Collision Avoidance Techniques for Unmanned Aerial Vehicles
Technical Report #CSSE11 - 01

Ben Gardiner
Willamette University

Waseem Ahmad
Rice University

Travis Cooper
Mercer University

Matthew Haveard
Jones County
Junior College

James Holt
Auburn University

Saad Biaz
Auburn University

August 22, 2011

Auburn University
National Science Foundation
Research Experience for Undergraduates

Abstract

In order for Unmanned Aerial Vehicles (UAVs) to increase in capability and use, more programs that allow for the safe use of these vehicles in all situations are needed. These will allow multple advancements in market use and innovation in the field of UAVs, and situations like dangerous search and rescue or commercial flight will be radicially changed with the emergence of efficient and error-free UAV auto-pilot systems. Currently, multiple methods have been devised and used to control UAVs in simulated enviroments, but there are few existing cohesive methods for dynamic collision avoidance beyond pre-calculated path planning. In this paper, we examine benefits and costs of several path planning approaches including Mixed-Integer Linear Programming (MILP), geometric methods, and airspace discretization. Then we detail our implementation of MILP using the Gurobi solver in the ROS framework, and our conclusions regarding Receding Horizon Control (RHC) to bring MILP into a technique that is more feasible in real time.

1 Introduction

In the context of multiple unmanned aerial vehicles (UAVs) flying in the same airspace, a major problem is the avoidance of collisions between multiple friendly UAVs. In order for UAVs to progress beyond remote controlled craft into fully autonomous machines that run whole missions without human control, collision avoidance needs to be automated with dynamic processes that deal with problems as they appear. In order for the algorithms to be reactive, they must be very efficient to minimize calculation time because collisions are often detected with limited time to both calculate and maneuver to avoid them. In Section 2 of this paper, we summarize some of the existing path planning methods including Mixed-Integer Linear Programming (MILP), geometric and vector-based approaches, and airspace discretization. In Sections 3 and 4, we discuss our specific implementation of the MILP method for use on the EasyStar UAV through the ROS framework and some specific challenges that we encountered. Then in Section 5 we discuss results of our simulations.

2 Literature Review

2.1 Geometric Approach

One of the most straightforward approaches to solving the collision problem between UAVs is the consideration of them in a simple geometric space. This is usually done by considering each UAV individually as a point mass that has some rate of change vector allowing for calculation of its future positions. There are a few different ways to avoid collision using this simple geometric approach.

2.1.1 Point of Closest Approach

A simple method of first calculating whether or not a collision need be avoided in this geometric space used by [5] is considering the Point of Closest Approach (PCA). This algorithm simulates a UAV's future path based on its current speed and bearing and its future way-point and checks to see at what time it is closest to the other UAV(s) in the area. From this it creates a miss distance vector between them. It then compares the length of the miss distance vector to the minimum separation distance that is required between them. If the actual distance will be less than the desired distance, the collision avoidance algorithm begins.

The actual collision avoidance from [5] is based off of the miss distance vector. The miss distance vector $\vec{r}_m$ and the time of closest approach τ, are calculated from the relative distance vector $\vec{r}$ and the unit vector of relative velocity $\hat{c}$ between the two UAVs, with $\vec{r}_m = \hat{c} \times (\vec{r} \times \hat{c})$ and $\tau = -\dfrac{\vec{r} \cdot \vec{c}}{\vec{c} \cdot \vec{c}}$. When a conflict or collision is eminent, both craft share the task of avoiding collision by turning along $\vec{r}_m$ away from the conflict space

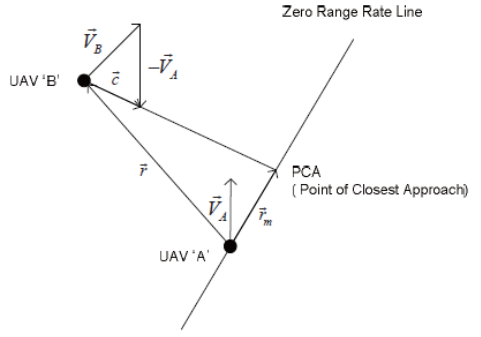


Figure 1: Relative motion of two UAVs

to extend the space between the planes without sending either one far out of the way. One UAV will be turned in the positive $\vec{r}_m$ direction and the other will be turned in the negative $\vec{r}_m$ direction, see Fig 1 from [5]. This is called a vector sharing resolution. Faster UAVs have a higher priority and take a smaller share of the avoidance as a slower craft can usually make a tighter turn and the faster vehicle will be out of the way sooner by taking the more direct route. A problem is encountered with a perfect head on collision as the miss distance vector is of zero magnitude [5]. A conditional statement is set to give a small magnitude to such a vector in order to allow for a positive and negative direction.

2.1.2 Collision Cone Approach

Another method of detecting a possible collision is the Collision Cone Approach [4]. The basic idea is to place a circle around the obstacle or UAV that must be avoided. The tangent lines from the circle to the UAV under consideration form the collision cone. If the path of the UAV that is currently under consideration is between the two tangent lines to the circle, then a collision check must be made. The check is based on the velocity and bearing of the UAV being considered. This is usually done to avoid static obstacles instead of dynamic, but can be modified to handle a dynamic environment.

The group that used this collision detection method [4] considered many different avoidance algorithms. The first method under consideration by [4] used differential geometric guidance. This creates an aiming line that will have a different x and y velocities than the original velocity. It will take the UAV away from the obstacle and then creates a way point on a spot on the aiming line to send the UAV to avoid the obstacle. This can be adapted to a dynamic problem by simply calculating the aiming line only if a collision is detected at a time step in the future.

2.1.3 Dubins Paths

One interesting approach to collision avoidance was the use of Dubins Paths that were slightly modified to make them more optimal with clothoid arcs [11]. A Dubins path is a series of either three tangential arcs, or two tangential arcs with a straight line in between, see Fig. 2. [11]

The arcs were modified by [11] to clothoid arcs which allow for a more continuous path than the tangential arcs provided by the Dubins paths. In order to determine the possibility of a collision, or even a conflict, a similar approach is taken to the point of closest approach. The paths are checked to see if they cross. If they do, the distance of each path up until they cross is measured. Assuming that the UAVs are flying at some constant speed, then it is easy to calculate the path length difference necessary to maintain a minimum separation distance. One must also account for instances when the paths do not cross, but come close to each other and create a possible minimum separation distance conflict without creating the possibility of a collision.

The creation of the paths is done in steps [11]. The first step is to find a flyable path. The only constraints are the physical limits on the UAV, for example, most aircraft besides

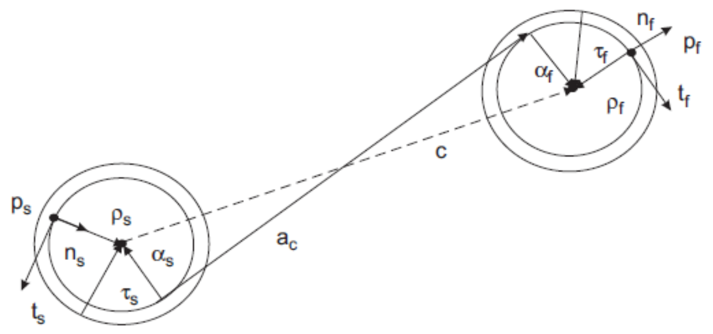


Figure 2: Dubins path with clothoid arcs

helicopters cannot stop and hover or make ninety degree turns. Next safety constraints are added. Each UAV has a minimum separation distance that must be kept between it and all other aircraft. [11] used two different ways mentioned earlier to handle checking for this. First, each UAV must maintain a minimum separation distance between all other UAVs. Second, no two UAVs paths may cross at an equal length, which would be the same as each UAV arriving at the same point at the same time. The paths are plotted to handle the constraints, but there is a safety net method to catch any unexpected obstacles or encounters. The method simply plots a single way point to change the route sufficiently to avoid the unexpected conflict. This backup algorithm is not designed to optimize as the main cooperative path plotting is, but is merely designed as an emergency avoidance maneuver for an unexpected obstacle.

