Evaluation of Reactive Collision Avoidance Algorithms for Unmanned Aerial Vehicles

by

David H. Jones

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 10, 2015

Keywords: Unmanned Aerial Vehicle, Collision Avoidance, Artificial Potential Fields, Inverse Proportional Navigation

Approved by

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) platform 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 two 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 two algorithms tested in this paper are based on artificial potential fields and inverse proportional navigation. Artificial potential fields maintain strong performance across all categories because of the algorithms handling of many special cases. Inverse proportional navigation outperformed artificial potential fields by handling all scenarios with perfection. Furthermore, artificial potential fields failed to handle all of the stressful simulations, especially when the airspace became congested, while inverse proportional navigation handled all scenarios. While artificial potential fields is able to handle up to sixteen aircraft on a 500 meter square field and thirty-two aircraft safely on a 1000 meter square field, inverse proportional navigation is able to handle up to thirty-two aircraft on both a 500 meter square field and a 1000 meter square field.

Acknowledgments

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. Dr. Biaz has not only been my graduate school professor for the past two years, he has been a mentor that has molded my actions and thought process in a way I wasn't capable of doing myself. I would also like to thank all the members of the Auburn 2012, and 2013 REU site for their algorithms development that made this study possible: David Fish, Eric Westman, Hosea Siu, and Miriam Figueroa. Id 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 their grants. Id also like to thank everyone whos worked on the ATTRACT project to get it to the point where it is today.

Id like to thank my family who has supported me every step of the way and encouraged me to succeed in computer science. Finally, Id like to show my utmost gratitude to my fiance', Claire, who has provided unwavering support and love throughout this whole process.

Table of Contents

List of Figures

Chapter 1

Introduction

Remotely piloted aircrafts have been evolving since the early developments of flight. In the later half of the American Civil War, the earliest documented forms of Unmanned Aerial Vehicles (UAVs) were used. They were very rudimentary in design; the first UAVs were balloons filled with explosives and designed to land inside enemy ammunition or a supply depot and explode[1]. Prior to the Vietnam War, the United States had developed the technology to make UAVs more efficient. Therefore, UAVs were used extensively in the Vietnam War, which is when the term Drone was coined. Drones were later referred to as UAVs to indicate a higher level of autonomy. The Depart Of Defense defines UAVs as powered, aerial vehicles that do not carry a human operator, use aerodynamic forces to provide vehicle lift, can fly autonomously or be piloted remotely, can be expendable or recoverable, and can carry a lethal or nonlethal payload[2]. Within the past 5 years, the term UAV has migrated to UAS to represent the entire system associated with operating the aircraft (every component on the aircraft, the ground-control station, and the wireless communication link)[3].

Unmanned aerial vehicles are a growing field of research that supports a wide range of applications from civil land surveys to military surveillance. Usually, each UAV requires a human pilot to manually control it for a mission or an autopilot with a preloaded set of waypoints to visit. Many applications require a fleet of UAVs with different missions over a shared airspace. For a fleet of UAVs on independent path missions, collisions will be likely for human as well as for autopilot-controlled aircraft. UAVs will be more convenient and powerful when a fleet of UAVs can autonomously and safely fly to fulfill independent missions without human operator(s). To reach this autonomy, UAVs need collision avoidance algorithms to prevent collisions.

Three years ago, a program started at Auburn University to build an Aerial Terrestrial Testbed for Research in Aerospace, Computing, and maThematics (ATTRACT). The ultimate goal of the ATTRACT project is to autonomously, safely, and efficiently fly up to 12 UAVs within a limited airspace. So far, over fifty students worked and contributed to design and implement many parts of this project. James Holt, a graduate student advised by Dr. Biaz, designed and implemented a general software architecture for the ground station[11]. Throughout the ATTRACT project, many advancements have been made to evaluate and implement collision avoidance algorithms. The focus of the ATTRACT project lies with the evaluation and determination of the best collision avoidance algorithm. This paper evaluates four different algorithms and recommends the best one to be used. These four algorithms are split into two groups: proactive and reactive. Proactive algorithms are designed to make decisions ahead of time, such that they can plan out entire missions before they begin. Reactive algorithms are in real-time; reactive algorithms take in all information at the current time and plan a path for the current time step based upon it. This research is focused more on reactive, as these algorithms are more fitting for an environment where multiple UAVs are flying missions in a confined airspace, although two proactive algorithms are covered throughout the Literature Review.

As mentioned, James Holt spent multiple years involved in the research efforts of the ATTRACT project. For Mr. Holt's Master's thesis he proved that the Artificial Potential Fields algorithm was the best algorithm that was evaluated in his thesis[11]. He also proved that APF outperformed many other algorithms, including MILP, Sparse A*, and Dynamic Sparse A*. Notice that APF is the only reactive algorithm among the bunch. Therefore, the research presented in this paper is aimed at improving upon APF, by evaluation another reactive collision avoidance algorithm, Reactive Inverse Proportional Navigation Algorithm, a.k.a RIPNA. The outcome of this research will prove that RIPNA is a much more efficient and effective algorithm for maintain the safety of aircraft, while still providing adequate flight efficiency. Furthermore, this research will prove that RIPNA outperforms, APF, as well as MILP, Sparse A*, and Dynamic Sparse A*, by using five metrics of evaluation: the number of collision, number of conflicts, deviation from the straight-line route, number of waypoints achieved, and average life span per UAV. These five metrics are sufficient at providing a thorough analysis of collision avoidance algorithms and will prove that RIPNA is the best collision avoidance algorithm.

Chapter 2
Literature Review

2.1 History

In the mid-1950s, a mid-air collision occurred between two US aircraft carriers over the Grand Canyon[4]. This sparked interest in the development of collision avoidance systems. For decades after this collision, a variety of collision avoidance techniques were explored, but in 1974 the FAA narrowed its focus to the Beacon Collision Avoidance System. BCAS is a transponder-based airborne system, meaning it uses radio frequencies to detect nearby threats. Four years later, another mid-air collision occurred between an air carrier and a general-aviation aircraft, which lead to the advancement of the BCAS effort. In 1981, the name was changed to the Traffic Alert and Collision Avoidance System. As the TCAS was refined, the first international version (version 7), was named the Aircraft Collision Avoidance System (ACAS). Although the FAA has been using collision avoidance systems for decades, there are a lot of areas for improvement.

2.2 Collision Avoidance Algorithm Categories

Collision avoidance algorithms are usually broken down into two categories: proactive and reactive. Proactive algorithms plan the entire mission out beforehand, in effort to avoid collisions by planning smart, cooperative paths. Proactive algorithms usually take a lot of computation time, which is a significant limitation among UAVs. Reactive algorithms allow the UAV to fly their mission without a pre-planned path. Then, once a conflict arises, the reactive algorithms take in all the available information and execute a quick collision avoidance maneuver. With the inherent low computation speed on embedded systems, UAVs can better utilize reactive algorithms for collision avoidance scenarios. Below, two proactive and two reactive algorithms will be covered in more detail.

2.3 Mixed-Integer Linear Programming

Linear Programming is the process of disassembling a problem into a mathematical model. That model represents decision variables and constraints that adhere to the problem. MILP is a specific subset of linear programming problems in which the decision variables are integers.

MILP is used to solve a variety of different situations. In the past, MILP has been commonly used for transportation scheduling and cellular network routing. More recently, MILP has been evolved to handle UAV collision avoidance.

MILP is an adaptation of linear programming, which allows for a scenario to be broken down into mathematical constraints[5]. Once the constraints are in place, a linear programming solver can be used to find an optimal solution.

In order to migrate MILP to solve a scenario with UAVs, a few constraints need to be considered. Most importantly, each UAV is mapped by its position, velocity, and acceleration. It is necessary for the solver to know the current location of each UAV in order to maintain a safe distance between other UAVs. Also, constraints are needed to force the solver to maintain a safe distance between each UAV. Likewise, in order to find an optimized solution, a goal must be set to minimize the travel distance of each UAV. This gives the solver a goal and a method of evaluating success of the solution. An example of a MILP solution is shown in 2.1. The biggest limitation with MILP is the computation time. To solve a linear program, a non-optimal solution is found, and then, iteratively, better solutions are found. This takes a lot of time, and can even take up to 200 iterations (sometimes 10+ seconds) to find the optimal solution. This is unacceptable for a UAV; UAVs need to quickly avoid high-threat situations by making a quick avoidance maneuver. Therefore, for MILP

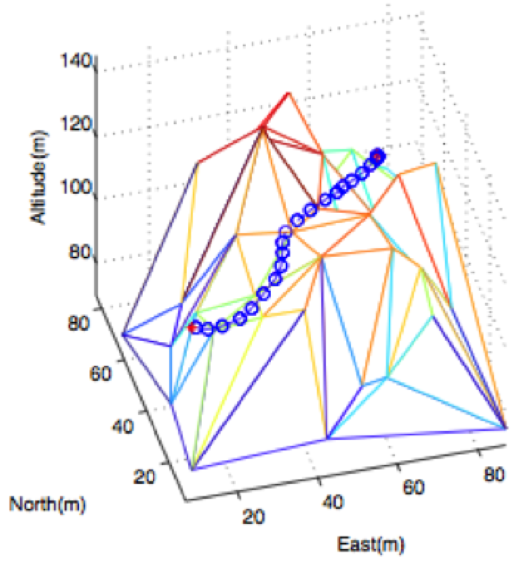


Figure 2.1: Path for a helicopter flying from initial position (10,10,120) to final position (70,60,120)[6]

to be a valid solution to collision avoidance, the problem needs to be heavily constrained to decrease the time complexity.

2.4 Sparse A*

Most algorithms use the aircraft as the focal point for collision avoidance. However, some algorithms represent the airspace as a grid and focus on the grid to maintain collision avoidance. These algorithms work by discretizing the airspace into a grid and assigning each grid cell cost values. Typically, these values are used to represent the threat of traveling to that cell, meaning a high threat would represent an obstacle and a low threat would represent a goal. These values are used to determine the best solution from point A to point B.

One of these algorithms is A* (pronounced A-Star). A* is a tree search algorithm, that once given a starting and finishing location a solution is produced. The algorithm guarantees to always find a solution if one exists. A* is a best-first search algorithm, meaning it expands to more promising nodes first. Which nodes are promising is determined by a heuristic function that estimates the cost of moving from the current node to the goal[7]. This

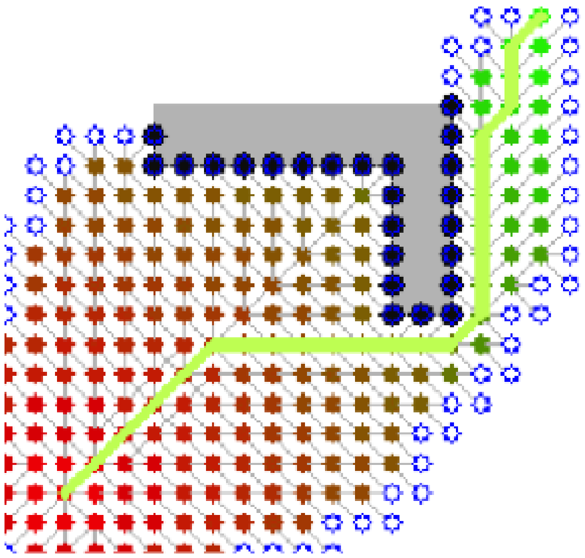


Figure 2.2: A* path planning example[8].

heuristic function can dramatically improve the overall performance of A*, when compared to breadth-first search or depth-first search. An example of an A* search is provided in 2.2. When migrating this algorithm to UAVs the airspace must be discretized into areas, such that A* can compute costs for moving from area to area. Also, heuristic functions have to be determined such that there is an advantage when moving toward waypoints, and a disadvantage when moving toward obstacles. Tweaking these heuristic functions can alter the output of the algorithm; therefore they are vital to the performance of the algorithm.

Similar to MILP, the limitation with A* is computation time. While the optimal UAV path is computed, every feasible path will be considered, making the program very computation heavy. In-order to minimize on the number of paths considered a receding horizon could be implemented. This can effectively prune the search space to allow the generation of an acceptable solution that converges in real-time. There has been some success in this approach, therefore the algorithm adopted the name Sparse A*[7].

2.5 Artificial Potential Fields

Artificial Potential Fields uses a simplistic approach to collision avoidance. In physics, particles have positive and negative charges, where like charges repel each other and opposite charges attract each other. Recently, researchers have applied this theory to collision avoidance, developing Artificial Potential Fields.

APF uses physics to map out the environment into particle charges. Using this approach, each aircraft is considered to have an artificial negative charge. Each goal, or waypoint, is given a positive charge in order to attract the UAV. Likewise, each obstacle (other UAVs or land-obstacles) is given an artificial negative charge in order to repel the UAV. To determine a collision avoidance maneuver, all of these charges are summed together to form a resulting vector. This resulting vector can be applied to direct the UAV into safe airspace.

APF takes significantly less time to execute then most other algorithms. This is due to the simplicity of the algorithm. APF is a reactive algorithm, therefore it makes a greedy decision based on the current information at a specific point in time. Also, APF doesnt make decisions about the future, unlike A* and MILP. Although this tremendously decreases the computation time, it can lead to situations that APF cannot solve.

2.6 Inverse Proportional Navigation

Similar to APF, Inverse Proportional Navigation uses a simplistic approach to collision avoidance. Although, rather than using particles forces, IPN uses geometry. IPN was adopted from a well-known algorithm used primarily by the military, proportional navigation. PN is commonly used to guide missiles. PN is based on the fact that two moving objects have a straight-line distance between them called the Line Of Sight. The PN guidance law applies a lateral acceleration to the missile so that the direction of the LOS vector stays constant[9]. This ensures the missile will collide with the target.

IPN is essentially the opposite of proportional navigation; IPN aims to avoid collision, not create them. Therefore, IPN is concerned with the lateral acceleration that takes an UAV away from the collision[9]. When PN is applied to missile guidance, PN dictates that the missile should rotate at a rate that is proportional to and in the same direction as the LOS rate of rotation. IPN does the opposite; Inverse PN dictates that both objects should rotate at a rate proportional to and in the opposite direction of the LOS rate of rotation.

IPN is very easy to compute. Therefore, IPN is the quickest algorithm covered in this paper. This is due to the simplicity of IPN. IPN uses the easiest approach to collision avoidance, geometry. Also, IPN is purely a reactive algorithm. Therefore, it doesnt compute information about the future, or the past, making it much faster than proactive algorithms. Since reaction time is vital among aircraft, IPN is an obvious choice for optimal performance.

2.7 Conclusion

There are many different algorithms that can solve collision avoidance situations. This paper has briefly described four collision avoidance algorithms: MILP, A*, APF, and IPN. MILP and A* perform optimally when theyre able to fully compute the solution, but many times UAVs are limited in computation time. Therefore, APF and IPN became an obvious choice for further analysis.

