation is low because the algorithm isn't really working until the problem is reduced through aircraft collisions. Unfortunately, this latency tended to lead to a worse performance than if there was no algorithm at all.

In summary, MILP is a very efficient algorithm for manageable problems. The algorithm performed very well when only working with four aircraft. The algorithm also performed well with eight aircraft, but didn't maintain perfect survival rate anymore. Unfortunately, sixteen and thirty-two aircraft proved to be too complex for this implementation on both field sizes due to the inability of the algorithm to reduce the problem into smaller subproblems. While this MILP implementation works well for a small quantity of aircraft, more work needs to be done on reducing the problem size and increasing the algorithm efficiency before it's ready for a large number of aircraft in the airspace.

7.5.2 A* performance

This implementation of A*, DSAS, struggled when compared to the other algorithms on simulations with a small number of aircraft. In particular, both the four, 500m and eight, 500m simulations have a significantly lower survival rate (15-35%) compared to MILP and potential fields. However, at the higher populations, this algorithm was capable of being competitive with potential fields in terms of survival.

DSAS also tended to perform much better on the 1000m field than it did on the 500m field. This is most likely due to the way the algorithm computes paths. On a 500m field, the grid space used for calculating a path is relatively small, but a 1000m field quadruples the number of grids that could be considered occupied or unoccupied. More grid space allows for more room for the aircraft to maneuver which gives DSAS more options when calculating a path. The DSAS algorithm will need to be modified to handle a constrained airspace better even with a small number of aircraft.

On the 1000m field, DSAS proved very capable of maintaining a high survival rate. In fact, the lowest average survival rate was 85.42% at thirty-two aircraft. This is approximately a 70% improvement in survival rate when compared to the base case. Furthermore, when there were collisions, the uptime of those aircraft before the collision were very high with an average time of death always above 500 seconds. While this algorithm wasn't the most efficient, it did provide significant improvements in the waypoints achieved by improving simulations at with sixteen and thirty-two aircraft by over 100%.

On the whole, DSAS provided a significant increase in survivability on a large field. In addition, large field simulations showed that DSAS can provide protection while maintaining efficiency. However, small fields created situations that weren't easily handled by DSAS which often led to collisions. Future work on this algorithm would be to handle some of the situations created by small field simulations and to further improve the survival rates and efficiency of large field situations.

7.5.3 Artificial Potential Fields performance

Artificial potential fields was actually considered the "best" performing algorithm in these specific tests for a few reasons. One of the most prevalent reasons was that it was capable of competing with both other algorithms at all simulation levels in respect to both conflicts and survival rate. On the 500m field, it had a perfect survival rate for both four

and eight aircraft. On the 1000m field, it maintained an almost perfect survival rate through sixteen aircraft and dropped to about 81% for thirty-two aircraft.

The second reason potential fields excelled in the comparison was because it tended to have a higher improvement in waypoints achieved, especially with a high number of aircraft on the field. On almost all simulations, this algorithm had either the highest improvement or next highest in waypoints achieved. While this may be due to the relative simplicity of the calculations, the fact that it can maintain a a high survival rate with this level of efficiency made it the highest performance algorithm in these implementations.

In summation, this implementation of artificial potential fields was comparable or better than the other implementations in terms of efficiency, survival rate, and number of conflicts. As mentioned earlier, there are a couple possible reasons for this result. One is the relatively simple calculation of the algorithm which allowed it to produce a result at all levels of stress. Second, this relative simplicity also allowed for the algorithm to be easily modified to handle several special cases or unique scenarios which may have further impacted. Basically, this algorithm's relative simplicity allowed time for it to be modified to handle some of its problem areas. Of course, this algorithm still is not perfect and will require further testing and modification to handle some of the other special cases generated by its implementation.

7.5.4 Conclusion

While each algorithm has strengths and weaknesses, it's equally important to recognize those traits with respect to the scenarios. Many of the scenarios presented in this research are extremely dangerous even with an exceptional collision avoidance algorithm. From this research, it's apparent that some algorithms are already be capable of flying eight aircraft in a 500m field and sixteen aircraft in a 1000m field despite the randomness of the scenarios. However, if an algorithm is capable of handling this randomness, it should be able to handle more realistic scenarios. The final results from the simulations are shown in Table 7.1. Note

Summary of Results: Best Performing Algorithms								
Field Size	500m				1000m			
Number of Aircraft	4	8	16	32	4	8	16	32
Conflict Reduction	MILP, DSAS	MILP	APF	DSAS	APF	APF	DSAS	DSAS
Collision Reduction	MILP, APF	APF	MILP	DSAS	Tie	Tie	APF	DSAS
Survival Rate	MILP, APF	APF	MILP	DSAS	Tie	Tie	APF	DSAS
Life Expectancy	MILP, APF	APF	MILP	APF	Tie	DSAS	APF	DSAS
Efficiency	MILP	APF	MILP	APF	MILP	MILP	APF	APF

Table 7.1: Table showing the best performing algorithm for each category

that for the 500m, thirty-two aircraft simulations, MILP is excluded from the efficiency section as an outlier due to the problems described earlier.