All of these methods are based off of simple geometric sensing of where the other UAVs are and where there projected positions will be. It is simple and direct but can get computationally complex with large numbers of UAVs due to the number of comparisons that must be made. Also, as a path is changed to avoid a collision, the comparisons may need to be made again to check with possible conflicts with the other UAVs in the area. This can be shortened by looking only a fixed distance into the future every time, but it may reduce optimality.

2.1.4 Swarm Algorithms

A unique method of collision avoidance that has its origins in nature is a swarm or flock type algorithm. This collision avoidance is more often used for formations of UAVs that are flying together, but its principles are also applicable to collision avoidance in free flight. The core of the idea is based around the principles of attractive and repulsive forces. Typically the goal way points of the UAVs are attractive forces while the other UAVs and obstacles represent repulsive forces [10]. The forces keep the UAVs from hitting one another and get stronger the closer they get to each other causing more extreme avoidance maneuvers. These forces are defined in a set of simple rules that define the behavior of the UAV. [10] have a listed a set of basic rules for collision avoidance. There are two rules that would be applicable to free flight. The Homing rule allows UAVs to head toward a signal source, the way point. The Dispersion rule keeps the UAVs at a safe distance form each other, the minimum separation distance. The other basic rules that apply to swarm or formation flying are the Cohesion rule that allows UAVs to stay close to each other and the Following rule that makes one UAV follow others. The fifth rule could also be used in free flight to perform cooperative collision avoidance. It is known as the Alignment rule and allows the formation of UAVs to maneuver successfully as a group. This basic behavior could be modified to allow UAVs to avoid each other cooperatively. The key to a good algorithm in this instance is to find the right balance of force to keep an optimal or close to optimal path.

2.2 Airspace Discretization

Airspace discretization is the process of dividing the flying space into discrete square or cubic cells of uniform size. The collection of cells is represented by a weighted graph in which the vertices are the cells themselves. The edges of the graph connect orthogonally and diagonally adjacent cells. In a two dimensional representation, a center vertex has 8 edges connecting to it to cells north, northeast, east, southeast, south, southwest, west, and northwest of its position. This representation allows graph search algorithms such as A* or Dijkstra's Algorithm to plan the paths of the UAVs.

2.2.1 A* Algorithm

Ruz, Arvalo, Pajares, and Cruz utilize the A* algorithm to plan non-colliding paths for multiple UAVs, and detail a few complications of doing so. One of which is the balance between resolution, as in graph nodes per unit length, versus computational cost [9]. For example, in two dimensional space, there is an order of N^2 for searching for the optimum path, where N is length. As the possible airspace grows, it is clear the computational complexity increases greatly. On the other hand, if the airspace is broken into fewer, larger cells, there might not be enough detail to plan efficient, non-colliding paths.

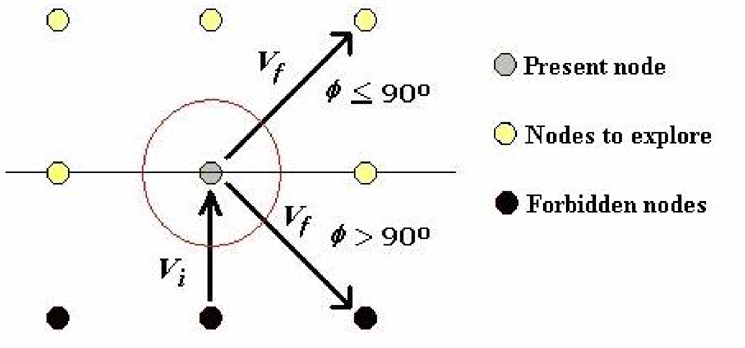


Figure 3: Grid Constraints

Although A* is easy to implement [9], it requires good heuristics to become efficient. The heuristics used in [9] are relatively intuitive. The first one, referred to as "Flying Direction Constraint" restricts possible nodes checked to ones that are less than $\pi/2$ from a given path, see Figure 3 from [9]. This simulates turning radii of fixed wing UAVs, and reduces computational complexity by one third. The other, "Inertia Constraint," simulates the fact that after reaching a way-point, the UAV will continue in the previous direction for a specified time. Both of these reduce computation in a minor way though, and [9] does not use the algorithms in real time.

2.2.2 Particle Filter Trajectory Planning

Rebollo, Ollero, and Maza associate each UAV with a trajectory, which they define as "a sequence of cells with an entrance time and departure time" [6]. Consequently, their communication between UAVs becomes much simpler because UAVs only have to transmit lists of cells. To detect collisions, each UAV only has to "find temporal overlapping between a cell of its trajectory and a cell that belongs to another UAV trajectory" [6]. The trajectories are known from the beginning based upon the next destination point and current heading. Therefore, based upon the initial paths, the problem that they address is how to minimally adjust velocities of all UAVs involved in a collision such that no new collision is created.

In [6], the authors use a combination of the Search Tree and Tabu algorithms to pursue a solution to multiple UAV collision avoidance. Each trajectory can be represented as a tree, with edge weights denoting how much time a UAV spends in an associated cell. Assuming that a conflict exists, the trees are first constructed in the order that the UAVs will cross the collision cell. The construction of the trees is based upon each UAV's predicted path until a potential conflict is detected, at which point, the current tree will backtrack to change the velocity of a UAV in previous cells so that it arrives at the conflict cell after the previous UAV has departed it. However, if the current tree must backtrack past the beginning of the tree, then no solution is found for that order. Consequently, the search tree algorithm changes the order in which UAVs encounter the conflict cell, by varying earlier velocities, and then builds the trees again, until a solution is found. Following this, [6] use a Tabu search to modify this proposed solution so that the overall velocity changes are minimal.

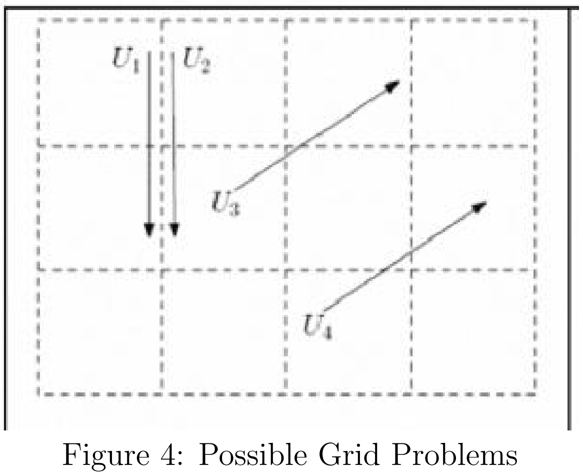


Figure 4: Possible Grid Problems

Alejo, Conde, Cobano, and Ollero build on the methods used in [6]. [1] defines the airspace and trajectories in a similar way as [6], but immediately present this statement regarding the methods used in [6]: "The grid model of [6] presents a disadvantage: two vehicles that are not in the same cell can be closer than other two that are actually in the same cell," see Figure 4 from [1]. Consequently, [1] defines a "minimum required separation" as a specified number of cells, as opposed to [6]'s approach of simply avoiding multiple UAV's simultaneously in the same cell. Also, [1] uses a particle filter to compute future trajectories and collision avoidance trajectories.

The predicted trajectory of the UAV and the minimum required separation will be determined from the particle filter [...] Thus, the proposed algorithm detects all the potential collisions whenever a prediction of the trajectory is received from the particle filter. [B2]

The goal of [1]'s method is to improve over the collision detection methods used in [6] by using a particle filter.

The particle filter uses a probability density function to estimate UAV trajectories and possible deviations from those trajectories based upon previous UAV states. The set of particles that it utilizes represent instances of previous state trajectories [1]. The goal of the particle filter is to estimate where a UAV will be at a given future time. In addition, the particle filter takes into account behavior and characteristics of each UAV, constraints such as minimum speed or turning radius, and atmospheric models. It also uses re-sampling to refine the estimated path and deviation, and relies on an accurate sensor system and UAV physical constraints such as turning radius for this process. Once the particle filter is used to predict a set of trajectories for the UAVs, [1] uses a search tree algorithm and optimization process, very similarly to [6].