APF and IPN both use simplistic methods for collision avoidance; APF maps the field into particle forces, while IPN uses basic geometry to determine the most efficient solution. Both algorithms will successfully solve most, if not all, threatening situations. Although both algorithms are successful, certain algorithms are better in certain situations. Since APF can handle many particle forces, it is better than IPN at handling some multiple-UAV scenarios. Even so, this paper will prove IPN is a much more efficient and quicker algorithm making it the best collision avoidance solution.

Chapter 3
Artificial Potential Fields

3.1 Artificial Potential Fields Basics

The fundamental theory behind APF are physics particle forces. For instance, like charges are used as attractants and opposite charges are used a repellents. Specifically, all aircraft in the airspace will have a negative charge associated with a positively charged destination waypoint[12]. Following the rules of magnetism, the aircrafts (negatively charged) will feel repulsive forces against other aircraft and attractive forces toward their destination waypoint (positively charged)[12]. Each of these forces factor into the final force vector calculation. Then, that vector is used to direct the UAV to safety.

In order to scale the particle forces of attraction and repulsion to accurately mimic an UAV environment, an attraction constant, γ, is used. This attraction constant is used to add weights to the attractive and repulsive forces[13], and thus is bounded such that $0 < \gamma < 1$. Finally, using the attraction constant, the overall force equation can be calculated as

$$\vec{F_{tot}} = \gamma * \vec{F_{attr}} + (1 - \gamma) * \vec{F_{rep}} \tag{3.1}$$

Sigurd and How used this equation in previous research to evaluate their APF implementation. During this research, their best results were achieved using $\gamma = 0.66$ for attractive forces and 0.34 for repulsive forces[13]. The proposed APF solution in this paper follows their implementation using Equation 3.1 and the specified attraction constant.

Equation 3.1 is the method for combining force vectors. To use Equation 3.1, APF needs a method to determine the values of these force vectors. The magnitudes of magnetic forces are primarily determined by the distance between the two objects. These forces can be measured at any distance, and they weaken as the distance increases. However, at some point, the force becomes negligible, so APF has to define a maximum distance at which one UAVs force can be felt, or d_{fmax}.

Unfortunately, UAVs do not operate exactly like physics particles, meaning the force vectors arent solely dependent on the distance between two objects. Therefore, APF must use a much more complex way to calculate d_{fmax}. First, d_{onesec} is used to represent the distance that one UAV can travel in one second, and a scale factor α is used to define the ratio of the collision zone to the size of the potential field. Through experimentation, it was determined an $\alpha = 5$ provided the most realistic results[8]. Furthermore, each UAVs potential field is modified to be an elliptical shape with the force field extending much farther in front of the UAV. This allows APF to closely mimic a UAV environment, since it is more likely that a head-on collision occurs on UAVs. To define this elliptical shape, two constants,λ_{scalef} and λ_{scaleb}, are used to scale the amount of the ellipse that is extended in front and behind the UAV. Through experimentation, $\lambda_{scalef} = 2.0$ and $\lambda_{scaleb} = 1.25$ provided the best results[8]. Finally, θ is used to represent the angle from the heading of UAV, U_k, to the position (excluding heading) of the current aircraft, U_i. Using all of these values, d_{fmax} can be calculated using Equation 3.2. Figure 3.1 represents the force field around a UAV that may be included in Equation 3.2.

$$d_{fmax} = \alpha * d_{onesec}(\lambda_{scalef} - (\lambda_{scalef} - \lambda_{scaleb})/2) + ((\lambda_{scalef} - \lambda_{scaleb})/2) * cos(\theta) \tag{3.2}$$

Along with the maximum distance a UAVs force field can be felt, there also needs to be a minimum distance, dubbed $d_{failsafe}$, which will modify our force value to represent an immediate collision[14]. If two UAVs are within this minimum distance, they both will feel a maximum force, $F_{failsafe}$, which will overpower all other forces in calculations. This will force the UAVs to avoid each other.

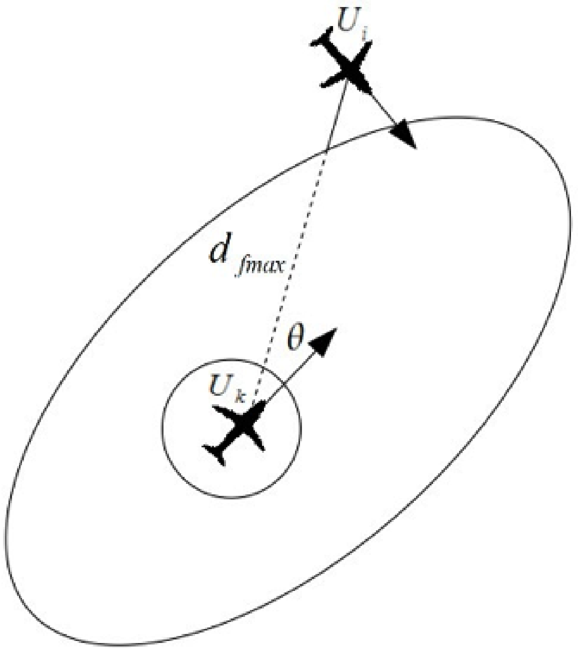


Figure 3.1: Determining the maximum distance of the repulsive force field[8].

Finally, we must consider the case when $d_{failsafe} < d \leq d_{fmax}$. While the UAV is within this range, the repulsive force felt by U_k is dependent on two factors: the distance between U_i and U_k, and the position and direction U_i and U_k are flying[14]. As stated earlier, the distance is the main contributor to the magnitude of the repulsive force. But, the orientation of each UAV also influences the repulsive force. For example, if one UAV is in front of the other, the leading UAV will feel a stronger force to avoid. In contrast, the trailing UAV will feel a weaker force to avoid, because the leading UAV is doing most of the work. In this final calculation, two constants, k_{emitf} and k_{emitb}, will be used to represent the scalar effecting the force in front of U_k and behind it, respectively. Also, the negative charge from a UAV is represented as q_{UAV}. Lastly, to help check the strength of the repulsive force, a scalar value, γ, is used. Using all of these variables and constants, the final force calculation is given as

$$r(\theta, d) = \begin{cases} 0 & \text{if } d > d_{fmax} \\ q_{UAV} * [(k_{emitf} - (k_{emitf} - k_{emitb})/2)] + ((k_{emitf} - k_{emitb})/2) * cos(\theta) \frac{d_{fmax} - d}{\gamma * \alpha} & \text{if } d_{failsafe} < d \leq d_{fmax} \\ F_{failsafe} & \text{if } d \leq d_{failsafe} \end{cases} \tag{3.3}$$

Through experimentation, the best results were provided with $q_{UAV} = 80$, $k_{emitf} = 1.5$, $k_{emitb} = 1.0$, and $\gamma = 4.0$[8].

Now that the final force is calculated, APF must determine how much of this force is felt by the UAV. To calculate this, APF must know three additional values. First, APF uses a variable, called ϕ_{rep}, to represent the angle between the UAVs bearing and the repulsive force. Second, APF uses a scalar, called β_{feelf}, to represent the force felt from an obstacle in front of the UAV. Lastly, APF uses another scalar, called β_{feelb}, to represent the force felt from an obstacle behind the UAV. Using these values Equation 3.4 can be used to increase the force felt from obstacles in the path of the UAV. For a visual representation of ϕ_{rep}, please refer to 3.2. Through experimentation, the best results were provided with $\beta_{feelf} = 1.0$ and $\beta_{feelb} = 0.5$[8].

$$s(\theta, \phi_{rep}, d) = r(\theta, d) * [(\beta_{feelf} - (\beta_{feelf} - \beta_{feelb})/2) - ((\beta_{feelf} - \beta_{feelb})/2) * cos\phi_{rep})] \tag{3.4}$$

Using Equation 3.3 and Equation 3.4, we can infer some vital information. Most importantly, the most powerful forces will be felt when two aircraft are heading directly toward each other. Extrapolating this scenario further, the resulting angles are $\theta = 0$, and $\phi_{rep} = \pi$, which will result in the maximum possible force emitted from the obstacle and subsequently felt by the UAV. In contrast, if two UAVs are flying directly away each other, the force emitted and felt will be minimal.

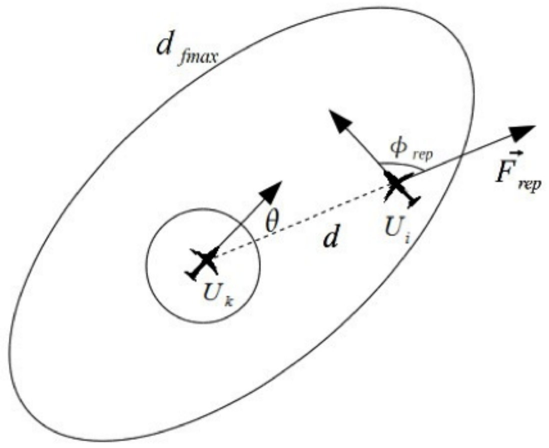


Figure 3.2: Geometry for calculating the repulsive force felt by U_i[8].

After using Equation 3.4 to calculate each repulsive force for a UAV, these values are combined into one value, $\vec{F_{rep}}$, by using Equation 3.5.

$$\vec{F_{rep}} = \sum_{k=1}^{n} s(\theta_k, \phi_{repk}, d_k) \tag{3.5}$$

Finally, APF can calculate the total force acted upon a UAV, and make a decision to safely route the UAV. To do this, APF uses the result in Equation 3.5 and inserts that value into Equation 3.1, along with a constant attractive force, $\left|\vec{F_{attr}}\right|$=100[8]. Using these two values, APF calculates the total force, as shown in Figure 3.3, and uses that force to determine where the U_i should fly during the next time step. The only limitation is the maximum turning angle. If $\vec{F_{tot}}$ is greater than the maximum turning angle, APF will command the UAV to fly at its maximum turning angle. Although, if $\vec{F_{tot}}$ isnt greater than the maximum turning angle, APF will command the UAV to fly at that specific angle.

3.2 Special Situations

Artificial Potential Fields use a simplistic method to achieve collision avoidance. Although using a simplistic method has its advantages, situations arise that are unsolvable or

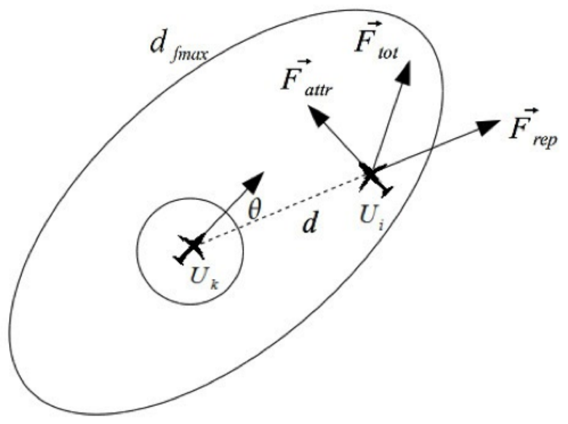


Figure 3.3: Calculating the total force acting on a U_i[8].

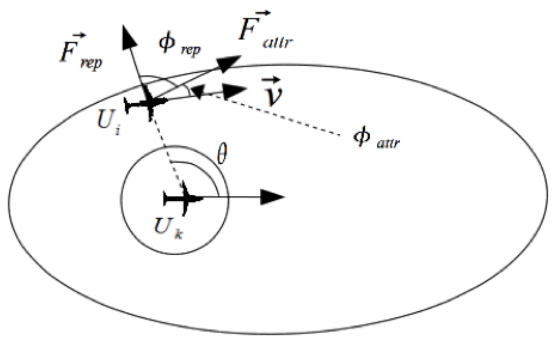


Figure 3.4: Situation in which the right hand turn rule should not be applied. U_i should continue towards its destination instead of traveling behind U_k.

poorly solved by this method. Therefore, APF must tackle these situations individually by looking for these specific scenarios. This section will cover these special situations.

3.2.1 Right-Hand Rule

As UAVs fly their missions, conflicts between them arise. Sometimes, the UAVs need to cross each others path to continue on their mission, and APF must make a decision on how these UAVs should cross. Usually, when APF is looking at a specific UAV, it decides that the UAV should fly in front of other UAVs to continue on its mission. The majority of the time this method provides the most efficient solution, but sometimes it is quicker for the UAV to fly behind other UAVs. Therefore, APF is alerted to execute the Right Hand Rule when this situation arises.

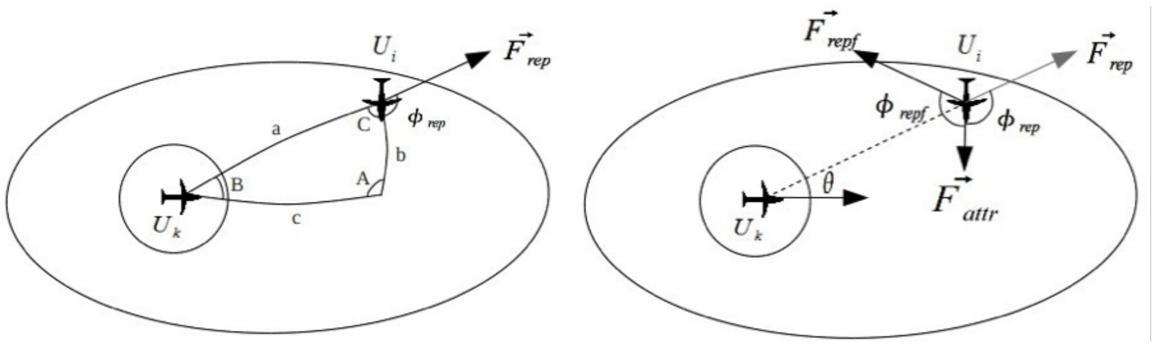


Figure 3.5: Calculating the new ϕ_{rep} to force U_i to fly behind U_k.

To detect this situation, APF calculates three values. The first value, θ, is the angle between the location of the current UAV, U_i, and the bearing of another UAV, U_k. Next, APF calculates ϕ_{rep}, the angle between the current bearing of U_i and the repulsive force acting on U_i from U_k. Lastly, APF calculates ϕ_{attr}, the angle between the bearing of U_i and the attractive force from its goal.

APF uses θ, ϕ_{rep}, and ϕ_{attr} to detect a situation when the Right Hand Rule needs to be applied. If $\phi_{attr} \leq 0$ and $\theta < -90$, then no corrective action should be taken because U_i is both to the left of U_k and turning left away from a collision as shown in Figure 3.4. But, if θ is found to be between [-135, 0], then U_i is to the left of U_k. Next, APF checks if ϕ_{rep} is between [-180, -90]. If it is, then U_i is attempting to turn left to avoid U_k, but this will place U_i in the path in front of U_k, and could potentially cause a hazardous situation[8]. Therefore, the Right Hand Rule must be applied to solve this situation.

If θ is between [-135, -25], then a right turn should be forced such that U_i passes behind U_k. To force this situation, ϕ_{rep} is flipped to make the UAV turn the equal amount, but opposite direction, that it was initially turning. This is shown in Figure 3.5. This will force U_i to avoid a collision by flying behind U_k.