After analyzing the results from all three algorithms, it's apparent that improvements can still be made to all three algorithms. For MILP, the algorithm needs to be improved to handle high stress situations. Specifically, simulations with a large number of aircraft cause the problem to become difficult for the current implementation of MILP to reduce. Future research on MILP would need to be focused on creating ways to reduce the problem structure so it can return valid collision avoidance paths. Similarly, DSAS (this implementation of A*) currently has difficulties in scenarios that have a small grid space. Future improvements on this algorithm would need to address this weakness and provide alternate methods for handling the small grid size. While this implementation of artificial potential fields has no noticeable, specific weakness, it didn't have perfect test runs even at feasible simulations. This means there are still special cases that aren't addressed by the current algorithm. In fact, all three algorithms most likely have more special cases that would be revealed through further testing of the algorithms.

In conclusion, there are several areas of future research in this topic. The improvements listed above are a starting point for further improvements and comparisons of the algorithms presented in this paper. In particular, both MILP and DSAS have room for a large number of improvements and handling special cases. The next step in development would be to test these algorithms in real-world, feasible scenarios. This presents additional challenges such

as having pilots and people to perform the test, but it would provide more accurate results than a simulator. Once the algorithms are functioning with a high safety level, the final step would be to apply these algorithms to real missions such as surveying an area. With these algorithms in place, survey style missions could be completed by several aircraft with a high level of safety of the aircraft. In this setup, an extra ROS node could be added to handle elements related to the mission such as planning the goals for each aircraft while the avoidance algorithm handles safety. With the growth of UAV technology, these algorithms and systems of management will continue to grow until UAVs can be safely controlled even in complex, stressful scenarios.

Bibliography

[1] Arduimu: Arduino based imu and ahrs. http://code.google.com/p/ardu-imu/wiki/HomePage?tm=6, June 2011.

[2] Ardupilot: Arduino based autopilot with gps. http://code.google.com/p/ardupilot/wiki/ArduPilot, June 2011.

[3] Kml documentation. http://code.google.com/apis/kml/documentation/index.html, June 2011.

[4] Ros.org. http://www.ros.org/wiki/, June 2011.

[5] J. Matthew Holt. Auburn attract project. https://sites.google.com/site/auburnattract/home, June 2011.

[6] O. Khatib. Real-time obstacle avoidance for manipulators and mobile robots. *International Journal of Robotics Research*, 5:90–98, 1986.

[7] D.K. Liu, D. Wang, and G. Dissanayake. A Force Field Method based Multi-Robot Collaboration. In *Robotics, Automation and Mechatronics, 2006 IEEE Conference on*, pages 1–6, June 2006.

[8] Jung-Woo Park, Hyon-Dong Oh, and Min-Jea Tahk. UAV Collision Avoidance based on Geometric Approach. In *SICE Annual Conference, 2008*, pages 2122–2126, Aug. 2008.

[9] Andrew Proctor. Development of an Avoidance Algorithm for Multiple, Autonomous UAVs in a Finite Space, June 2010.

[10] D. Rathbun, S. Kragelund, A. Pongpunwattana, and B. Capozzi. An Evolution based Path Planning Algorithm for Autonomous Motion of a UAV through Uncertain Environments. In *Digital Avionics Systems Conference, 2002. Proceedings. The 21st*, volume 2, pages 8D2:1–12 vol.2, 2002.

[11] Arthur Richards and Jonathan P. How. Aircraft trajectory planning with collision avoidance using mixed integer linear programming. In *American Control Conference, 2002. Proceedings of the 2002*, volume 3, pages 1936 – 1941 vol.3, 2002.

[12] Varun Sampath and Chester Hamilton. Au-proteus project. http://www.seas.upenn.edu/~vsampath/research/au/, June 2011.

[13] Tom Schouwenaars, Bart DeMoor, Eric Feron, and Jonathan How. Mixed integer programming for multi-vehicle path planning. In *In European Control Conference 2001*, pages 2603–2608, 2001.

[14] Zhang Shengxiang and Pei Hailong. Real-time optimal trajectory planning with terrain avoidance using milp. In *Systems and Control in Aerospace and Astronautics, 2008. ISSCAA 2008. 2nd International Symposium on*, pages 1 –5, dec. 2008.

[15] Karin Sigurd and Jonathan How. UAV Trajectory Design using Total Field Collision Avoidance. *American Institute of Aeronautics and Astronautics*, 2003.

[16] R.J. Szczerba, P. Galkowski, I.S. Glicktein, and N. Ternullo. Robust algorithm for real-time route planning. *IEEE Transactions on Aerospace and Electronic Systems*, 36(3):869–878, July 2000.

[17] J. van Tooren, M. Heni, A. Knoll, and J. Beck. Development of an autonomous avoidance algorithm for UAVs in general airspace. Technical report, EADS Defence and Security, Military Air Systems, 2007.

Appendix A

Links to source code

Test bed repository: https://github.com/holtjma/AU-UAV-ROS

MILP Team Documentation: http://pfduav.com/

DSAS Team Documentation: https://sites.google.com/site/uavcaau/

APF Team Documentation: https://sites.google.com/site/auburn2011uav/