Both [6] and [1] propose a solution that changes velocity profiles of each UAV and uses two heuristic-based search algorithms to do so. However, there are many possible scenarios in which speed cannot be changed, and the path must be modified, so the algorithms proposed in [6] are possibly problematic. In addition, the grid algorithms proposed are very suboptimal solutions; the advantage being that the possible solutions are found very quickly.

2.3 Mixed Integer Linear Programming

In the last decade, much research has been done on using MILP to plan trajectories for multiple UAVs taking both dynamic and static obstacles into account. Mixed integer linear programming is a powerful optimization technique used for modeling, complex planning, and control problems involving both continuous and discrete decisions. MILP has been used for optimizing various control problems such as crew scheduling, production planning, and vehicle routing. A comprehensive survey on integer programming with various algorithms and uses are described in [3].

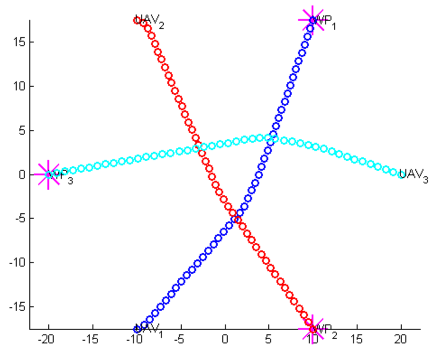
Using MILP, modeling can be done of logical constraints such as obstacle and collision avoidance rules or no-fly zones, as well as continuous constraints such as the maximum velocity and minimum turning radius of the vehicle. The system is optimized using advanced algorithmic techniques such as cutting plane methods, branch-and-bound algorithms, integral basis methods, and approximation algorithms; more information can be found on these in Bertsimas and Weismantel's book Optimization over Integers [2]. Free and commercial MILP optimizers are available for use such as LP_Solve and CPLEX respectively.

The major advantage of using MILP for collision avoidance is that it can handle hard constraints such as a minimum speed, maximum speed, minimum turning radius and maximum acceleration, in addition to soft constraints defined in cost functions. It globally plans a path for all the UAVs under control to avoid collision in the first place. Furthermore, it can account for multiple unordered waypoints and static obstacles such as no fly zones. And most importantly, it finds a feasible solution if it exists.

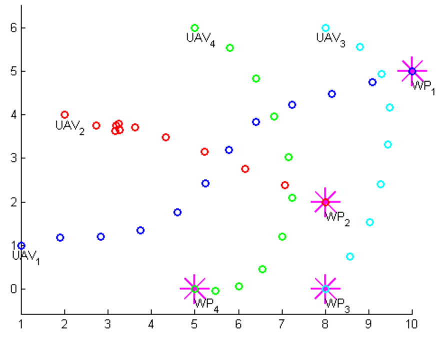
Richards and How formulate a 2-D model using MILP in their paper [7] which is briefly described here to explain how a trajectory is computed: In order to make the kinematics of the aircraft compatible to the MILP framework, the aircraft is approximated as a point mass moving with limited speed and acted on by a force of limited magnitude. The force magnitude limit acts as the constraint for the minimum turning radius for an aircraft that is in motion. Applying the maximum force perpendicular to the plane's moving direction would result in a maximum turn, along the curve with the minimum turn radius.

Using the current state vector of the plane and applying a force vector results in the future state vector of an aircraft, repeatedly doing this in T time steps for N planes results in a trajectory planned for all the planes T time steps into the future. The states of the plane are subject to more constraints such as a minimum distance of separation between all the planes at all times, and a separation between the planes and static obstacles. A destination waypoint is used for all the planes. The optimization of the trajectory using the minimization of a time function or a cost function results in the fastest or the most fuel-efficient path respectively which follows the constraints specified to the MILP model to avoid collision.

Although MILP provides a very elegant and feasible solution to the problem, it is very computationally intensive and cannot compute a path fast enough for the dynamic environments of UAVs. Using the path planning software provided by Richards and How [8], different scenarios were simulated involving 2-4 UAVs in a limited airspace that were initially bound to collide. With intermediate waypoints computed in 2 second intervals for all the UAV between their initial and final positions, a typical simulation took anywhere from half a second to a minute or more. Clearly, in the case of a plane which has a



(a) Number of UAVs: 3 Time to Compute: 1.06s



(b) Number of UAVs: 4, Time to Compute: 72.71s

Figure 5: Comparison of Route Planning

minimum speed required to maintain altitude, this method would not be practical as a collision could easily occur by the time a safe trajectory is computed. Furthermore, by the time the trajectories are calculated, the positions of all the planes might have changed so much that the trajectories calculated using the planner are no longer useful.

Although increasing the time steps reduces the computation time, it also reduces the optimality of the path as fewer intermediate waypoints would be specified between the UAVs initial position and destination waypoint. Moreover, an MILP based approach will be even slower for a 3-D environment where altitude is varied to account for terrain, buildings, or other UAVs for collision avoidance.

In order to work around the computation bottleneck, there is ongoing research on a receding horizon control (RHC) or model predictive control (MPC) to compute trajectories in real-time typically within a second. The idea behind a RHC based approach is to significantly reduce computation by only computing a trajectory within a limited, defined planning horizon that brings the UAV closer to its destination waypoint. A series of short trajectory segments can be computed in real-time and followed by the UAV until its goal is reached. Computing these short segments takes lesser time, however the optimization of these segments does not necessarily result in the optimization of the overall path to the destination waypoint. The flight time difference however can be trivial in most cases between a path computed with and without the use of a receding horizon.

Shengxiang and Hailong present an efficient algorithm in their paper [12] that computes collision free trajectories in real-time using a receding horizon framework with MILP in a 3-D terrain environment. The computation time of each intermediate waypoint is well within a second in a two-dimensional projection of a path computed. For a RHC based approach, the scope of the path planning is limited to the horizon specified in the algorithm; therefore the UAV is short-sighted. If the horizon is small enough, and given such a situation arises where multiple UAVs all converge towards the same point, in the next iteration when the UAVs become aware of each other, it might be too late to produce a collision free path. Similarly a UAV can try to approach its destination through a long U-shaped obstacle and run into a dead-end.

While MILP can find a optimal set of paths for multiple UAVs, its high computation time makes its pure form infeasible for real time problems such as UAV collision avoidance. Also, MILP is not dynamic and thus cannot react to unexpected problems such as non-cooperative aircraft or other obstacles. A solution to these difficulties is to use RHC with MILP to constantly solve partial solutions. RHC will allow the planes to react to situations, and is also fast enough to be updated in very quick intervals, but this comes at the cost of fully optimal paths.

2.4 Literature Review Conlusions

Dividing the airspace into discrete cells allows for easy implementation of graph algorithms such as A* or Search Trees [6, 1, 9]. However, it brings up the difficulty of creating suboptimal paths [1] or lengthy computational times [9]. Also, these algorithms often require very capable heuristics in order to be feasible, and even then, they are not necessarily as efficient if obstacles or collisions are detected mid-flight. However, they are very capable means of path planning, and should not be discounted.

Geometric approaches allow for simple solutions to collision avoidance that can adhere to many constraints. However, the calculations become quickly intensive as the program must recheck each plane that is averted. To counter this one sacrifices long range calculation and risks close range issues if rapid course changes occur. Swarm algorithms are a reliable method of collision avoidance but require more fine tuning to be very efficient. This efficiency drops in the case of multiple UAVs in a small area and becomes unusable to a degree at set numbers of planes.

A set of optimal trajectories for all the UAVs in the centralized system can be found through the use of mixed integer linear programming (MILP). The solutions are very elegant, but the amount of time it takes to compute them makes them infeasible for real-time path planning. A receding horizon is used to continuously compute the optimal path only a few time steps into the future. This reduces the computation time and makes it possible to use MILP for producing collision free paths for multiple UAVs.

After review of the previous methods of collision avoidance, the MILP method seems to have the most potential for route optimization while avoiding multiple, dynamically changing obstacles. MILP shows more promise in optimality compared to geometric approaches, but it requires more development to operate under the same computational time frame. In comparison to airspace discretization, MILP appears to surpass the potential of grid based systems with vehicles under real world constraints. In situations with vehicles with omnidirectional movement and speed control, grid based methods will out-perform the MILP method. Additional research in a combined strategy which implements multiple methods on a situation based case is also possible.