The last unchecked interval is when θ is between [-25, 0]. This scenario is on the verge of whether to avoid, or not to avoid. Therefore, APF uses a spherical triangle to calculate the corrective action. Referring to Figure 3.5, a is defined as the distance between U_i and U_k, b is defined as the distance between U_i and the point of intersection with U_k, and c is defined as the distance between U_k and the point of intersection with U_i. b and c are calculated using the law of cosines and are used for determining the corrective action, If $(cb) \leq (\phi_{rep}90)$, then the same corrective action should happen as if θ is between [-135, -25]. If this isnt the case, it simply means that no corrective action is required, and U_i can safely cross the path of U_k.

3.2.2 Deadlocks

When two UAVs are flying directly at each other, APF enters a deadlock situation where it cannot determine a corrective action[15]. In this case, the repulsive force is pointed directly behind each UAV; no force to the left or right is present to break the deadlock situation. If the UAVs were to continue on their path, the UAVs would simply fly into each other. Therefore, APF detects this situation beforehand and eliminates the deadlock. To determine if two UAVs are on a head-on collision course, APF converts the attractive and repulsive forces into unit vectors and adds them together. If these vectors cancel each other out, then a head-on collision is imminent. To resolve this situation, APF turns each UAV to the right 15 degrees, so they can safely pass each other[15]. The head-on collision situation and APF resolving this situation is shown in Figure 3.6.

3.2.3 Priorities

One of the problems with APF, is when a UAV approaches its destination, repulsive forces from other UAVs can push the original UAV off course. This can cause several negative effects such as looping around the destination, unnecessary turns, or just being pushed completely off course. Although the intervening UAV will eventually fly away, before it does, it potentially could cause hazardous situations for the main UAV.

To solve this situation, APF prioritizes the UAVs based upon how far it is from its destination. To order the priorities, APF uses a variable $d_{priority}$, defined as $d_{priority} = 4.5$ * d_{onesec}. If the distance between the UAV and its destination is within $d_{priority}$, then that

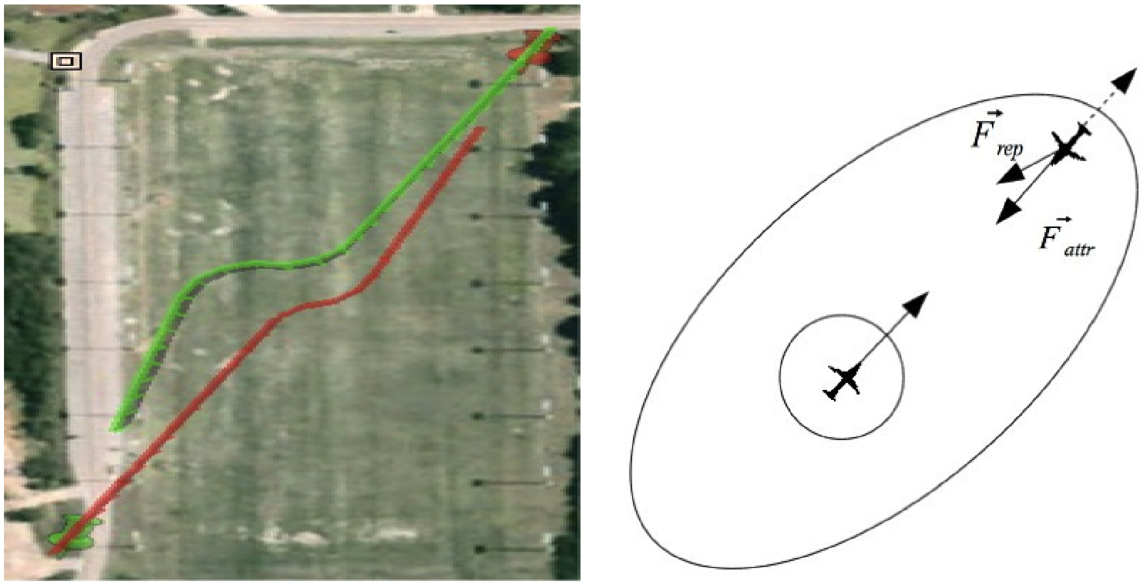


Figure 3.6: Adjusting the UAVs to handle head-on collisions

UAV becomes prioritized. Therefore, when APF calculates the summation for a UAV with priority m, only aircraft that are higher priority are factored into the summation, which solves the issue where lower priority UAVs can alter the path of higher priority UAVs[8]. This is shown in Equation 3.6, which takes the UAVs priority into account when calculating ($\vec{F_{rep}}$).

$$\vec{F_{rep}} = \sum_{k=1}^{m-1} s(\theta_k, \phi_{repk}, d_k) \tag{3.6}$$

Although this solved the problem presented above, it introduces an entirely new problem. Without the priority system, both UAVs make a simultaneous avoidance maneuver, when they need to. But, when the priority system is in place, only one UAV (the one with lower priority) will attempt to avoid. This can lead to conflicts because, usually, both aircraft would avoid each other. To solve this side effect, APF expands the potential field of prioritized aircraft by using scaling factor, called p_{mult}. This alters the potential field to be defined as $d_{fmax} * p_{mult}$[8].

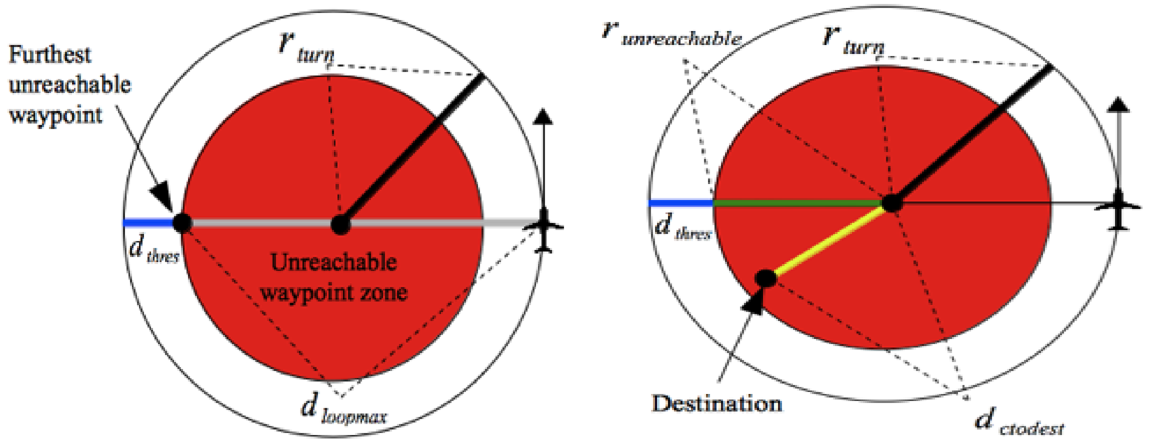


Figure 3.7: The geometry involved in looping detection[8]

3.2.4 Looping

The final issue that arises with APF is indefinite destination oscillations. For example, image a scenario where two sequential waypoints are very close to each other. It could happen where the second waypoint is within an unreachable area that the UAV will never be able to reach by simply turning at the maximum rate in the direction of the waypoint. Therefore, APF developed a way to detect and correct this situation.

APF uses basic geometry to detect if a waypoint is unreachable. First, the waypoint must be within a predefined distance before APF considers the waypoint for looping conditions. This distance is defined as $d_{loopmax} = 2 * r_{turn} - d_{thres}$, where r_{turn} is the UAVs maximum turning radius and d_{thres} is its distance threshold to contact its destination. Figure 3.7 shows the geometry involved in calculating looping. In this figure, the red zone cannot be reached by a normal turn maneuver and the white zone can. If the UAV is within $d_{loopmax}$, then APF calculates the value $d_{ctodest}$, which is the distance between the center of the unreachable zone, c, and the destination. If $d_{ctodest} \leq r_{unreachablezone}$, then the UAV cannot reach the destination with a typical turn maneuver. To fix this situation APF simply flips the polarity of the destination to repel the UAV until $d_{ctodest} \geq r_{unreachablezone}$.

Chapter 4

Reactive Inverse Proportional Navigation

4.1 Background

Reactive Inverse Proportional Navigation Algorithm is an effective solution to collision avoidance. RIPNA solves many issues that arise with other collision avoidance algorithms like computation time, routing efficiency and most importantly, decreasing collisions. It does this by combining two unique algorithms for two specific purposes; IPN is used to determine collision avoidance maneuvers and Dubins path is used for non-collision scenarios.

4.2 Greatest Threat Identification

The first step of RIPNA is treat identification. RIPNA surveys the airspace looking for UAVs on collision courses. This is determined by examining each UAV and using a specified separation distance to choose other UAVs imposing a threat. Also, each UAV is assumed to continue on its current trajectory. Using geometry to project each UAVs vectors, the ideal minimum distance or Zero Effort Miss, is calculated. To calculate the ZEM of a pair of UAVs, consider the 2-D engagement scenario of two UAVs, U_1 and U_2. To help calculate the ZEM, let $p_1 = (x_1,\, y_1)$ and $p_2 = (x_2,\, y_2)$ represent the initial positions of U_1 and U_2 respectively. Also, let $d_1 = (l_1,\, m1)$ and $d_2 = (l_2,\, m2)$ represent the direction cosines of the respective headings of U_1 and U_2. Supposing both UAVs travel at constant speed V and bearing, the distance S between them can be calculated as a function of time, given as

$$S^2 = (p_1 - p_2) * (p_1 - p_2) + 2(p_1 - p_2) * (d_1 - d_2)(Vt) + (d_1 - d_2) * (d_1 - d_2)(Vt)^2 \quad (4.1)$$

To find the time at which U_1 and U_2 are closest to each other, RIPNA finds the time at which S is the smallest, by solving for t when $\frac{ds}{dt}$=0. George and Ghose[9] call this the t_{go}. It may be calculated as

$$t_{go} = \frac{-((p_1 - p_2) * (d_1 - d_2))}{(V(d_1 - d_2) * (d_1 - d_2))} \tag{4.2}$$

Now, RIPNA can calculate the ZEM as S(t_{go}). If $t_{go} > 0$, then there exists a point in the future that U_1 and U_2 will collide. Also, t_{go} can be less than 0, which means that U_1 and U_2s paths extrapolated backwards results in a minimum distance and thus a negative t_{go}.

Each UAV compiles a list of other UAVs in its neighborhood, meaning each UAV will only consider other UAVs within a distance of R_{des}. After compiling this list of other UAVs and their corresponding ZEM and t_{go}, RIPNA can make a decision on which UAV to avoid. In order to find the greatest threat, RIPNA uses a greedy selection; from the list of threats, RIPNA chooses the UAV with the smallest positive t_{go} as the greatest threat, and the current UAV will perform a maneuver to avoid that UAV.

4.3 Collision Avoidance Maneuver Determination

After selecting a great threat, a collision avoidance maneuver must be performed. Figure 4.1 depicts a 2-D engagement of 2 UAVs, U_1 and U_2. Assume U_1 and U_2 have selected each other as greatest threats. To ensure a safe maneuver, each UAV will turn to increase the magnitude of the rate of rotation of their LOS. Let $\theta 1$ and $\theta 2$ be the angles between the LOS and heading of U_1 and U_2. The LOS rate of change is given as

$$\theta = \frac{(Vsin(\theta_2) - Vsin(\theta_1))}{r} \tag{4.3}$$

In this scenario, if $(\theta_2 > \theta_1)$, then accelerations a_1 and a_2 are applied to U_1 and U_2, respectively. Using these two accelerations, the UAVs will safely avoid their collision course. The magnitude of the accelerations is given as

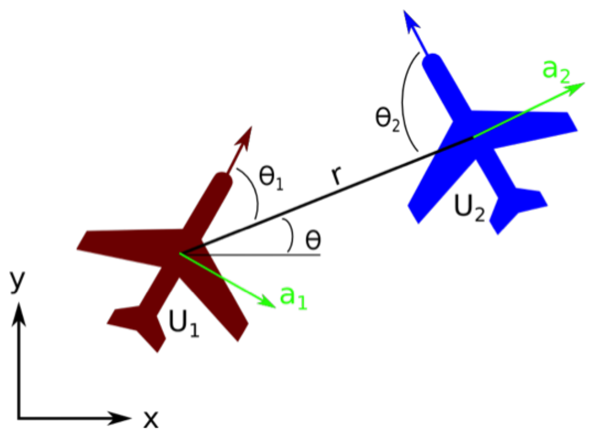


Figure 4.1: An engagement of 2 UAVs in the 2-D plane where $\theta_2 > \theta_1$[9]

$$a = \frac{V^2}{R} \tag{4.4}$$

where R is the calculated turning radius for the UAVs. George and Ghose[9] suggest an equation for the computation of R, by using the exponential function

$$R = R_{min} * exp(\frac{\lambda * ZEM}{R_{des}}) \tag{4.5}$$

where R_{min} is the minimum turning radius of the UAVs and λ is a tuning parameter.

In the scenario in Figure 4.2, when ($\theta_2 < \theta_1$), accelerations are applied in the opposite direction depicted in Figure 4.2. This will effectively execute the most effective collision avoidance maneuver. Likewise, with a ZEM of zero, the UAV is expected to perform the tightest turn possible. For a higher ZEM, there is a lower risk of a collision; therefore, it is desirable to minimize the deviation from the normal path by lessening the turning radius. One way to achieve this is by using an exponential function, like Equation 4.5, to achieve a demand radius of R.

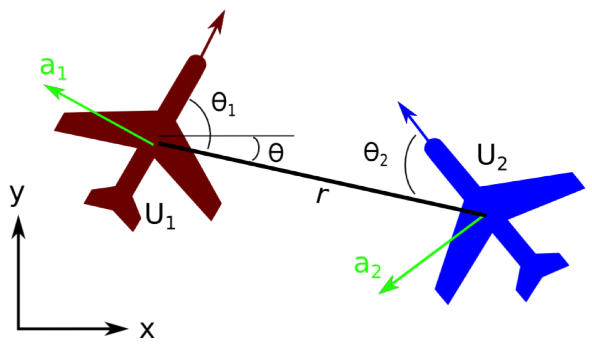


Figure 4.2: An engagement of 2 UAVs in the 2-D plane $\theta_2 < \theta_1$[9]

4.4 Dubin's Path

If a UAV doesnt have an immediate threat, it is routed to its next waypoint using Dubins path. Dubins path provides the shortest distance between two points, given the initial heading and destination heading. When implementing Dubins path in UAV airspace, the destination heading isnt critical, given that the destination is a waypoint rather than a runway. Therefore, the destination heading isnt considered, which alters Dubins path to simply make the UAV turn as sharply as possible until it is headed toward its destination. However, this may cause the UAV to repeatedly circle its destination if the waypoint is within the circle of UAVs minimum turning radius. This is illustrated if Figure 4.3. In this example, turning along R_1 towards destination 1 would require the UAV to circle its destination. Therefore, Dubins path has been altered to allow the UAV to fly along the circle R_2 to reach destination 1. This simply requires the UAV to slightly turn left until the destination is no longer inside the circle of the minimum tuning radius, at which the UAV turns along the circle of its minimum turning radius (R_2) to reach its destination.