3 Implementation

In order to use MILP, we must formulate a system of linear constraints and an objective function for a solver software to find solutions for. In this section, we summarize the constraints that we use for UAV collision avoidance and path planning, then detail some specific challenges that we ran into, and finish with our addition of receding horizon control for making MILP into a more dynamic tool for real time problems.

3.1 Constraints and Objective Function for the Solver

Given that we have a set of UAVs each with an ordered list of destination waypoints, our objective function is to minimize the time it takes for each UAV to reach its appropriate destination, with a set of constraints that we define for the solver. In mixed integer linear programming, we define the objective function as J, so the function given to the solver is Equation 1

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

where N is the total number of UAVs and T_{F_v} is the time that UAV v reaches its final waypoint. Our first constraint is to model real world dynamics, so we define a UAV's current state and force vectors in Equation 2

$$state = \begin{bmatrix} x \\ y \\ v_x \\ v_y \end{bmatrix} \qquad force = \begin{bmatrix} f_x \\ f_y \end{bmatrix} \tag{2}$$

where x and y represent the UAV's location, v_x and v_y are components of the velocity vector, and f_x and f_y are components of the velocity vector. Using these definitions, we have to relate an individual UAV's next state ($state_{t+1}$) to its current state ($state_t$) with Equation 3.

$$state_{t+1} = A \times state_t + B \times force \tag{3}$$

where A and B are constant matrices defined in Equation 4 that are used with the state and force vectors to represent the displacement equation, Equation 5

$$A = \begin{bmatrix} 1 & 0 & dt & 0 \\ 0 & 1 & 0 & dt \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad B = \begin{bmatrix} \frac{1}{2}(dt)^2 & 0 \\ 0 & \frac{1}{2}(dt)^2 \\ (dt) & 0 \\ 0 & (dt) \end{bmatrix} \tag{4}$$

$$x_{t+1} = x_t + V_t dt + \frac{1}{2}a(dt)^2 \tag{5}$$

where dt is change in time.

Our second constraint is to model minimum and maximum velocities. Therefore, we

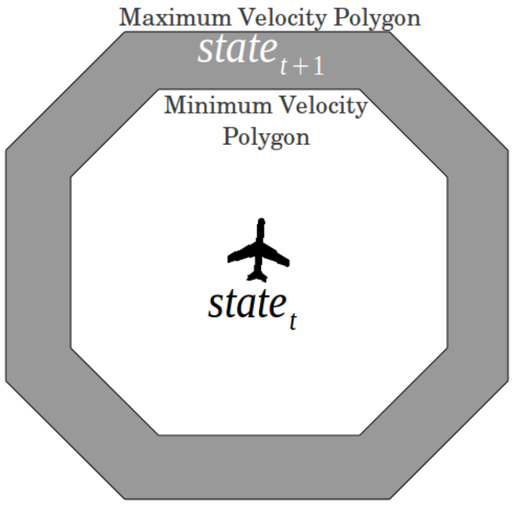
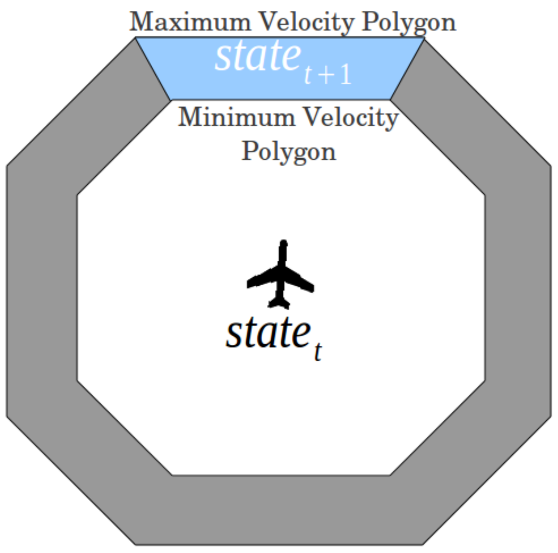
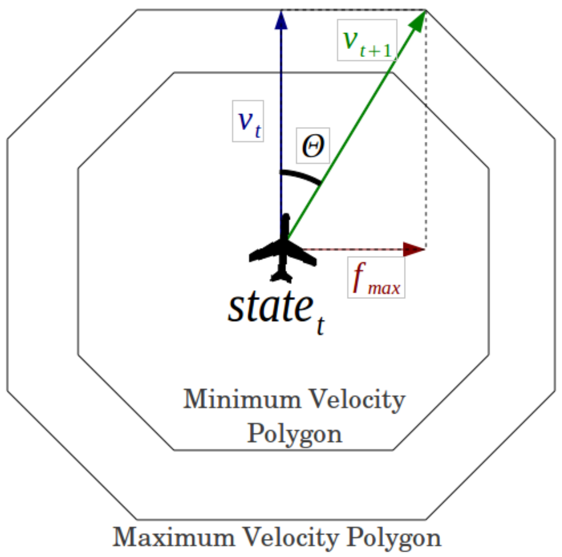


Figure 6: Minimum and Maximum Velocities

model minimum and maximum velocity as polygons around the UAV's location at time t, and its location at time t must be between the two polygons as demonstrated in Figure 6. We use polygons as opposed to actual circles so that they can be denoted as linear constraints for the solver, and the more sides the polygons have, the closer the approximation to a circle. However, we cannot simply represent the UAV's next location as between the two polygons, because we are simulating fixed wing UAVs and as such they can only move in a forward direction with limited turning constraints, so we must limit its next state with a turning constraint, as shown in Figure 7a. The way to implement this to create



(a) Added Turning Constraint



(b) Force

Figure 7: Turning Constraint

an additional constraint for maximum turning force, denoted f_{max} in Figure 7b, which is always perpendicular to the current velocity, to create a new velocity vector. The angle θ represents the difference between v_t and v_{t+1}, and it is this angle that must be kept under a specified value. We had some problems specifying this angle, detailed later in the section.

The final constraint is to maintain a minimum separation distance between all UAVs so that collisions are avoided. Thus we have created an objective function, to minimize time it takes for all UAVs to reach their waypoints, and set of constraints for an MILP solver. The constraints are dynamics and physics constraints that model real world motion and a minimum separation distance. These are static constraints in our framework for the solver, and then we input UAV locations and waypoints each time the algorithm is run.

3.2 Latitude and Longitude to Virtual Cartesian Coordinates

One of the challenges that we faced in our implementation is that MILP solvers use cartesian coordinates, which are not synonomous with latitude and longitude because the earth is curved. Therefore, we use the haversine formula [13] to translate all UAV's coordinates to a virtual coordinates, using a "virtual origin" which is a point on one edge of the field. First, we convert the geographic coordinates to polar coordinates by finding a bearing using Eq (6)

$$\theta_{UAV} = \tan^{-1}\left(\frac{\sin(\Delta_{long})\cos(lat_U)}{\cos(lat_O)\sin(lat_U) - \sin(lat_U)\cos(lat_U)\cos(\Delta_{long})}\right) \tag{6}$$

where Δ_{long} is the change in longitude from the origin point to the UAV's location, lat_O is the origin latitude, and lat_U is latitude of the UAV. Then, we find the distance in meters between the UAV and the "origin" using Eq (7)

$$d_{UAV} = 2R\sin^{-1}\left(\sqrt{\sin^2(lat_U - lat_O) + \cos(lat_U)\cos(lat_O)\sin^2(\Delta_{long})}\right) \tag{7}$$

where R is the radius of the earth, estimated at 6371009.0 km. However, θ is in relation to north, so when we do our translation from polar to cartesian coordinates, we must account for this as well so the positive X axis is east, and the positive Y is north, which is easier to visualize.

$$x_{UAV} = d_{UAV}\cos(90 - \theta_{UAV}) \tag{8}$$

$$y_{UAV} = d_{UAV}\sin(90 - \theta_{UAV}) \tag{9}$$

Then our solver returns coordinates in this virtual cartesian system (x_{dist}, y_{dist}), so we must translate back into latitude and longitude to pass to the autopilot, so first we have an easy translation to polar coordinates in equations (10) and (11), and we want our θ_{dest} to be in terms of north.

$$\theta_{dest} = 90 - \tan^{-1}\left(\frac{y_{dest}}{x_{dest}}\right) \tag{10}$$

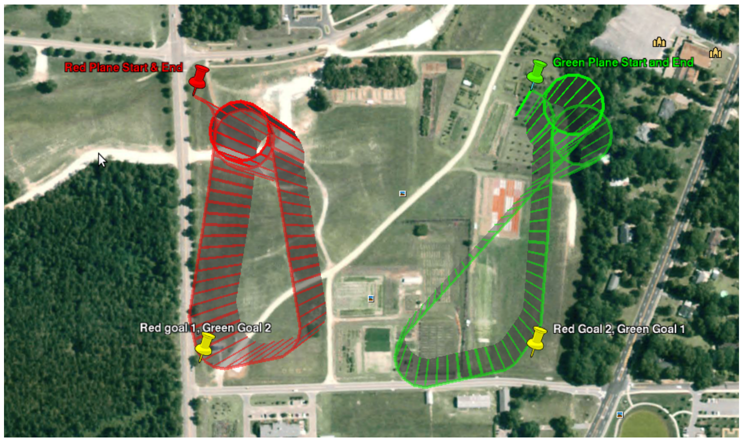
$$d_{dest} = \sqrt{x_{dest}^2 + y_{dest}^2} \tag{11}$$

Then, we reverse the haversine formula with equations (12) and (13)

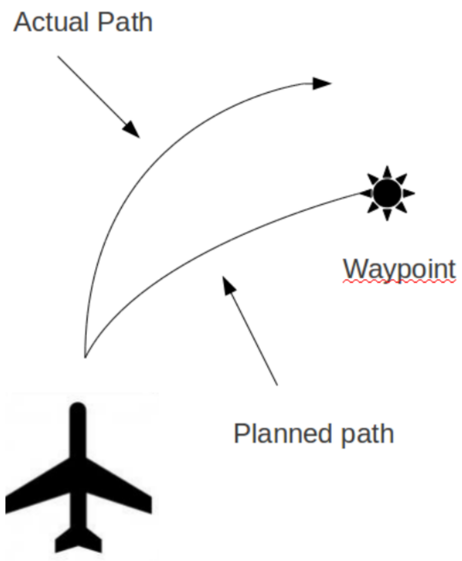
$$lat_{dest} = \sin^{-1}\left(\sin(lat_O)\cos\left(\frac{d_{dest}}{R}\right) + \cos(lat_O)\sin\left(\frac{d_{dest}}{R}\right)\cos(\theta_{dest})\right) \tag{12}$$

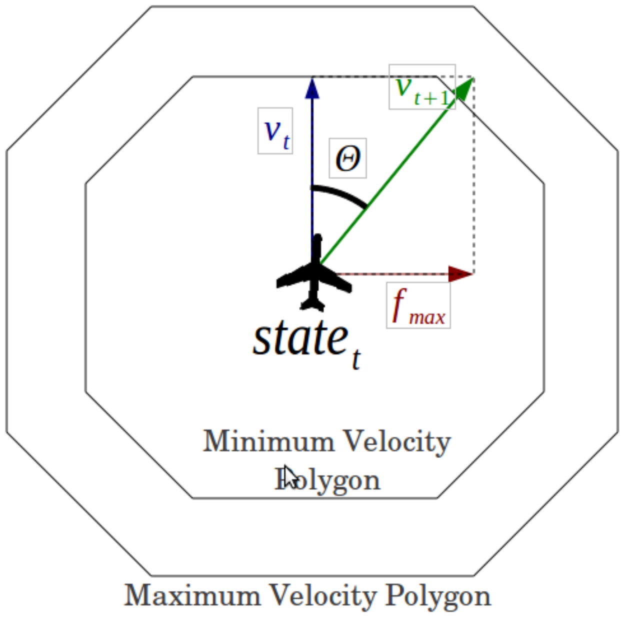
$$long_{dest} = long_O + \tan^{-1}\left(\frac{\sin(\theta_{dest})\sin\left(\frac{d_{dest}}{R}\right)\cos(lat_O)}{\cos\left(\frac{d_{dest}}{R}\right) - \sin(lat_O)\sin(lat_{dest})}\right) \tag{13}$$

The equations are used to take curvature of the earth into account and add negligible time to the algorithm analysis.



(a) Loop Problem in Implementation





(b) Planned and Actual Paths

(c) Minimized Velocity

Figure 8: Turning Problem

3.3 Turning Constraint

Another challenge is refining MILP to work with a turning constraint. If a plane has a waypoint that it would have to turn more than its maximum turning angle to reach, then it will simply loop around the waypoint, having no way to reach it see Figure 8a. The problem is created because the planned path was using a sharper turn than the simulated UAV was capable of making, resulting in a misconception of being able to turn sharply enough to achieve the waypoint that it was headed to. In effect, the solver was calculating a future state that was unachievable in actuality due to the real turning radius of the UAV, see Figure 8b. This error resulted in the UAV circling its waypoint multiple times as the solver tried over and over to send it directly to the waypoint when it should have headed off to make a larger turn and correct its approach.

The turning radius of the UAV in the algorithm is based on the maximum velocity and the maximum acceleration, see Figure 7b. However, in Figure 7b, v_t is maximized. If v_t were minimized like in 8c, then θ becomes increased. Thus, we have to adjust both the relationship between f_{max} and v_{max} and between f_{max} and v_{min}, using empirical tests to determine precise values. Then we adjust the force constraint so that the solver can plan more realistic paths and remove the looping issue.

3.4 Receding Horizon Control

In order to compute path trajectories for multiple UAVs in real-time so that our algorithm can become more robust and dynamic, we utilized a Receding Horizon Control (RHC) to generate trajectories that lead our UAVs to their destination way points in a collision-free manner. Although the MILP framework can solve the problem globally and compute optimal paths, it is not feasible as it might take several hours to produce a flight plan for a few UAVs with a few ordered destination waypoints. Furthermore, a UAV might need to deviate from it's specified path, waypoints might need to be changed mid-flight, or an unexpected obstacle might be introduced. The paths for all UAVs would have to be recomputed for any of these situations and that would further take up more time, making this approach infeasible.

By solving only up to a certain number of time steps into the future, locally optimized trajectories are generated that bring the UAVs closer to their destination waypoints. The MILP solver is repeatedly used mid-flight to produce a path for the UAVs based off of their current positions and velocities. This allows the UAVs to continually re-correct their paths in a real-life situation based on the new updates that they receive of their own position and the environment.

A set of interpolated coordinates are produced as a path for the UAV to follow. The coordinates are spaced dt seconds apart which specifies our time step size for the dynamics of the optimization framework. An Nt number of time steps are solved to produce a path. However, only a fraction of the path is followed upon which a new path is produced and the previous one is discarded. This is done to avoid a situation where the last points of the paths of two UAVs brings them dangerously close to each other. In the next iteration, as the UAVs approach their respective points, they might be too close to turn away from each other or the MILP solver might not be able to compute a feasible path. If only a fraction of the path is followed and then a new path is produced, we can avoid such a situation.