4.5 Special Situations

After Fish and Westman implemented RIPNA, they experienced some special situations that RIPNA solved poorly[16]. These situations may not have presented themselves prior to Fish and Westmans research due to the simulation environment (i.e. short run-time, random

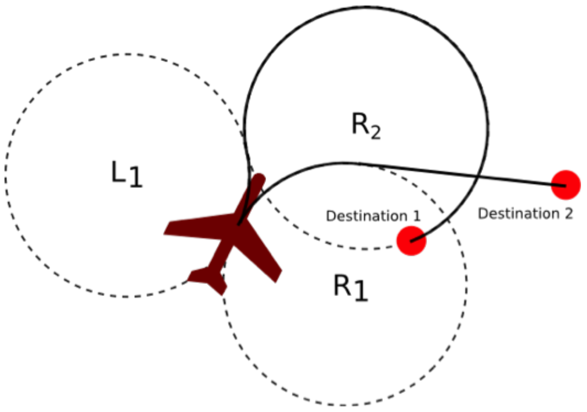


Figure 4.3: A UAV taking Dubins path to one of the two possible destinations[9]

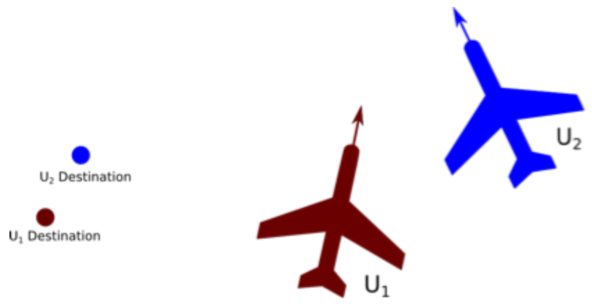


Figure 4.4: A situation that induces chattering between U_1 and U_2[16]

destination arrangement, one destination per UAV, etc.). To solve these situations, special detection and correct techniques were integrated into the existed RIPNA implementation[16].

4.5.1 Chattering

One of the most common situations occurs when two UAVs are traveling with similar headings, they are close to each other, and they both need to turn in the same direction to their destination. This causes the UAVs to oscillate from avoiding each other to trying to fly to their destination. A geometry that can cause chattering is shown in Figure 4.4.

Once chattering occurs, it usually pushes the two UAVs several hundred meters off course until the oscillation pattern is broken. This greatly lowers the efficiency of RIPNA, which could make other collision avoidance algorithms falsely rank higher than RIPNA during evaluation. Therefore, Fish and Westman took a preventative approach to solving this problem: two UAVs with similar headings (within 30 of each other) will not attempt to avoid each other unless the LOS distance is below a provided threshold[16]. This will allow the UAVs to orient themselves by traveling further toward their destination, rather than initiating the chattering situation. This will resolve most long-term chattering situations with minimal restrictions to RIPNAs overall objectives.

4.5.2 Inaccurate Threat Selection

Inherently, RIPNA is amazingly good at solving two-UAV scenarios. Although, when RIPNA is dealing with multiple-UAV scenarios, it has trouble accurately choosing the correct UAV as the greatest threat. This can lead to unexpected near misses and a reduction in flight-path efficiency. For example, consider the three-UAV scenario shown in Figure 4.5. When RIPNA is selecting the greatest threat for U_1, it will consider U_2 and U_3, because their ZEM values are less than the desired separation. Then, RIPNA would select U_3 as $U_1's$ greatest threat because $U_3's$ t_{go} is less than $U_2's$, but this is an inaccurate threat selection. It would be in the best interest of U_1 to avoid U_2, because even though its t_{go} is much larger than that of U_3, its ZEM is less than the near miss radius, which makes it significantly more dangerous than U_3.

To solve this situation, Fish and Westman implement a basic two-threshold system. This gives RIPNA a method for determining if a smaller ZEM is enough to trump the smaller t_{go}, like in Figure 4.5. To implement this fix, Fish and Westman use the near miss radius as one of the thresholds. If one or more of the UAVs being considered have a ZEM less than the near miss radius, then the UAV with the smallest t_{go} will be considered the greatest threat. Otherwise, RIPNA performs as it would regularly. This two-threshold system is not a complete solution, although it does offer a preventative approach to minimize the near misses that are due to inaccurate threat selection[16].

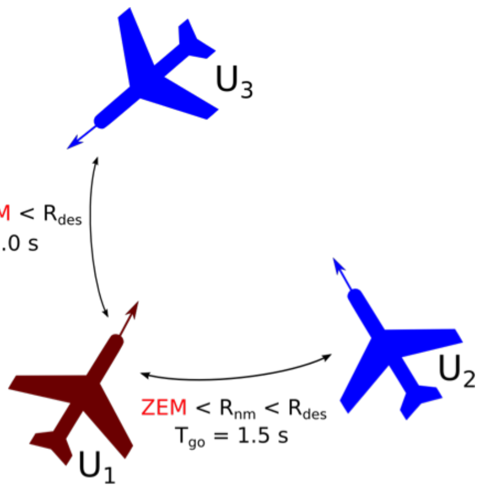


Figure 4.5: Three-UAV scenario where RIPNA performs inaccurate threat selection. R_{nm} is the near miss radius and R_{des} is the threshold ZEM value for avoiding UAVs

Chapter 5
Test-bed Design

5.1 Test-bed Requirements

One of the key requirements to perform this research is having 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. This would allow for easy comparisons between multiple collision avoidance algorithms. Additionally, this system needed to work with a pre-existing, 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 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. Ability to perform laboratory simulation
4. Ability to perform field tests with real UAVs
5. Collect and visualize GPS data
6. Provide an interface for loading waypoints, paths, or courses into the system

James Matt Holt, a previous Masters student under Dr. Saad Biaz, developed part of this system for his Thesis project. Matt was able to complete items 1., 2., 3., and 6., and compare three collision avoidance algorithms: MILP, Sparse A*, and APF[11]. Although Matt made a significant dent in the overall project, there was still plenty of work left to finish the test-bed and collision avoidance testing, therefore I was tasked with with completing items 4., 5., and further testing new collision avoidance algorithms in an effort to out perform Matt's final results.

5.2 Hardware Considerations

5.2.1 Ground Control Station Hardware

The most important hardware consideration is the Ground Control Station. The GCS is the brains of the operation; It needs to be able to monitor all of the UAVs, while making decisions on where and when to route each UAV. Since the architecture is discussed in detail below, this section will only touch on the hardware involved in the GCS.

Most importantly, the GCS needs to be run on a computer, and if real UAVs are being flown, the GCS needs to be in-range of these UAVs inorder to undergo communication. Therefore, a lightweight, transportable, yet powerful laptop is the best choice. This laptop needs to be able to run a Linux OS, e.g. Ubuntu, because the GCS is written on top of the Robot Operating System. Personally, I use a Linux virtual machine to operate the GCS, but any computer running a Linux OS that has ROS configured correctly will suffice.

Secondly, the GCS computer needs to be able to communicate with the real UAVs. This is done by using a Xbee telemetry module that is connected to the GCS computer. The Xbee module is a product made by the company Digi International that is a lightweight arduino-based board which performs wireless communication. Xbees allow for point-to-point, point-to-multipoint, and multipoint-to-multipoint communcation, between two or more Xbee modules. The Xbee product was chosen for this research because the modules are cheap (30 per module), they provide a very long range (1-2 miles), and they are easily configured to a desired network configuration. Therefore, every real UAV (fixed wing or multi-rotor) are equipped with a Xbee telemetry unit, forming a point-to-multipoint network configuration.

Lastly, since the GCS is equipped with Google Maps visualization, an internet connection is needed for the visualization to operate correctly. For future endeavors, a 3G/4G WiFi hot-spot should be purchased so that the GCS can operate anywhere that has cellphone reception, since recently we have been using phone-tethering which will not always be available. Another idea is to cache the field's data beforehand, then when operating in the field WiFi will not be required. Furthermore, it should be noted that the fundamental operation of the GCS is not hindered by the WiFi requirement; the GCS can operate perfectly without WiFi, but the visulation aspect of the GCS requires the internet connection.

5.2.2 UAV Hardware

As this research has progressed over several years the UAV platform has also evolved from fixed-wing aircraft to multi-rotor aircraft. Initially, the research was all done using a basic R/C fixed-wing aircraft equipped with an autopilot control chip. More recently, the research has progressed to using multi-rotor aircraft, also equipped with an autopilot control chip. This transition was done in an effort to further the research into the latest technologies available, test collision avoidance algorithms among all system available, and because multi-rotor UAVs are much easier to pilot than fixed-wing UAVs.

5.2.3 Fixed-wing Hardware

The fixed-wing hardware is essentially a remote controlled airplane equipped with smart sensors. The airframe for the fixed-wing is a Bixler model that is controlled by a 6 channel DX6i Spektrum remote. In order to have autonomous flight, the airframe needed to be equiped with an autopilot mechanism, which is obtained by using the ArduPilotMega, a.k.a APM. The APM is an arduino micro-controller that controls the servos onboard the Bixler acting as the autopilot for the autonomous system. The APM has a lot of sensors onboard to achieve a fully autonomous system. Some of the sensors include a compass, barometer, accelerometer, gyroscope and a GPS. The APM uses these sensors to read the current orientation of the aircraft and then manipulates the servos accordingly to keep the Bixler in the air. Furthermore, the UAV is equipped with a Xbee telemetry unit for communication between the aircraft and the GCS. With all of these parts combined, the fixed-wing UAV is able to fly waypoint-to-waypoint missions, while avoiding other aircraft, which is all directed by the GCS.

5.2.4 Multirotor Hardware

The multi-rotor hardware contains a lot of the same hardware involved in the fixed-wings, but a much different airframe. The airframe was bought from 3DRobotics, and the DIYDrones community. It is called the IRIS, but essentially its a basic quadcopter, meaning it uses four propellers to keep the aircraft flying. With that being said, quadcopters are nearly impossible to fly manually, so they always have a micro-controller attached to help with the flight stabilization. The IRIS uses the Pixhawk micro-controller, just like the fixed-wing uses the APM. Likewise, the Pixhawk has many of the same sensors available, e.g. magnetometer, barometer, accelerometer, gyroscope, GPS. Like the APM, the Pixhawk uses these sensors to maintain stabilization during flight. Also, like the fixed-wings, the IRIS is equipped with the same Xbee telemetry unit so that it too can communication with the GCS, which gives the IRIS the ability to fly waypoint-to-waypoint missions.

5.3 Software Considerations

The software is involved in two locations: the GCS and the aircraft. The main focus is to have these two pieces of software communicate, therefore they have to talk the ”same language”. This language is the Micro Aerial Vehicle protocol, called MAVLink. This protocol is widely used among amateur UAV enthusiasts, was already implemented on the autopilot system we were using, and was easily implementable on the GCS, so we decided this was the best protocol for our system.

The aircraft has one huge piece of software: the autopilot system. For the fixed-wing and multi-rotor hardwares, this system is innately different, and called the ArduPilot and ArduCopter, accordingly. They are different because the flight dyanamics of a fixed-wing aircraft and quadcopter are very different, and the software components are tailored to a specific system. With that being said, the communication of the two is exactly the same, which is good for the needs of this research. Therefore, the only software consideration for the aircrafts are the communication message that will be used on the GCS and the aircraft. For this research the AU_UAV message was implemented on both the GCS and the aircraft. This message contains the following information:

1. Current latitude
2. Current longitude
3. Current altitude in meters
4. Target latitude
5. Target longitude
6. Target altitude in meters
7. Current ground speed in meters/second
8. Current airspeed in meters/second
9. Desired bearing to wp in degrees
10. Distance to target waypoint in meters
11. Target waypoint index

On the GCS there are a lot more software considerations in order to achieve all of the needs of this research. First, and Foremost, the GCS needs the ability to test multiple collision avoidance algorithms in the same environment, and provide results from these tests. Therefore, the GCS will need to have a simulation mechanism so the environment can be held constant while different collision avoidance algorithms are tested. Second, and almost as important, the GCS needs to monitor all the aircraft (simulated and real) and direct them during flight. This requires the GCS to keep track of all the current aircraft, monitor them to see when they need to be sent to their next waypoint, or avoid another aircraft, and direct the corresponding aircraft to the correct location. Finally, the GCS needs to be able to communicate to real UAVs, which requires the GCS to have a Xbee telemetry unit, and software that receives the AU_UAV message sent from each aircraft.

All in all, the Robot Operating System allows for the GCS to encapsulate all of these requirements. The Robot Operating System is built for programs that need multiple threads running simultaneously, performing different operations. In robots, this allows for the robot to be able to move, while using on-board sensors to collect data and make decisions. In terms of a Ground Control Station, there are many separate tasks that need to be running simultaneously. As stated above, the GCS needs to be able to receive incoming information from real UAVs using the wireless Xbee unit, while monitoring all of the UAVs, sending them waypoints and collect data for collision avoidance evaluation. Therefore, ROS is the perfect platform to encapsulate the GCS, since it gives the ability to have all of these tasks running at once.

5.4 Ground Control Station Architecture

5.4.1 ROS Overview

The GCS is responsible for coordinating all of the aircraft while maintaining a safe flight; therefore ROS is a perfect fit for aiding in implementation. ROS is a framework used for controlling robots, because it splits up specific tasks into their own processes, which can run independently of each other. Imagine a robot that has three sensors: infrared, sonar, and infrasound. Each sensor would be controlled by their own process, which allows for the user to run each sensor independently of each other. The advantage of ROS is that the sensors can easily be dynamically started and stopped without failure of the entire system.

The ROS framework is used for the ground control station in the RIGORS project. The GCS will be used to perform extensive test flights to collect data for real and simulated UAV missions. Furthermore, the GCS can be used to test multiple collision avoidance algorithms and compare the differences in each algorithm. This will allow for real UAV testing in order to compare collision avoidance algorithms in a real environment.

5.4.2 ROS Components

ROS has three main components: nodes, services, and topics. These three components make up the entire ROS system and allow for a developer to create anything from a personal autonomous cleaning robot, to an UAV ground control station.

5.4.2.1 Nodes

Nodes are the processes that run within ROS. Each node has a specific task for it to perform. Because of this partitioning, most ROS based programs require multiple nodes to fulfill all requirements of an overall program. The node structure allows for the user to easily swap out nodes to perform different tasks. For example, if there are two nodes both tasked with the collision avoidance for vehicles, the nodes can be used interchangeably even though they may use completely different algorithms to calculate avoidance maneuvers.

5.4.2.2 Services and Topics

Services and topics are used to communicate information in ROS. Topics are the many-to-many communication protocol in ROS. Topics are a lot like a bulletin board; anyone can post information to the bulletin board (given a specific format), and anyone can read from the bulletin board. Therefore, there can be multiple publishers and multiple subscribers to each topic. Services are the one-to-one communication protocol in ROS. Services are issued by a node to another node, usually asking for specific information that the issued node has. For example, imagine if one node is controlling the movement of the robot and another node is controlling a sonar sensor. The movement node will issue a service to the sonar node, before moving the robot, in order to keep the robot a safe distance from any objects.

5.4.3 GCS Nodes

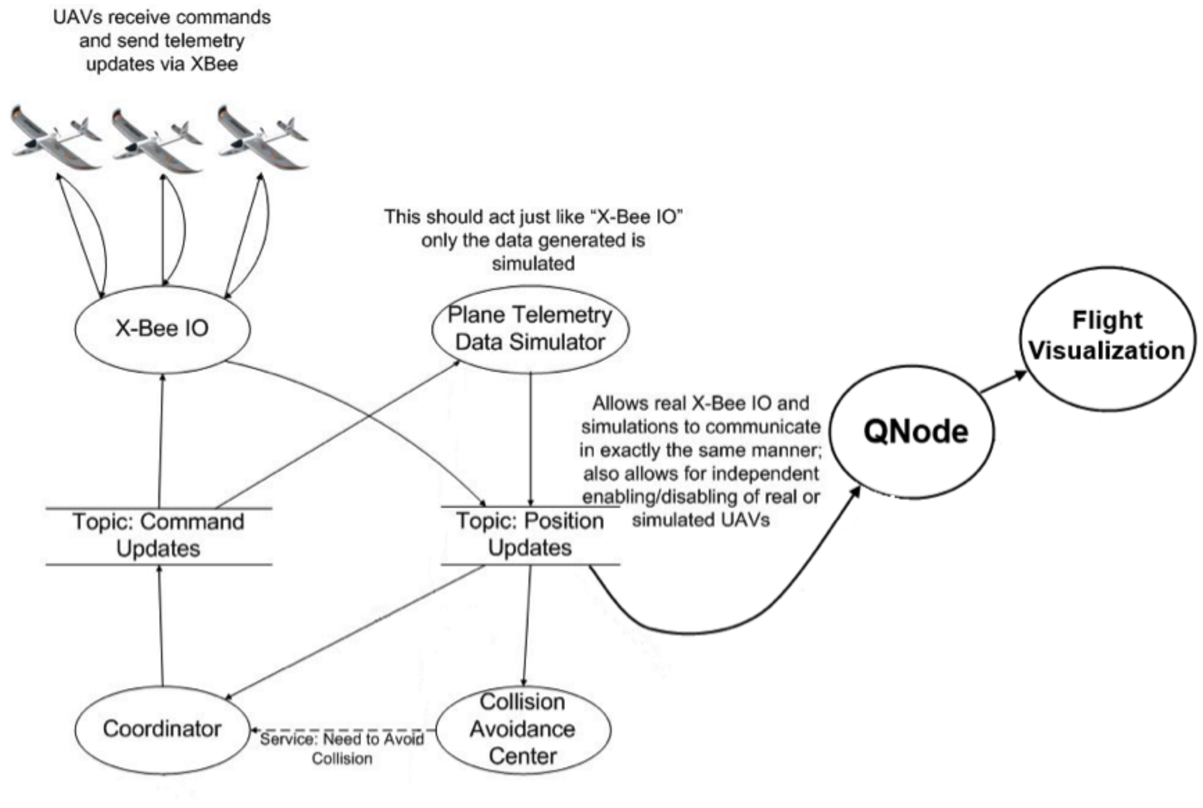

UAVs receive commands and send telemetry updates via XBee
This should act just like "X-Bee IO" only the data generated is simulated
X-Bee IO
Plane Telemetry Data Simulator
Flight Visualization
QNode
Allows real X-Bee IO and simulations to communicate in exactly the same manner; also allows for independent enabling/disabling of real or simulated UAVs
Topic: Command Updates
Topic: Position Updates
Coordinator
Service: Need to Avoid Collision
Collision Avoidance Center

Figure 5.1: The Ground Control Station Architecture

5.4.3.1 Coordinator

The coordinator is the core node that keeps order in the GCS. It communicates to the Simulator, Collision Avoidance, Xbee In, Xbee Out and the Visualization node. Specifically, the coordinator is in charge of monitoring all of the active vehicles so that they fly their desired mission safely and efficiently. In order to insure safety, the coordinator passes information to the collision avoidance system. This allows the coordinator to issue collision avoidance commands, if needed. To insure efficiency, the coordinator is constantly monitoring the active vehicles to see if they have hit their desired waypoint in order for the vehicle to continue on its mission.

When the system is started, a course file is read into the coordinator to start the mission. Each active UAV will have a set of waypoints to fly to. The coordinator is in charge of issuing the commands to send each UAV to the correct location. Likewise, whenever a UAV has hit its desired waypoint, the coordinator will remove that point from the mission queue and issue a command to send the UAV to its next location through the commands topic.

Aside from the normal missions, the coordinator is in charge of issuing collision avoidance commands. The coordinator has a callback function that is executed every time a telemetry message is published to the telemetry topic. The coordinator passes the telemetry information off to the collision avoidance algorithm, which determines if a collision avoidance maneuver is necessary. If one is deemed necessary, the coordinator issues the collision avoidance waypoint through the commands topic.

5.4.3.2 Simulator

Since the major purpose of the system is to test collision avoidance algorithms, the simulator was built to simulate an UAV environment. Very similar to real UAVs, the simulator is a publisher to the telemetry topic, and a subscriber of the commands topic. This is done in order to mimic real UAVs as closely as possible; real UAVs publish their current information to the telemetry topic and receive commands from the commands topic. One of the beneficial aspects of this implementation is the coordinator cannot distinguish the difference between the real and simulated UAVs through the communication protocol. Therefore, the coordinator can communicate to real or simulated UAVs in the same fashion, which is very beneficial for implementing the coordinator. Furthermore, this functionality provides the ability to fly real and simulated UAVs simultaneously.

The functionality of the simulated was built to mimic real UAVs as closely as possible. The simulator takes the estimated cruise speed and generates a telemetry update based upon that speed. However, when the simulated UAVs need to make a turn, a maximum turning angle is applied, which for the testing purposes of this project is set to 22.5. Therefore, over a 4 second time frame a UAV is able to turn 90. Also, its important to bring to light that the current implementation is simplistic; the UAV telemetry simulation doesnt factor in the current weather such as the wind conditions. With that being said there have been successful efforts in implementing wind into the telemetry simulation, although it has not been fully integrated at this moment.

5.4.3.3 Collision Avoidance

The collision avoidance system maintains the slew of collision avoidance algorithms. The coordinator node calls the collision avoidance system every time the coordinator receives a telemetry update. The collision avoidance system uses every active aircrafts most current telemetry update to decide which UAVs are in danger and should be issued an avoidance command. This implementation is useful because the coordinator doesnt know which collision avoidance algorithm is under operation, which allows for low cohesion within the GCS, meaning the coordinator doesnt have to alter its functionality depending on which collision avoidance algorithm is being used. Also, this allows for different collision avoidance algorithms to be interchanged easily and tested using the same UAS mission, which leads to better analysis of the collision avoidance algorithms.

5.4.3.4 Xbee IO

Two nodes are in charge of communicating with the UAVs. These two nodes are called Xbee In and Xbee Out. As you would think, Xbee In handles the incoming communication, while Xbee Out handles the outgoing communication.

The Xbee In node uses the Xbee telemetry unit and setups a communication socket to receive information from each UAV. The GCS uses the MAVLINK communication protocol to interface with the UAVs used for testing. Since MAVLINK is a commonly used protocol, this allows the GCS the ability to integrate any other vehicles using the same MAVLINK protocol. Once the Xbee In node receives a message from a UAV, that message is deciphered and posted on the telemetry topic for other nodes to use.

The Xbee Out node is in charge of sending commands to each UAV. The Xbee Out node subscribes to the commands topic and reads any commands posted to this topic. If the command is for itself, then Xbee Out will read the commands and send it to the corresponding UAV. Common commands are Go to this waypoint, Change your telemetry rate to X Hz, or Change your ID to X. The Xbee Out node also uses the MAVLINK protocol, so that the UAV recognizes the commands that are sent.

5.4.3.5 Visualization

The flight visualization is using the QT software development kit. It is built as a front-end graphical user interface, such that the ROS component is the back-end. Although, there is a node in the back-end called QNode, which is the interface to the QT-built GUI. QNode simply subscribes to the telemetry topic and relays that information to the QT interface. This allows the QT component to visualize each UAV on a map. QNode can also the driver of the system, allowing the user to start a mission through the GUI. In order to do this, QNode issues a service to the coordinator to load a specific mission. There is a lot of improvement available for the visualization component that I think could drastically improve the ability of the system.

5.4.4 GCS Messages

The messages within the GCS are the second most vital piece of the GCS. The messages allow the Nodes to operate smartly. Within the information in each message, each Node would be a stand-alone process. The advange to using ROS is the communication that can happen between Nodes, and the messages propagate this information.

There are two main messages in the GCS; the Telemetry update and the commands message. These two messages are covered in detail below.

5.4.4.1 Telemetry Update

The telemetry update message contains all of the information from each UAV, real and simulated. This message contains vital information unique to each UAV and is implemented as a topic in ROS. Remember, topics in ROS give each node the ability to monitor the information posted to the topic, and post information to the topic. The Coordinator node, as well as other nodes, need this message to keep track of each UAV. Specifically, this message contains the following information:

1. Current latitude
2. Current longitude
3. Current altitude in meters
4. Target latitude
5. Target longitude
6. Target altitude in meters
7. Current ground speed in meters/second
8. Current airspeed in meters/second

9. Desired bearing to wp in degrees

10. Distance to target waypoint in meters

11. Target waypoint index

The rate at which this information is sent is dependent on the user, but currently each UAV sends a telemetry update message every second. This is one of the most time dependent aspects of the GCS, because if an update is lost or isn't sent on time, two UAVs could crash into each other.

Furthermore, one of the most rewarding aspects of the GCS is the way these messages are received. In terms of the Coordinator node, there is not a different between a real UAV's telemetry update and a simulated UAV's telemetry update, which provides a very realistic evaluation environment.

5.4.4.2 Commands

After the telemetry updates have been received and a UAV either needs to avoid a collision, or proceed to a mission waypoint, the GCS needs a way to send commands to each UAV. This is handled by the commands message. This message, like the telemetry update, is implemented as a topic so that multiple nodes can post and receive the command information.

The commands message contains the following information:

1. Plane ID

2. Simulation

3. Command ID

4. Target latitude

5. Target longitude

6. Target altitude in meters

The Plane ID is self explanatory - each UAV has a unique plane ID number. The simulation component of the commands message is a boolean item that specifies whether or not the command is for a simulation UAV. This is done in order for the Xbee IO Nodes to know whether or not to send out a MAVLink message with the command. Moreover, if the command is not for a simulation UAV, the simulation node will not process the command. The command ID component of the command message is used for sending a specific command to real UAVs. The command ID can take on four values:

1. COMMAND_NORMAL_WP
2. COMMAND_AVOID_WP
3. COMMAND_SET_ID
4. COMMAND_SET_STREAMRATE

COMMAND_NORMAL_WP and COMMAND_AVOID_WP are used to send waypoints to the UAVs - avoidance waypoints and mission waypoints. COMMAND_SET_ID is used to change the ID of a UAV, and COMMAND_SET_STREAMRATE is used to alter the frequency at which the telemetry update message is sent down.

Finally, the target latitude, target longitude and target altitude are used to direct the UAV to a target waypoint. This is used by the COMMAND_NORMAL_WP and COMMAND_AVOID_WP commands. Specifically, the target waypoint is used to direct any UAV to any location. Moreover, this is how the GCS manipulates the flights of the UAVs.

Chapter 6
Results

As mentioned in the Introduction, James Holt, a previous Master's student that worked for Dr. Biaz, spent multiple years conducting research for the ATTRACT project. During his research Mr. Holt produced many significant results, which are the foundation for the research presented in this paper. Most importantly, Mr. Holt evaluated three algorithms: Mixed-Integer Linear Programming, Dynamic Sparse A-Star, and APF. While MILP, and DSAS outperformed APF in some minor metrics, the most important metrics, Survival Rate, and Efficiency, were dominated by APF. This resulted in an overall recommendation of APF, as it was the "winning" algorithm among the three. Therefore, the research in this paper builds upon Mr. Holt's findings to improve upon APF's results, and consequently improve upon the results on MILP and DSAS as well. With that being said, this section will expand upon the results of APF and RIPNA, and will show that RIPNA is the clear winner among most all the evaluation metrics.

Four metrics were used to evaluate the two collision avoidance algorithms. These metrics were aimed at evaluating every aspect of the algorithms, in order to highlight the key points where a collision avoidance algorithm should succeed, and to evaluate the efficiency of each algorithm. First, and foremost, a collision avoidance algorithm should avoid collisions, therefore the number of collisions is the primary metric. Likewise, the number of conflicts is extremely important, therefore it is also recorded. The efficiency of the algorithms are further tested by evaluating the deviation from the straight line distance, which is calculated as $\frac{actual-distance-traveled}{minimum-possible-distance}$. This provides a solid metric to evaluate how much extra work the algorithm is requiring. Furthermore, the total number of waypoints reached is evaluated. Obviously, if some planes collide this value will be drastically affected, but in a traditional waypoint mission scenario, this is the main objective, therefore this is a good metric in evaluating the performance of the algorithms.

These four metrics were collected in a simulation environment, using many different constraints. First off, each simulation runs for 600 seconds, or 10 minutes. Since 24 simulations were executed, this meant that each algorithm takes four hours to simulate. Therefore, a total time of 12 hours is needed to simulate the baseline - no collision avoidance - scenario, and the two collision avoidance algorithms. Each of the 24 simulations were executing using two main constraints: the field size and the number of aircraft. The field size varied between a 500x500 meter grid and a 1000x1000 meter grid; Half of the simulations were run with one size where the other half ran with the other size. Using a smaller and larger grid provides the ability to evaluate how well the algorithms work in a confined space as well as a more spacious environment. The number of aircraft were varied from 4, up to 32, doubling for each increment, meaning the number of aircraft is 4, 8, 16, 32. This seems to be a pretty exhaustive range of aircraft, since anything lower than four aircraft is trivial and anything higher than 32 aircraft is unlikely. Furthermore, the values in-between (6,12...etc) are not tested, which is a limitation of the overall simulation. This is left out due to the lengthiness of the simulations, although the results do show a trend which would follow if these values were simulated. All in all, the range used on the number of aircraft provide a very good way to evaluate the ability for the algorithms to perform in a extremely crowded airspace, as well as the ability to evaluate how the algorithms perform when the airspace is spacious.