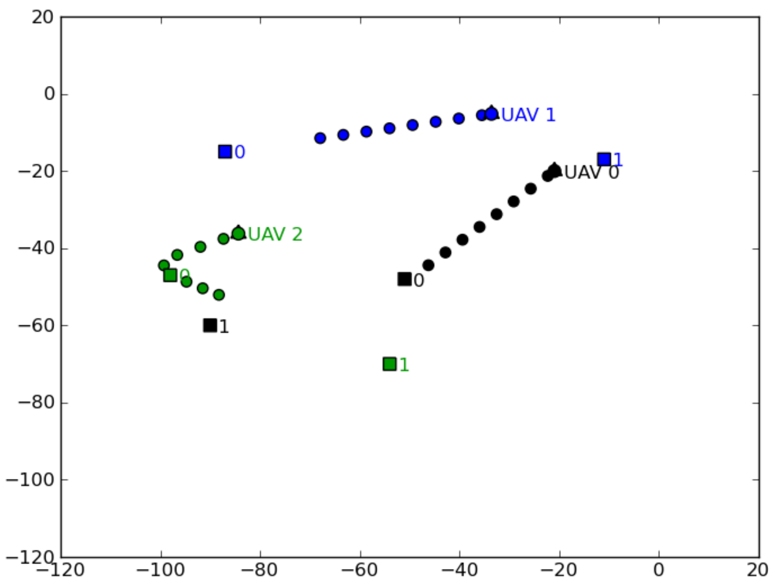
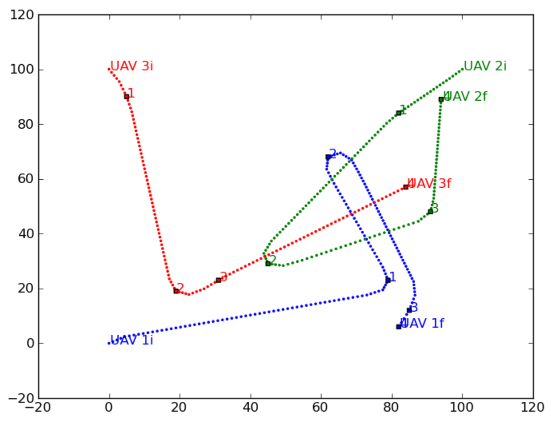


Figure 9: One iteration of 3 UAVs' computed path segments using RHC

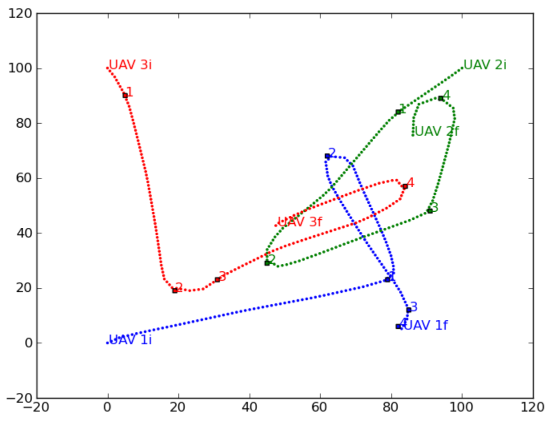
For a situation where UAVs are following a series of ordered waypoints. The MILP solver is invoked with up to two destination waypoints so that if one waypoint is within the reach of the UAV in the current iteration, it approaches it from an angle so that it is facing the next waypoints. This produces shorter paths and reduces the flight times of the UAVs.

One of the characteristics of an MILP solver is that often it will find a fairly optimal solution relatively fast, and then spend a lot of time searching the solution space for a more feasible solution. However, since we are generating paths online, a sub-optimal path computed within a second or two is preferred over a path that takes longer to produce. Gurobi, the solver that was used to produce paths allows you to specify a time limit. A reasonable time limit on the solver would be 2-4 seconds. Upon calling the solver, it will either return an optimal solution within the time-limit, a sub-optimal solution once it runs out of time, or no solution at all in the case that the input is too constrained or there was not enough time to produce a feasible solution. In the case where the solver returns an infeasible solution, the UAVs can continue to follow their current path segments in the hope of getting a new path segment in the next iteration, or they can enter a loiter circle while a new path is being generated with a higher time-limit.

In Figure 10, there are 3 UAVs each with a Minimum Speed of 1.2 units/s. Maximum Speed of 1.5 units/s. Maximum force of 0.20 units/s. These settings effectively enforce a



(a) MILP Flight Map



(b) RHC 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 10: RHC versus Full Path Planning

turning constraint of 22.5 degrees per second. All UAVs start out from the corners of a 100x100 units field and have random waypoints within the field. The last UAV to reach it's 4th destination waypoint terminates the simulation. The path trajectories produced were specified by interpolated coordinates to be followed spaced 4 seconds apart each.

As shown by Figure 10, it can be seen that globally computing paths results in more optimal trajectories and a shorter flight. However, the trade-off between optimality and feasibility due to computation time is immense with a RHC. Each path segment produced was 32 seconds long. However, only 12 seconds of the path were followed before a new path was computed and the older one was discarded. At times, the solver did not return a solution, so the UAVs continued to follow their paths beyond 12 seconds and soon a new path was returned.

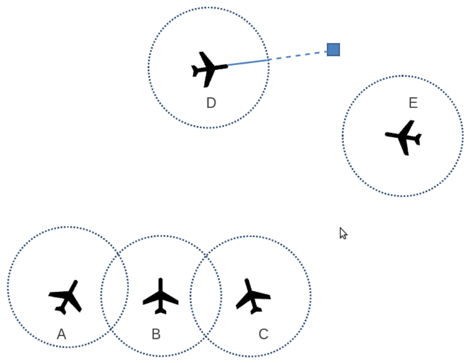


Figure 11: Connected Components

As we increase the number of UAVs to 8, 16, 32 or more, it was noted that even a receding horizon was too slow to compute collision-free trajectories online. A more robust algorithm was needed to handle a larger number of UAVs. In a situation with two UAVs that are very far away from each other and could not possibly collide in the next iteration, it was noted in our research that using the solver once to produce a path for both UAVs took longer than to use the solver twice, once for each UAV. This led to the idea of connected components of UAVs. If two UAVs are a safe distance away from the span of their horizons, the path produced by the solver could not lead them to a collision in the current situation so they should be solved separately to reduce computation time. Thus, the problem can be broken down into a set of disjoint components, where all the UAVs in each component have an overlapping horizon with at least one other UAV.

For example in Figure 11, UAVs A, B, and C are all in one connected component, because their horizons lead to a possibility of a collision. Whereas UAVs D and E are not close enough to the other set to need to solve together. In another iteration, there might be a different set of connected components. Therefore, the connected components have to be computed during each iteration, however that is trivial in terms of computational time.

Not only does the use of connected components inherently speed up computation, it also allows for parallel computation by utilizing multiple processing cores for producing paths. This allows a single coordinator to command a very large number of UAVs given enough processing power and cores. With the use of connected components, there is room for further research on coordinating a massive number of UAVs. Using a larger Nt value produces longer paths, and using a smaller dt value for the solver produces nicer curves but both take longer to compute. Therefore, a variable dt, Nt, and horizon length can be used based on the number of UAVs in the given component and their characteristics. A dynamic receding horizon can be implemented in this manner, one that finds a balance between optimality and feasibility of solutions returned by the MILP solver based on the circumstances. The implementation and use of a dynamic receding horizon has been left for future work.

4 Results

We utilized the Robot Operating System (ROS) to create a framework for directing real and simulated UAVs in our implementation of MILP. ROS allows for management of multiple parts that it calls nodes and communication between them using messages and services. The version of ROS used was Diamondback 1.4.8 on a Ubuntu 10.10 Linux operating system as it was fully supported by ROS. The UAVs are simulations of the EasyStar UAV (EasyStar UAV) fixed-wing aircraft that have a constant speed of 11.60 m/s and a maximum turning angle of 22.5°. The autopilot and simulated craft operate by navigating directly to specified destination waypoints given in terms of latitude, longitude and altitude. Inside the ROS framework, the simulator node posts a message per second detailing its current location and destination. Given that we are limited to one update per second per UAV, we define a collision as two UAVs being within 12 meters of each other, because that is the farthest away from each other that they could be with an actual collision occurring between updates. We then define a conflict as multiple UAVs being within 24 meters of each other, indicating an impending collision. To evaluate the tests, we define a scoring system where each destination waypoint reached is worth 5 points, each conflict worth -1 points, and when two planes collide, they are simply removed from the score at that time, but their previous scores are recorded, Figure 12.

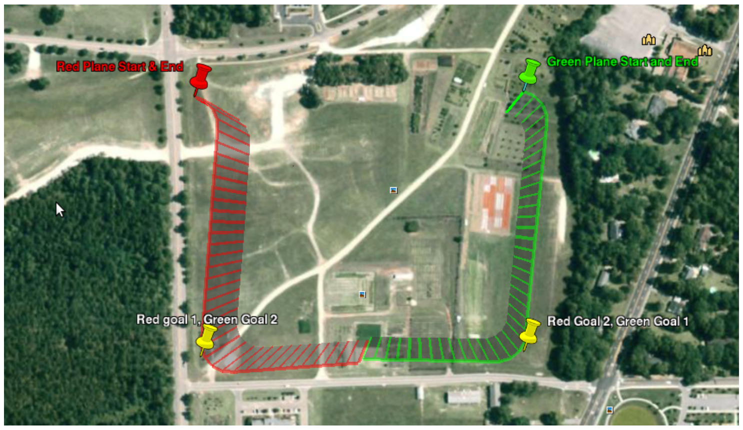
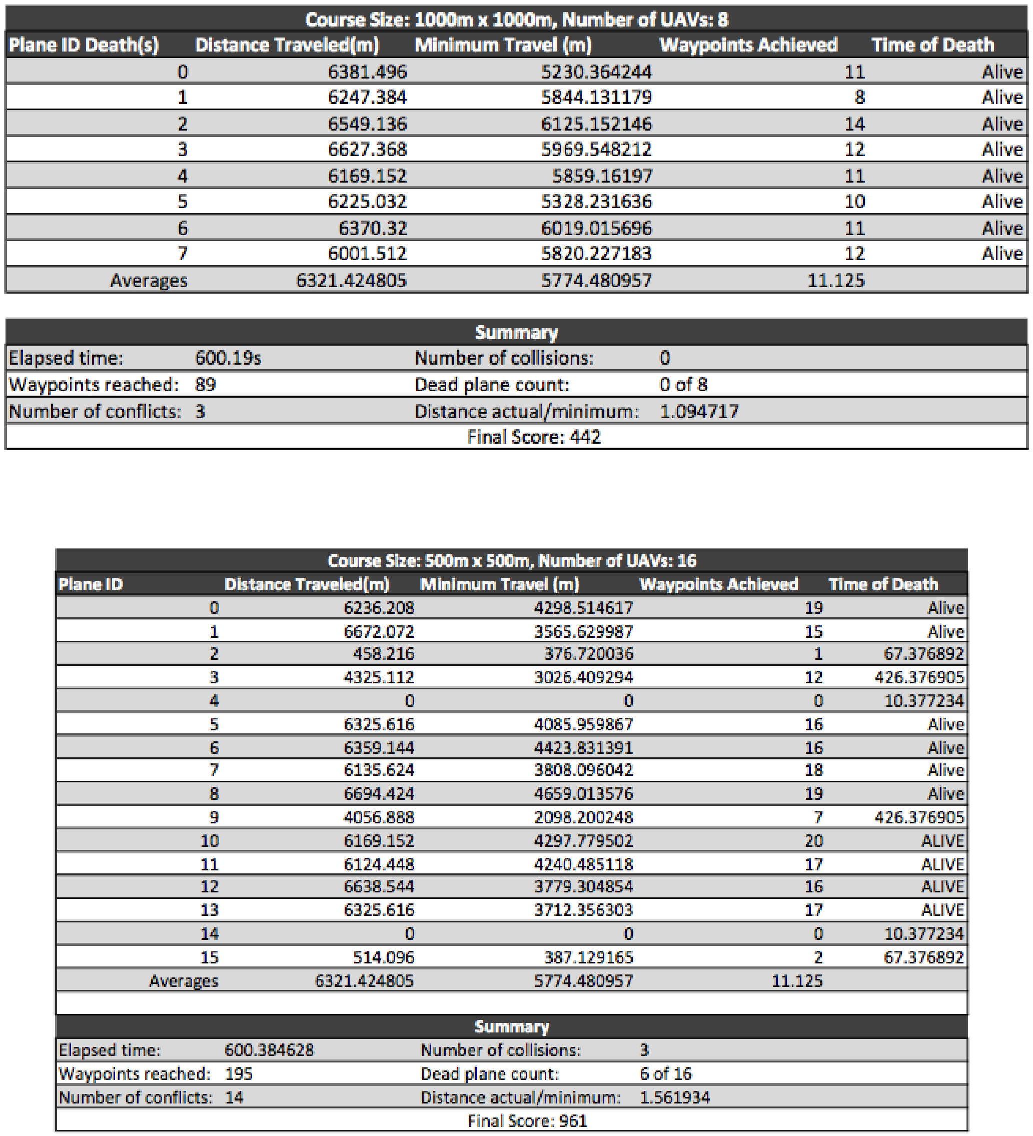


Figure 12: Collision between UAVs

Our algorithms were implemented using Python due to it's easy syntax rules and elegant design. Readily available mathematical, arrays, and plotting packages were used to make programming simpler, and the code easier to read. Pylab, a python plotting package was used for plotting the positions of the UAVs and updating them live. This allowed for easier observations of results. In order to parallelize the computation for connected components, Parallel Python a package that allows the sending of functions as jobs to servers was used. The MILP solver used was Gurobi which natively supports python, and utilizes multiple cores for optimization which has drastic results on a multi-core computer in comparison to a solve which does not support multiple cores. An academic license of Gurobi 4.5.1 was obtained and used on an Intel Core 2 Quad Q6600 machine with 4 GB Ram. The clock speed of each core is 2.4 GHz.



Course Size: 1000m x 1000m, Number of UAVs: 8				
Plane ID Death(s)	Distance Traveled(m)	Minimum Travel (m)	Waypoints Achieved	Time of Death
0	6381.496	5230.364244	11	Alive
1	6247.384	5844.131179	8	Alive
2	6549.136	6125.152146	14	Alive
3	6627.368	5969.548212	12	Alive
4	6169.152	5859.16197	11	Alive
5	6225.032	5328.231636	10	Alive
6	6370.32	6019.015696	11	Alive
7	6001.512	5820.227183	12	Alive
Averages	6321.424805	5774.480957	11.125	

Summary			
Elapsed time:	600.19s	Number of collisions:	0
Waypoints reached:	89	Dead plane count:	0 of 8
Number of conflicts:	3	Distance actual/minimum:	1.094717
Final Score: 442			

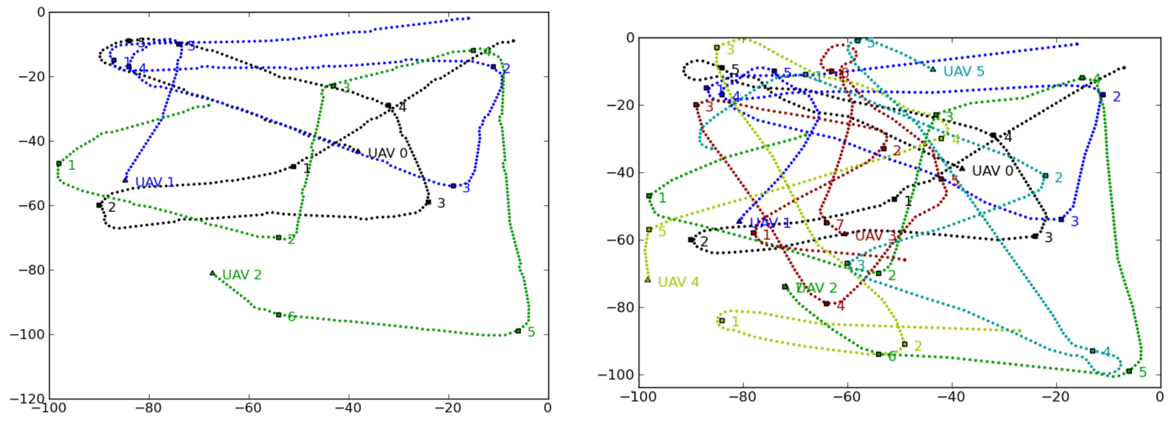
Course Size: 500m x 500m, Number of UAVs: 16				
Plane ID	Distance Traveled(m)	Minimum Travel (m)	Waypoints Achieved	Time of Death
0	6236.208	4298.514617	19	Alive
1	6672.072	3565.629987	15	Alive
2	458.216	376.720036	1	67.376892
3	4325.112	3026.409294	12	426.376905
4	0	0	0	10.377234
5	6325.616	4085.959867	16	Alive
6	6359.144	4423.831391	16	Alive
7	6135.624	3808.096042	18	Alive
8	6694.424	4659.013576	19	Alive
9	4056.888	2098.200248	7	426.376905
10	6169.152	4297.779502	20	ALIVE
11	6124.448	4240.485118	17	ALIVE
12	6638.544	3779.304854	16	ALIVE
13	6325.616	3712.356303	17	ALIVE
14	0	0	0	10.377234
15	514.096	387.129165	2	67.376892
Averages	6321.424805	5774.480957	11.125	

Summary			
Elapsed time:	600.384628	Number of collisions:	3
Waypoints reached:	195	Dead plane count:	6 of 16
Number of conflicts:	14	Distance actual/minimum:	1.561934
Final Score: 961			

Figure 13: Results from ROS

Figure 13 shows the results of two 10 minute simulated flights in ROS. In a field of size of 1000m x 1000m with 8 UAVs. The performance of the algorithm was excellent with a distance actual to distance minimum ratio below 1.10 and very few conflicts within the limited airspace. However, upon reducing the size of the field to 500m x 500m and increasing the number of planes to 16. There were 3 collisions and several conflicts, furthermore, the paths produced were relatively inefficient. This is due to the excessive maneuvering of the planes to avoid collisions with one another.