These tests present a wide range of stressful situation to these algorithms. First, on a small 500m field, its 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 the aircraft in time which may result in conflicts and/or collisions. Finally, waypoints in these courses were randomly generated to present the challenge of flight paths 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.

6.1 Number of Collisions

As described earlier, a collision occurs when two aircraft are within one second of each other. For example, if all aircraft are traveling at 12 m/s, the collision distance is defines at 12 meters away from each other. Rather than simply seeing if two aircraft are at the same position, this allows for a collision radius, which allows for a more realistic simulation since aircrafts would usually collide with extremities of the aircraft rather than in the central location of the vehicle.

The number of collisions is a very important metric in evaluating the efficiency of a collision avoidance algorithm, since, as you would expect, the main goal of the algorithm is to ensure safety. The number of collision for APF and RIPNA will be compared to the baseline, which is using no collision avoidance. Using this baseline shows the overall improvement in the system, and well as a fair comparison as to the amount of improvement. Furthermore, the two algorithms will be compared against each other in order to prove the best overall algorithm. As you would expect, the higher the percent decrease in the number of collision, compared to the baseline, the better the algorithm is. Also, since APF has already been compared by previous researchers, comparing RIPNA to APF produces a lot of information about the ranking of all algorithms tested using the current research platform.

Figures 6.1 and 6.2 show the results in the percent decrease in the number of collisions over to the baseline (no collision avoidance). The higher percent decrease in the number of collisions, the better the algorithm is. As mentioned in the Figures 6 section above, APF was clearly the best collision avoidance algorithm in previous simulation attempts. In Figures

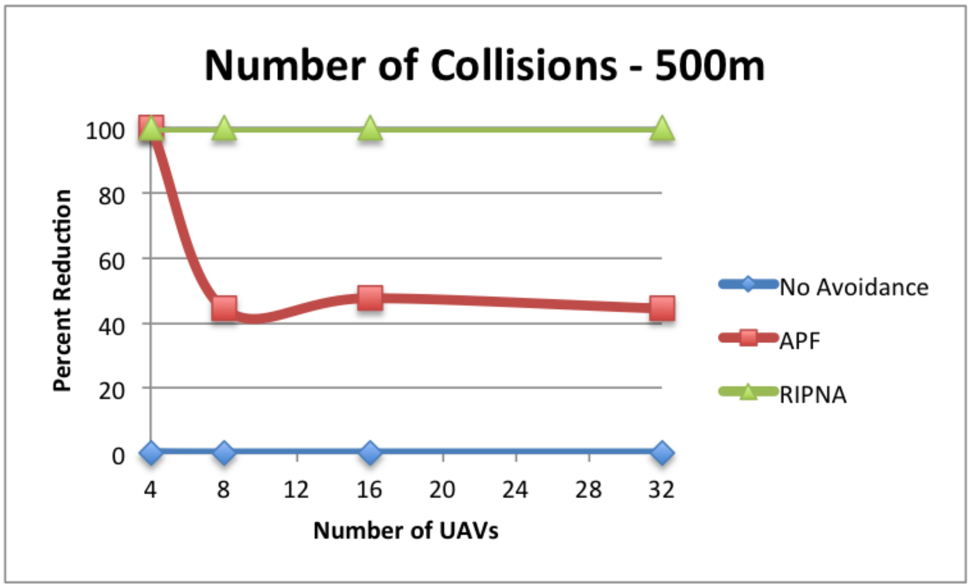


Figure 6.1: The percent decrease in number of collision on a 500m grid

6.1 and 6.2 it is clearly shown that RIPNA improves vastly upon APF. In fact, RIPNA is 125 percent better in the best case for a 500m grid, and 63.63 percent better in the 1000m grid. This alone proves to be a drastic increase, which makes RIPNA the front runner for the best collision avoidance algorithm. With that being said, the other four metrics need to be evaluated before making a final decision.

6.2 Number of Conflicts

A conflict between two aircraft is described as being within a two second distance of each other. For the purpose of this simulation, the speed of each simulated aircraft has been set to a constant of 12 meters/second. Therefore, a conflict between two aircraft occurs when they are 24 meters apart from each other.

Conflicts are a very important metric. Imagine if two aircraft are only two seconds apart from each other. This situation will more than likely end up in a collision, unless a collision avoidance maneuver is already underway. Therefore, decreasing the number of conflicts is a very beneficial way to, in turn, decrease the number of collisions and provide an overall better collision avoidance algorithm.

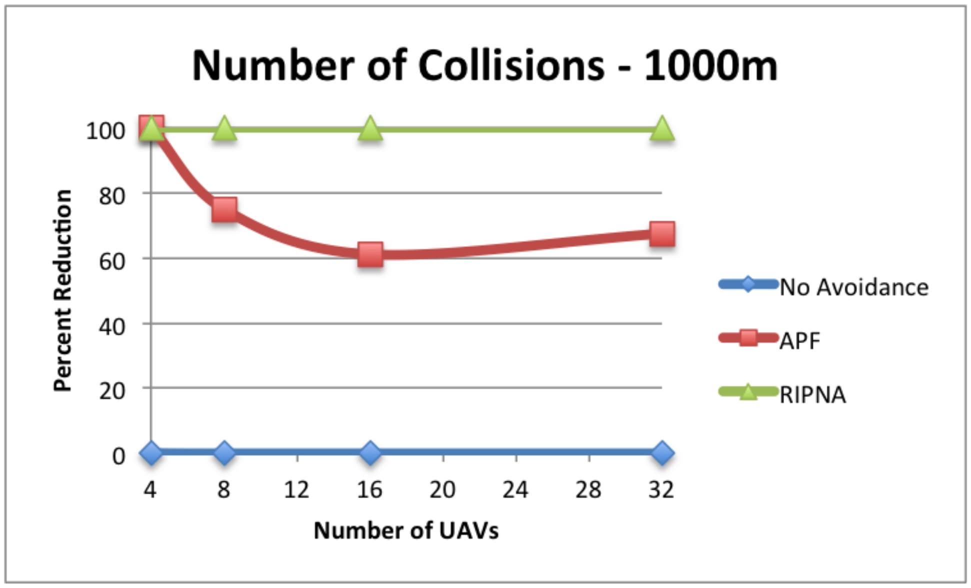


Figure 6.2: The percent decrease in number of collision on a 1000m grid

Since APF has already been evaluated against many other collision avoidance algorithms, comparing APF and RIPNA is a beneficial way to evaluate all algorithms together[11]. Likewise, if RIPNA is better than APF, RIPNA will be the best algorithm tested using this simulation platform. The two figures in this section, Figures 6.3 and 6.4, show the percent decrease in the number of conflicts for a 500m and 1000m grid. As expected, there should be a clear trend of percent decrease, for both algorithms, compared to the baseline of no collision avoidance. This is proven to be true in the two figures. Furthermore, there is a clear trend in Figures 6.3 and 6.4 that shows how much better RIPNA is compared to APF. In fact, in the best case RIPNA is 183.33 percent better than APF in a 500m and 330 percent better in the 1000m grid. This is very drastic improvement over APF. This drastic improvement directly affects the number of collisions, which is why RIPNA is significantly better in both categories. All in all, in the two most significant metrics (collisions and conflicts), RIPNA has significantly beaten APF, meaning RIPNA is a clear improvement over the previously best algorithm.

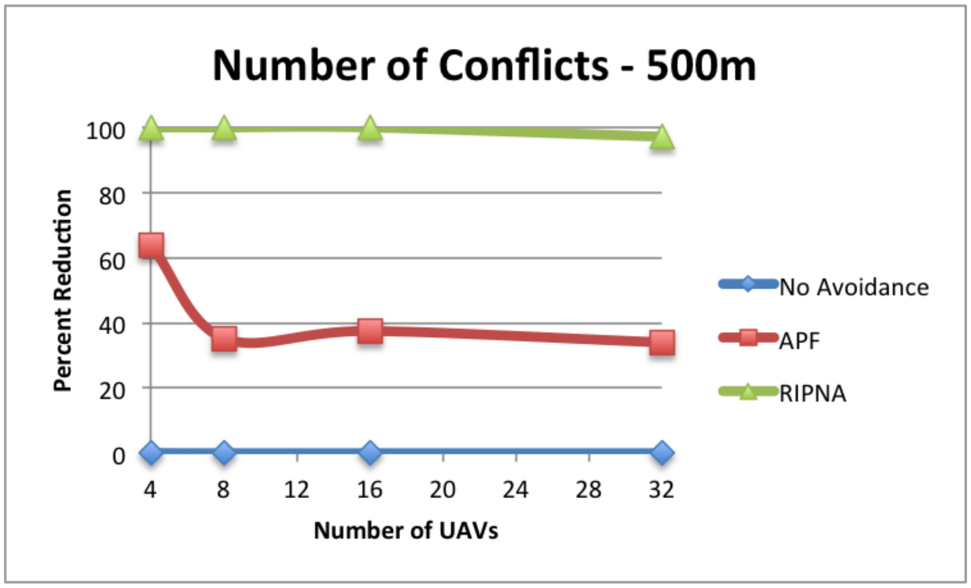


Figure 6.3: The percent decrease in number of conflicts on a 500m grid

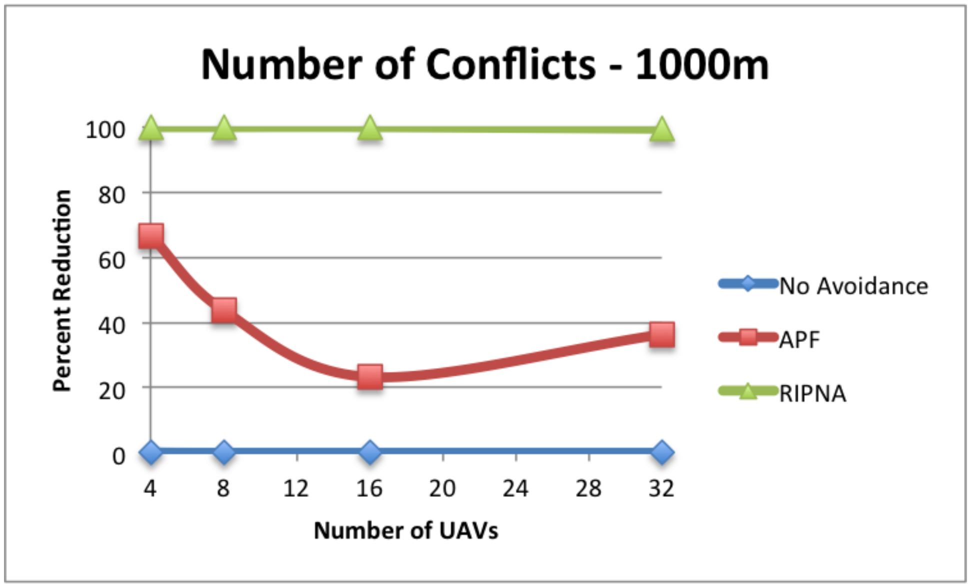


Figure 6.4: The percent decrease in number of collision on a 1000m grid

6.3 Straight-line Deviation

When there isn't any collision avoidance used during a mission, each aircraft attempts to fly directly from waypoint to waypoint. This is known as the straight-line distance, which is the most efficient route between two points. The desired result from any collision avoidance algorithm is to avoid collision while not deviating from the straight-line distance to much. This metric evaluates that deviation. A good algorithm should minimize this deviation, while maintaining safe flight.

Unlike the number of collisions, and number of conflicts, RIPNA is not a clear winner over APF in the straight-line deviation metric. This is due to the nature of APF and RIPNA; APF attempts to use the least means necessary to avoid collisions, while RIPNA is more focused on avoiding the collision at all costs. Therefore, Figures 6.5 and 6.6 do not show a clear trend of which algorithm is better at keeping this deviation to a minimum. With that being said, APF does seem to be better, in most cases, than RIPNA at keeping the straight-line deviation increase to a minimum.

Figures 6.5 and 6.6 show the straight-line deviation compared to the baseline of not using a collision avoidance algorithm. These figures clearly show that both algorithms drastically increase this deviation. This is expected, since both algorithms are introducing avoidance maneuvers into the simulations. In fact, in the worst case, RIPNA has a 157 percent increase in the straight-line deviation on the 500m grid, which is very bad. In this same situation, APF improves upon that by lowering the straight-line deviation to a 103 percent increase over the baseline. Although this seems like a drastic increase, keep in mind that innately any collision avoidance algorithm will have an increase in the straight-line deviation over the baseline. If this weren't the case, the collision avoidance algorithm would be completely unsuccessful at avoiding collisions. Therefore, the attempt of these algorithms should be to minimize this increase, rather than nullify it. Moreover, it should be noted that as the number of UAVs is increased, the straight-line deviation is drastically increased, which is expected. The more planes that are included in the simulation, the harder it is for the

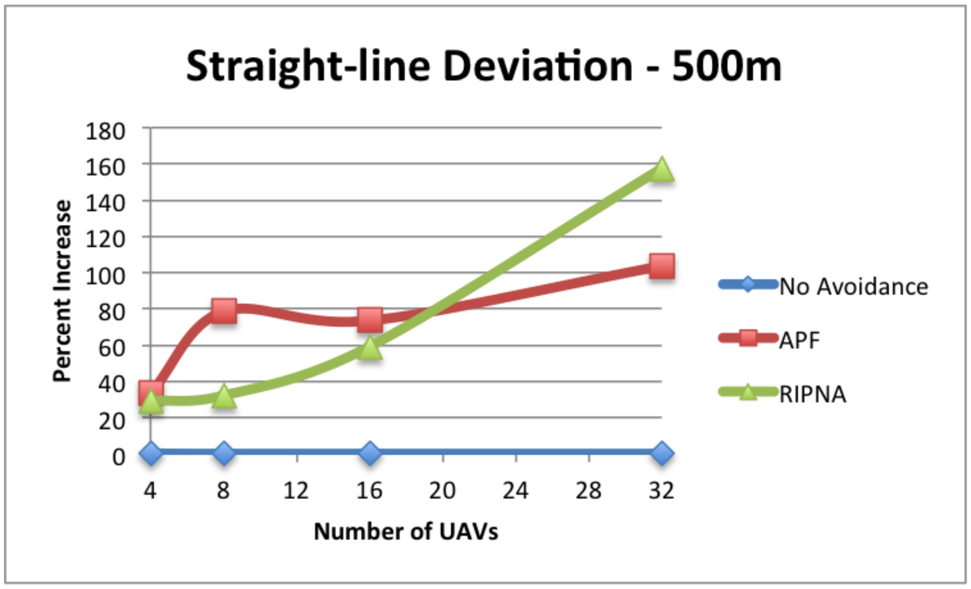


Figure 6.5: The average percent increase in the straight line deviation on a 500m grid

algorithms to make an efficient avoidance maneuver, meaning that more deviation will occur in an effort to keep all aircraft safe. All in all, especially in the 1000m scenarios, both algorithms do a good job at minimizing this deviation.