For fewer UAVs, our algorithm worked effortlessly and the CPU usage remained below 20% for the majority of the flights, see Figure 14.



Real-time Flight Simulated using ROS
Number of UAVs: 3
Number of Waypoints per UAV:: Unlimited
Random (only achieved shown)
Field Size: 1000m x 1000m (Flight Map scaled to 10m/unit)
Flight Time: 300s
UAV Speed: 11.76 m/s
Mean Iteration Computation Time: 0.0573s
Distance actual/distance minimum: 1.046

Real-time Flight Simulated using ROS
Number of UAVs: 6
Number of Waypoints per UAV:: Unlimited
Random (only achieved shown)
Field Size: 1000m x 1000m (Flight Map scaled to 10m/unit)
Flight Time: 300s
UAV Speed: 11.76 m/s
Mean Iteration Computation Time: 0.107s
Distance actual/distance minimum: 1.063

Figure 14: Map of Results

In order to compare the computation times for a globally solved path using a fixed horizon, a receding horizon implementation, and connected components for parallel solving, a series of identical simulations were run with an increasing number of UAVs in the same field of 1000m x 1000m to observe the relation of computation time for the three implementations. The simulation is terminated once all UAVs have reached at least 2 waypoints. Figure 15 show the expected feasibility of using a receding horizon for computing paths and shows the significant improvement in computation with the use of connected components.

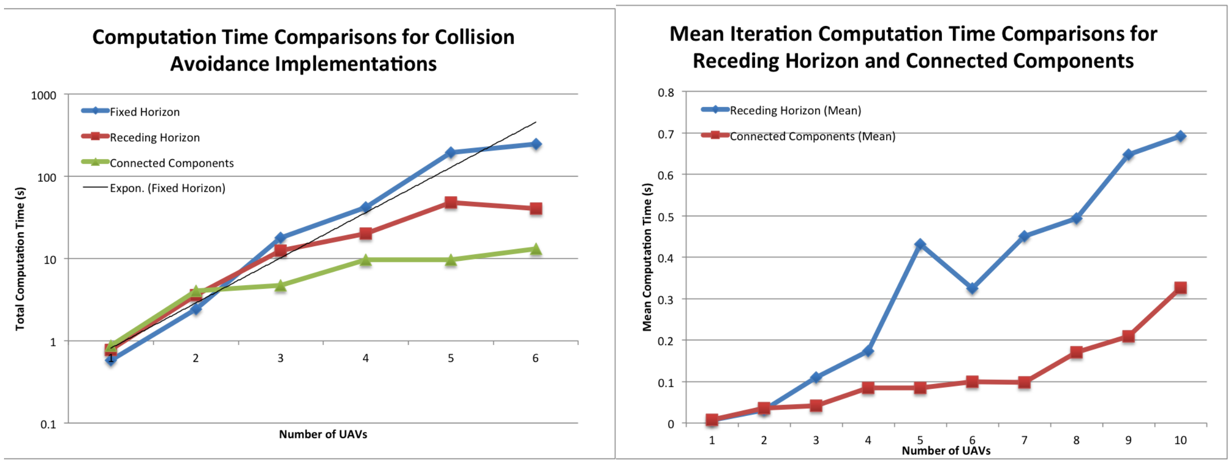


Figure 15: Map of Results

5 Conclusions

Mixed Integer Linear Programming (MILP) implemented using a Receding Horizon Control (RHC) system with Connected Components works very well for directing 8 or fewer Unmanned Aerial Vehicles (UAVs) but does not handle larger numbers well. Because of the greater amounts of UAVs, a lack of concurrency between the ROS framework and Gurobi solver began to appear such that the solver was often solving for locations of UAVs several seconds behind where it should have been, leading to collisions that should not have occurred. However, RHC and Connected Components show great promise in making MILP into a dynamic and reactive algorithm and there is excellent opportunity in future research to fine tune these two optimizations. In the future, we would first focus on improving the relations between the framework for the UAVs and the solver. In addition, we would develop a dynamic RHC that has a flexible horizon for different situations. Finally, we would implement a smart version of Connected Components to further reduce the size of the problems for the solver.

6 Notes

Framework used for UAV management: Robot Operating System
    Version: Diamondback 1.4.8 (ROS)
Simulated UAVs: EasyStar UAV (EasyStar UAV)
MILP Solver: Gurobi Solver version 4.5.1 (Gurobi)
Map Images generated with: Google Earth (Google Earth)
Graphics generated with: PyLab (PyLab)
Parallelized code using: Parallel Python (Parallel Python)
Testing Platform: Intel Core 2 Quad Q6600, 4gb RAM, 2.4 GHz Clock Speed
Operating System: Ubuntu 10.10

References

[1] D. Alejo, R. Conde, J.A. Cobano, and A. Ollero. Multi-uav collision avoidance with separation assurance under uncertainties. In *Mechatronics, 2009. ICM 2009. IEEE International Conference on*, pages 1 –6, april 2009.

[2] Dimitris Bertsimas. *Optimization over integers*. Dynamic Ideas, Belmont, Mass, 2005.

[3] M. Junger, M. Junger, T.M. Liebling, D. Naddef, G. Nemhauser, and W.R. Pulleyblank. *50 Years of Integer Programming 1958-2008: From the Early Years to the State-of-the-Art*. Springer Verlag, 2009.

[4] A. Mujumdar. Nonlinear geometric and differential geometric guidance of uavs for reactive collision avoidance. Technical report, DTIC Document, 2009.

[5] 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.

[6] J.J. Rebollo, A. Ollero, and I. Maza. Collision avoidance among multiple aerial robots and other non-cooperative aircraft based on velocity planning. In *In Proceedings of ROBOTICA 2007 Conference, Paderne, Portugal*, 2007.

[7] A. Richards and J.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.

[8] Arthur Richards and Jonathan P. How. Mixed-integer linear programming for control. `http://acl.mit.edu/milp/`, 2006. This is an electronic document. Date of publication: [Date unavailable]. Date retrieved: June 2, 2011. Date last modified: June 14, 2006.

[9] J.J. Ruz, O. Arevalo, G. Pajares, and J.M. de la Cruz. Decision making among alternative routes for uavs in dynamic environments. In *Emerging Technologies and Factory Automation, 2007. ETFA. IEEE Conference on*, pages 997 –1004, sept. 2007.

[10] D. Scheidt and J. Stipes. Cooperating unmanned vehicles. In *Networking, Sensing and Control, 2005. Proceedings. 2005 IEEE*, pages 326 – 331, march 2005.

[11] Madhavan Shanmugavel, Antonios Tsourdos, Brian White, and Rafal Zbikowski. Cooperative path planning of multiple uavs using dubins paths with clothoid arcs. *Control Engineering Practice*, 18(9):1084 – 1092, 2010.

[12] 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.

[13] M.J.D. Smith, M.F. Goodchild, and P. Longley. *Geospatial analysis: a comprehensive guide to principles, techniques and software tools*. Matador, 2007.