6.4 Number of Waypoints Achieved

The last metric is the number of waypoint achieved throughout the simulation. This is a significant metric because the overall objective of any simulation is to fly through all of the waypoints. Ideally, all of the waypoints in a mission should be reached. To add some value to this metric, this simulations in this paper are terminated after running for 10 minutes. This provides more significant results, since the more efficient algorithm will reach more waypoints in the given 10 minute time frame. Furthermore, truncating the simulation to 10 minutes allows for a clear analysis of multiple algorithms; The most waypoints achieved the better.

Like the first three metrics presented, RIPNA is a clear winner in improving the number of waypoints achieved. Surprisingly, APF doesn't improve that much over the baseline; The most significant improvement APF had over the baseline is a 70 percent increase, while the

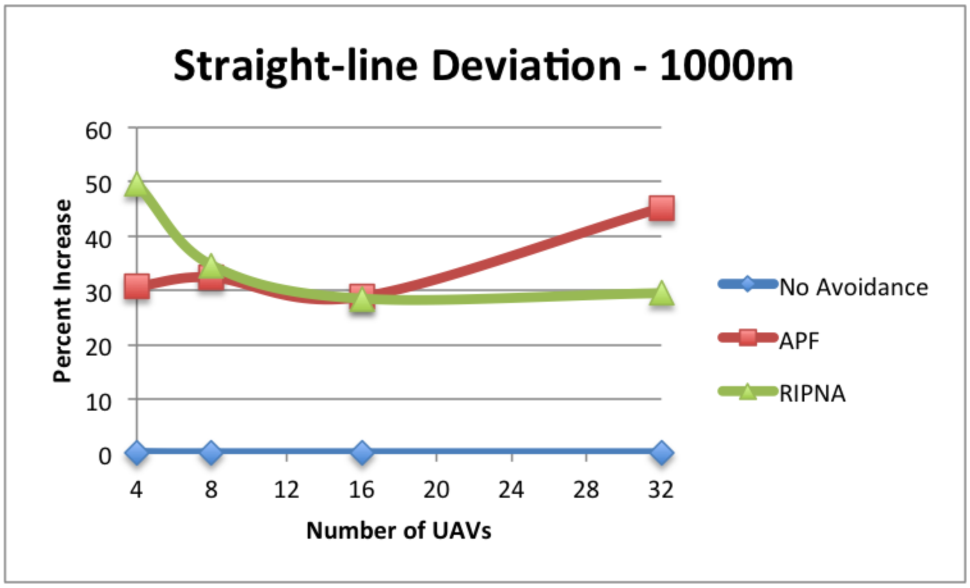


Figure 6.6: The average percent increase in the straight line deviation on a 1000m grid

least improvement for APF is 1.62 percent. This is directly due to the number of collisions that APF allows. If more collisions occur, less number of waypoints will be hit. Therefore, RIPNA really shines because the number of collision are nullified, where APF falls short because it produces some collisions.

Figures 6.7 and 6.7 show that RIPNA is the overall better algorithm in this metric, for both grid sizes. This is expected since RIPNA has beaten APF in both the number of collisions and number of conflicts metric. As shown in Figure 6.7, RIPNA is 127 percent better than APF in the scenario with 32 UAVs. Since RIPNA performs so well in a highly constrained environment, this is expected. All in all, RIPNA is a much better algorithm at increasing the number of waypoints achieved by each aircraft. This is a significant advantage over APF, especially since APF didn't improve that much over the baseline, and RIPNA has a drastic improvement.

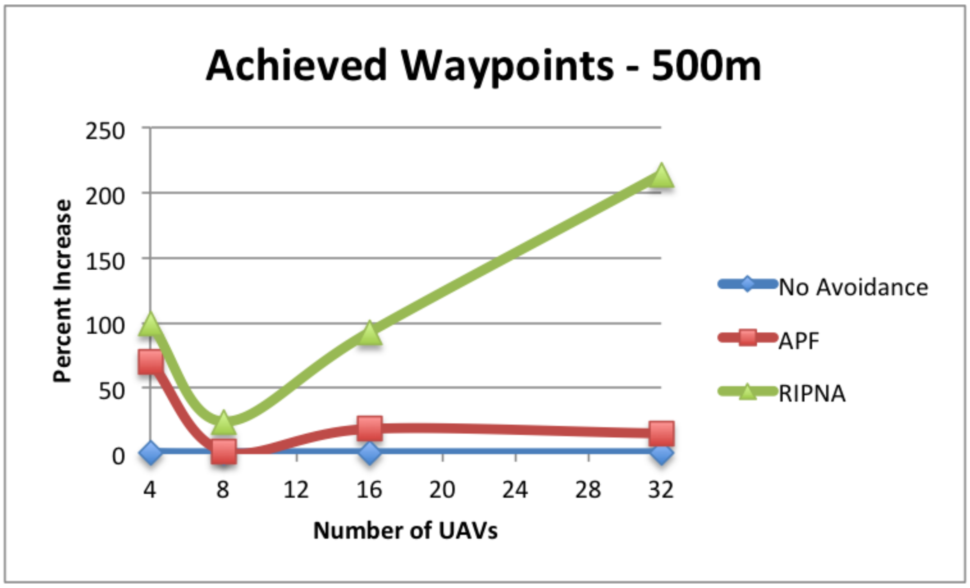


Figure 6.7: The average percent increase in the straight line deviation on a 500m grid

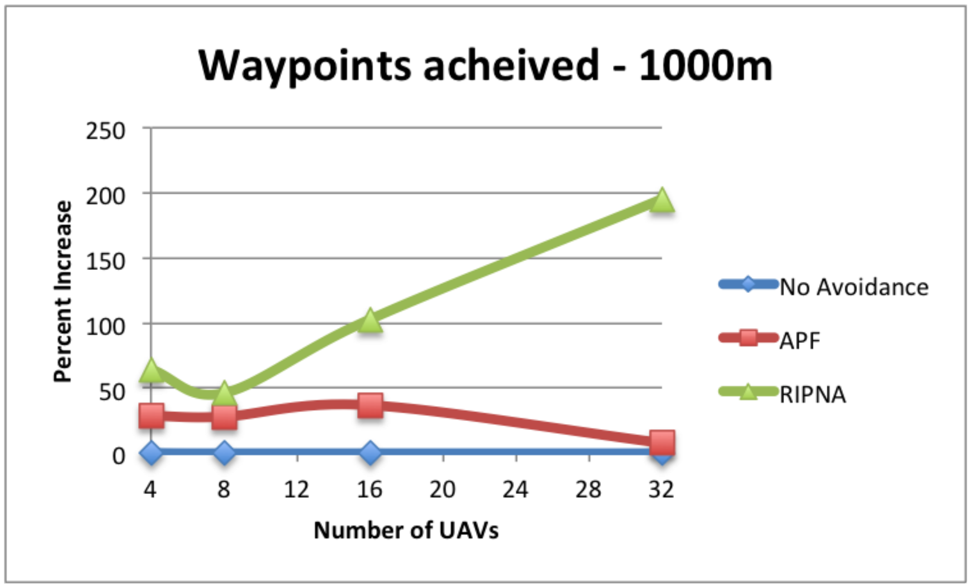


Figure 6.8: The average percent increase in the straight line deviation on a 1000m grid

6.5 Algorithm Performance

6.5.1 APF Performance

While APF wasn't the best performing algorithm overall, it still proved that it's capable of working under the more reasonable simulations. In particular, APF excelled at solving problems with only four aircraft. It had a zero collisions on both the 500m and 1000m grid. Also, it achieved the most waypoints out of any of the collision avoidance algorithms for four aircraft in the 1000m grid scenario. Likewise, APF excelled in the straight-line deviation metric, where it beat all other algorithms for four, eight, and sixteen aircraft, on the the most constrained environment, the 500m grid.

Unfortunately, compared to RIPNA, this implementation of APF struggled to handle the stress test simulations. As described earlier, APF becomes much more heavyweight with each aircraft added to the algorithm. This is due to the nature of APF; APF maps everything in the airspace to particle forces, which then produces a resulting force, which is mapped to a collision avoidance maneuver. When more aircraft are added into the airspace, more particle forces have to be considered. In a tight airspace, this becomes even more complex because there isn't a one-to-one relationship between the magnitude of a force vector and how that correlates to where a collision avoidance waypoint should exist. Moreover, imagine if a resulting force produced a force vector for 10 magnitude. How should a magnitude of 10 map to where a collision avoidance waypoint should be issued? Should the waypoint be placed at 10 times the current speed of the aircraft, or one-tenth of the current speed? Currently, the direction of the resulting for the driving factor in where the waypoint is placed, and the magnitude is negligible. For a more reliable APF implementation, the magnitude should be taken into consideration. This is most noticeable in a very high constrained environment. Particularly, for all metrics, APF is underwhelming for 32 aircraft. For both the 500m grid and the 1000m grid, APF is outperformed for all metrics. In fact, for the 32 aircraft scenarios, APF only had a 7.9 percent improvement in the number of waypoints achieved on the 1000m and a 15.1 percent improvement in the 500m grid.

With that being said, APF does perform well in taking the closest to straight-line distance possible, even if sometimes that may result in a collision. Therefore, APF performs very well in the straight-line deviation metric. Once again, this is due to the implementation of APF, where the objective waypoint a given aircraft is mapped to a very high attractive particle force, and harmful aircraft are mapped to a less-significant repulsive force. This makes APF very greedy, in the sense that at all costs it tries to proceed toward the destination, rather than flying away from the destination, if needed, to avoid all collisions. While this sounds counter-intuitive, it does produce some interesting results, especially in the straight-line deviation metric. Specifically in the 1000m meter grid, for 32 aircraft, APF outperforms the baseline by 45 percent, and even outperforms all other algorithms by 53 percent. Like was said earlier, this is where idea of APF shines the most.

In summary, APF is a very efficient algorithm for manageable problems. The algorithm performed well when only working with four aircraft. The algorithm also performed well with eight aircraft, but didn't maintain a high survival rate. Unfortunately, sixteen and thirty-two aircraft produces lackluster results for this implementation on both field sizes. While this APF implementation works well for a small quantity of aircraft, more work needs to be done on mapping more realistic particle charges to the problem space, and mapping resulting force vectors to more realistic collision avoidance waypoints. With more work, this algorithm has the potential to handle a large number of aircraft safely, and efficiently.

6.5.2 RIPNA Performance

The Reactive Inverse Proportional Navigation Algorithm was 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 keep all aircraft alive, for all scenarios. The second was that RIPNA drastically improved upon the number of waypoints achieved, especially in high traffic scenarios. Finally, RIPNA improved significantly upon the number of conflicts that occurred.

Being the main objective of a collision avoidance algorithm, the survival rate, or the number of collisions is the most important metric, although the number of conflicts a very closely related metric. In both the number of collisions and the number of conflicts RIPNA beat APF drastically. In fact, on both a 500m and 1000m field, RIPNA has a perfect survival rate for all number of aircraft, and near perfect conflict rate for all situations; RIPNA only had conflicts when thirty-two aircraft were involved. APF wasn't even close to achieving this. Furthermore, having a perfect survival rate is the best any algorithm could possibly perform, therefore RIPNA excelled in the first two metrics, number of collisions and number of conflicts.

The second reason RIPNA excelled was it tended to have a higher improvement in waypoints achieved than any other algorithm, especially in the high traffic situations. Overall, this is a very important metric because it evaluates the usefulness of the overall simulation. Imagine if a collision avoidance algorithm made all aircraft fly directly away from a central point of the field, but never actually directed the aircraft back to their desired destination. This algorithm would receive a perfect survival rate, but if zero waypoints were achieved it would be useless. Therefore, this metric evaluates the usefulness of the mission. In low traffic situations RIPNA is lackluster. With that being said, RIPNA still performs optimally in low traffic situations, compared to APF. In high traffic situations is where RIPNA is useful. When thirty-two aircraft are present, on the 500m grid RIPNA outperformed the next best algorithm by 172 percent!

Finally, RIPNA significantly improves upon the number of conflicts that occurred. As you can imagine, since RIPNA performed very well in keeping aircraft alive, the number of conflicts per aircraft also drastically increased. In fact, RIPNA outperformed the next best algorithm by 206 percent. This improvement is directly related to how well RIPNA performs in highly constrained environments.

6.5.3 Conclusion

While each algorithm has strengths and weaknesses, it's equally important to recognize those traits with respect to scenarios. Many of the scenarios presented in this research are extremely dangerous, especially if a ill-suited collision avoidance algorithm is used. From this research, it is apparent that APF is very capable, and even exceptional, at handling up to eight aircraft in a 500m field, and up to sixteen aircraft in a 1000m field, despite the randomness of the scenarios. Furthermore, RIPNA was not limited by the number of aircraft or field size, although with more simulations this threshold could have been found. RIPNA is able to handle up to thirty-two aircraft on both a 500m field and 1000m field, meaning RIPNA successfully handled all scenarios.

After analyzing the results for all algorithms, it is apparent that improvements can still be made. For APF, more experimentation could lead to better overall results, by mapping the particle forces more realistically. By altering the amount of attractive force that a waypoint exerts or the amount of repulsive force an opposing aircraft exerts will drastically affect the performance of APF. Furthermore, how the resulting force APF produces is interpreted is a portion of the APF implementation that also affects the results significantly, and is an area capable of improving upon. For example, if APF produces a resulting force of magnitude 10, and a resulting force of magnitude 100, these forces should lead to different avoidance maneuvers. With the current implementation, the resulting force magnitude isn't considered. Furthermore, RIPNA also has areas for improvement. In some situations RIPNA deviates from the straight-line distance too far. Decreasing this deviation will improve RIPNA. Likewise, RIPNA should be stress-tested in more high-traffic simulations than it was tested in this report. Finally, the best approach could combine both RIPNA and APF into one hybrid algorithm that would utilize the strengths of both, resulting in the best performance, in all metrics.

All in all, for every single metric, the advantages of RIPNA are shown. Specifically, RIPNA is the best performing algorithm for four out of the four metrics: number of collisions, number of conflicts, and number of achieved waypoints. The only metric that RIPNA did not entirely beat other algorithms on was the straight-line deviation metric, although, among the four metrics, this is the least significant metric. Out of all the four metrics, this is the only metric that evaluates how much of a deviation the collision avoidance algorithm uses to avoid collisions, and although this metric is directly related to the algorithm efficiency, avoiding collisions is the main concern of the algorithm. Therefore, the trade-off between avoiding collisions and deviating from the straight-line is necessary. As shown through this paper, RIPNA has proven to be the most efficiency and reliable algorithm presented in this paper, and should be used for furthering research related to multi-aircraft simulations.

Bibliography

[1] Garamone, Jim. ”United States Department of Defense.” http://www.defense.gov/News/NewsArticle.aspx?ID=44164. American Forces Press Service, 16 Apr. 2002. Web. 07 Oct. 2013.

[2] Bone, Elizabeth, and Christopher Bolkcom. ”Unmanned Aerial Vehicles: Background and Issues for Congress.” http://www.fas.org/irp/crs/RL31872.pdf. N.p., 25 Apr. 2003. Web. 7 Oct. 2013.

[3] Ballinger, Mark. ”What's in a Name?” http://www.uasvision.com/2011/08/31/whats-in-a-name/. N.p., 31 Aug. 2011. Web. 7 Oct. 2013.

[4] Kuchar, James, and Ann Drumm. ”The Traffic Alert and Collision Avoidance System.” 16.Nov 2 2007.

[5] Richards, Arthur . ”Aircraft Trajectory Planning With Collision Avoidance Using Mixed Integer Linear Programming.” May 13 2002

[6] Hailong, Pei. Shengxiang, Zhang. Real-time Optimal Trajectory Planning with Terrain Avoidance Using MILP. Oct 14 2013.

[7] Ternullo, Noah. Glickstein, Ira. Robust Algorithm for Real-Time Route Planning. IEEE Transactions on Aerospace and Electronics Systems. vol 36, pg. 869-878, 2000.

[8] Ruchti, Jason. Senkbeil, Robert. Carroll, James. Dickinson, Jared. UAV Collision Avoidance Using Artificial Potential Fields.

[9] George, Joel. Ghose, Debasish. A Reactive Inverse PN Algorithm for Collision Avoidance among Multiple Unmanned Aerial Vehicles.

[10] Watson, Steve. ”Spy Drone Almost Causes Mid Air Collision With Jet Over Denver.” http://www.infowars.com/spy-drone-almost-causes-mid-air-collision-with-jet-over-denver/. N.p., 17 May 2012. Web. 07 Oct. 2013.

[11] Holt, James M. ”Comparison of Aerial Collision Avoidance Algorithms in a Simulated Environment.” http://etd.auburn.edu/etd/bitstream/handle/10415/2976/holtThesis.pdf. N.p., 6 May 2012. Web. 7 Oct. 2013.

[12] Khatib, Oussama. Real-time obstacle avoidance for manipulators and mobile robots. International Journal of Robotics Research, 5:9098, 1986.

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

[14] Liu, D.K.; Wang, D.; Dissanayake, G. A Force Field Method based Multi-Robot Collaboration. In Robotics, Automation and Mechatronics, 2006 IEEE Conference on, pages 16, June 2006.

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

[16] Westman, Eric; Fish, David. An Inverse Proportional Navigation Algorithm for Collision Avoidance among Multiple Unmanned Aerial Vehicles. June 2012.

Appendices

Appendix A

Abbreviations

Federal Aviation Administration (FAA)

Unmanned Aerial Vehicles (UAV)

Unmanned Aircraft System (UAS)

Aerial Terrestrial Testbed for Research in Aerospace, Computing, and maThematics (ATTRACT)

Beacon Collision Avoidance System (BCAS)

Traffic Alert and Collision Avoidance System (TCAS)

Aircraft Collision Avoidance System (ACAS)

Mixed Integer Linear Programming (MILP)

Artificial Potential Fields (APF)

Attraction Constant (γ)

Maximum distance at which one UAVs force can be felt (d_{fmax})

Distance that one UAV can travel in one second (d_{onesec})

Ratio of the collision zone to the size of the potential field (α)

Scale factors for the amount of the ellipse extended out from the UAV (λ_{scalef} and λ_{scaleb})

Angle from the heading of one UAV to the position of another UAV (θ)

Minimum distance a UAVs force field can be felt ($d_{failsafe}$)

The maximum force felt when a UAV is within dfailsafe ($F_{failsafe}$)

Scalar affecting the force in front of a UAV (k_{emitf})

Scalar affecting the force in behind of a UAV (k_{emitb})

The negative charge from a UAV (q_{UAV})

Scalar to check the strength of the repulsive force (γ)

The angle between the UAVs bearing and the repulsive force (ϕ_{rep})

The force felt from an obstacle in front of the UAV (β_{feelb})

The angle between the location of the current UAV and the bearing of another UAV (θ)

The angle between the current bearing of a UAV and the repulsive force acting on another UAV (ϕ_{rep})

The angle between the bearing of a UAV and the attractive force from its goal (ϕ_{attr})

4.5 * the donesec ($d_{priority}$)

Scalar to expand the potential field of prioritized aircraft (p_{mult})

Proportional Navigation (PN)

Inverse Proportional Navigation (IPN)

Reactive Inverse Proportional Navigation Algorithm (RIPNA)

Line Of Sight (LOS)

Zero Effort Miss (ZEM)

Separation Distance (R_{des})

Time-To-Go (t_{go}).

Department of Defense (DOD)

Appendix B

Link to Source Code

Test bed repository: https://github.com/dhj0001/au_uav_pkg

APF Team Documentation: https://sites.google.com/site/auburnreuteam12012/

RIPNA Team Documentation: https://sites.google.com/site/auburnuavreu2012team5/

Appendix C
GCS Tutorial

This appendix section is for new users. This section will give a step-by-step tutorial on how to setup, build, and run the GCS. Also, this section will provide in-depth explanations about different launch settings, how to test and evaluate different collision avoidance algorithms, and how to use the GUI.

C.1 Configuring ROS

In order to run the GCS you will need to setup the Robot Operating System on a machine. This can be done manually in Linux, or it can be setup through a Linux virtual machine.

If you are setting up ROS manually, it is easiest to follow the ROS tutorial at http://wiki.ros.org/ROS/Installation. This guide will walk you through step-by-step on how to get ROS configured for your operating system. NOTE: The current state of the GCS has only been tested on ROS distributions up to ROS-Hydro, although later distribution should be backwards compatible.

Rather than setting up ROS manually, I think it is easier to setup the virtual machine, since ROS comes pre-configured on the virtual machine. In order to do this download VirtualBox from https://www.virtualbox.org/wiki/Downloads. Once you have setup VirtualBox, you can download the Ubuntu-ROS image, and setup your virtual machine. You can download the Ubuntu-ROS image from http://nootrix.com/downloads/#RosVM, and specficially for Hydro, http://nootrix.com/2014/04/virtualized-ros-hydro/. Follow the installation guide, http://nootrix.com/2012/09/virtualizing-ros/, in order to setup the image in VirtualBox.

Another option, which is probability the best option thus far, uses the virtual machine to import a pre-configured appliance that contains the virtual machine with the source code already installed and set up. This will give you the ability to start exactly where I left off, without the hassle of setting everything up. To do this option, you will first need to download VirtualBox, from https://www.virtualbox.org/wiki/Downloads. Now, you can download the pre-configured appliance. I gave Dr. Biaz this appliance the week before I graduated. Setup the appliance by doubleclicking the .ova file. This will automatically setup a virtual machine in the VirtualBox application.

C.2 Downloading the GCS

The GCS source code is located on my github account at https://github.com/dhj0001/au_uav_pkg/. Please clone, or download this repository, and if you plan on working on this code a lot, I would suggest setting up your own repository since I will more than likely not be available to make publish your commits. Once you download the code, place all of these files into your ROS workspace. Now you are ready to start implementing!

NOTE: If you used the last option suggested in the above section, the source code will already exist in the virtual machine, and you can skip this section.

C.3 Building and Launching the GCS

First, navigate to your ROS workspace by opening the command prompt and typing

```
roscd
```

Now, you can build the source code by typing

```
catkin_make
```

If you successfully built the source code you can launch the GCS by typing

```
roslaunch au_uav_ros "launchFile"
```

where, launchFile is one of the files located in /src/au_uav_ros/launch/. The launch file designates which nodes are executed at run time, and also reads any parameters from the command line.

You can specify parameters while executing the roslaunch. An example of this is

```
roslaunch au_uav_ros launch.launch XBeeIO:=true GUI:=false CAAlgorithm:=RIPNA
```

C.3.0.1 Launch File Explanation

There are two launch files currently written: launch.launch and evaluation.launch. These files could be combined into one launch file, but for simplicity sake they haven't been. The launch.launch file is the generic launch file that should be executed the majority of the time. The only time evaluation.launch needs to be executed is when you want to run the evaluator node, which runs diagnostics on the running collision avoidance algorithm; The evaluator node evaluates the running collision avoidance algorithm based on the four metrics described earlier in this paper.

In each launch file, there are three main tags that are used: arg, for commandline arguments, param, for parameters used by ROS nodes, and node, for specifying which nodes to run. The arg tag is used to specify which commandline arguments are expected. The param tag is usually populated by the information provided by the arg, but not always. The param tag specifies a variable that can be used in a ROS node. Finally, the node tag specifies which nodes should be run when the system is launched.

The launch.launch file runs these nodes:

1. Coordinator

2. Simulator

3. Gui Interfacer

4. au_uav_gui - the QT GUI

5. XbeeIn/XbeeOut

The launch.launch file has these arguments:

1. sim_speed - the fight speed at which the simulated planes fly - default: 1
2. sim_freq - the frequency at which simulated planes update - default: 1
3. planPath - a boolean to specify if path planning is wanted - default: false NOTE: this is used in conjunction with centralized, and is current nonoperational
4. centralized - a boolean to specify if the system is running in a centralized or decentralized environment - default: true - NOTE: decentralized is nonoperational at the moment.
5. XBeeIO - a boolean to specify if you are flying UAVs with XBee connections; this is needed for flying real UAVs - default: true
6. GUI - a boolean to specify if you want to launch the GUI or not - default: true
7. CAAlgorithm - a string to specify which collision avoidance algorithm to run - default: "NONE"

All of these arguments are used to either launch specific nodes, or are read in at runtime to be used by a node.

The evaluation.launch file, as said earlier, is used to evaluate the running collision avoidance algorithm based on four metrics. The node that is in-charge of this is the evaluator. In order to automate the evaluation process, more arguments were added so that a driver can be used to run through many simulations.

The evaluation.launch file runs these nodes:

1. Coordinator
2. Simulator

3. Gui Interfacer
4. au_uav_gui - the QT GUI
5. XbeeIn/XbeeOut
6. Evaluator

The evaluation.launch file has these arguments:

1. sim_speed - the fight speed at which the simulated planes fly - default: 1
2. sim_freq - the frequency at which simulated planes update - default: 1
3. planPath - a boolean to specify is path planning is wanted - default: false NOTE: this is used in conjunction with centralized, and is current nonoperational
4. centralized - a boolean to specify if the system is running in a centralized or decentralized environment - default: true - NOTE: decentralized is nonoperational at the moment.
5. XBeeIO - a boolean to specify if you are flying UAVs with XBee connections; this is needed for flying real UAVs - default: true
6. GUI - a boolean to specify if you want to launch the GUI or not - default: true
7. courseFileParam - a string parameter to specify which courseFile to run. The course file must be in the /course/directory. There is no default for this parameter. It must be specified.
8. outputFileParam i a string parameter to specify the name of the output file. The output file will be built and stored in the /score/directory. There is no default for this parameter. It must be specified.
9. CAAlgorithm - a string to specify which collision avoidance algorithm to run - default: "NONE"

In order to automate the evaluation by running multiple course files, you can run the automator.cpp file. This is located in /ExtraTools/directory. This file will parse through the input file that you specify and run all of the course files that exist in that file. I have provided the automatorTest.txt file for an example.

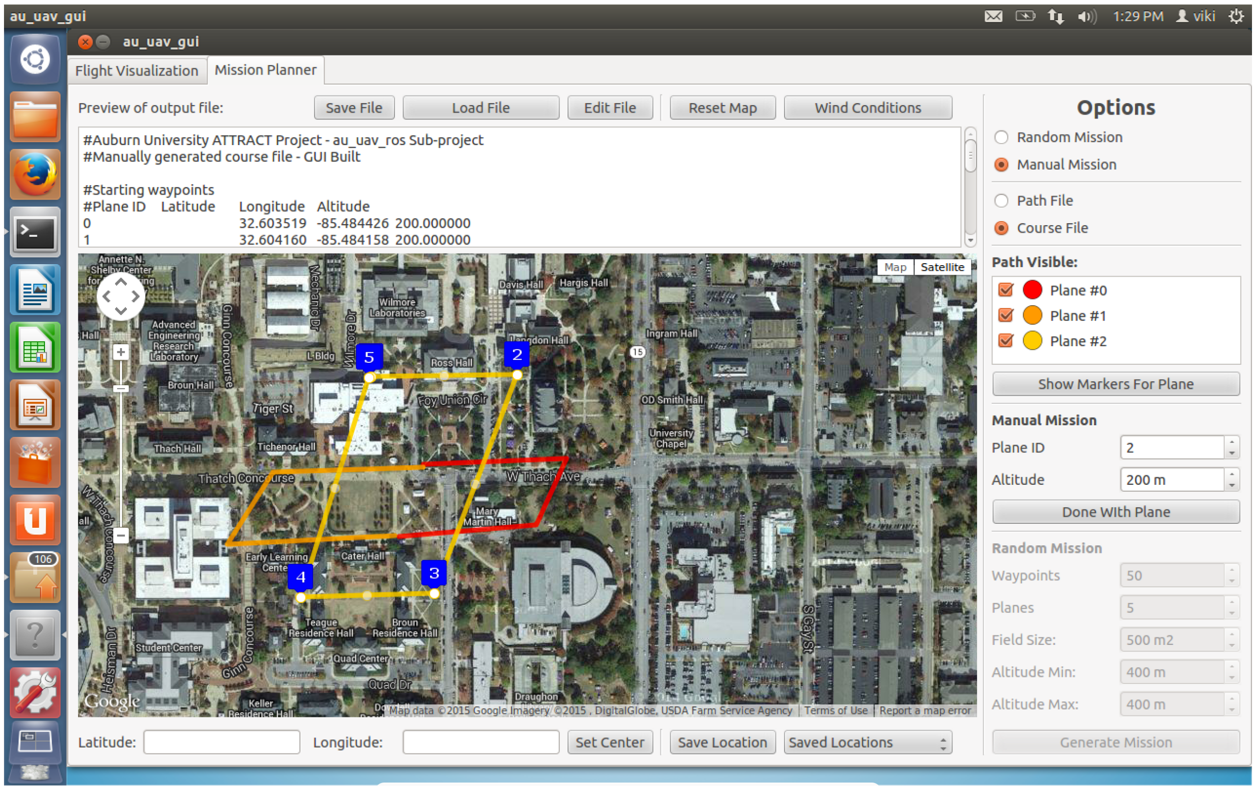
C.4 GUI Explanation

The GUI is a separate system apart from the ROS framework, that is launch by the ROS system. The GUI provides a graphical representation as to what is happening in ROS. The GUI is highly recommended with flying real UAVs, and even when running simulations. In fact, the only time I have suppressed the GUI from launching is when I'm using the automator to script a bunch of tests.

The GUI has two main features: The Flight Visualization, and the Mission Planner.

The Mission Planner tab allows you to plan course files, that can be ran on the Flight Visualization tab. This tab allows the user to point and click to add waypoints to a mission. Also, you can set the height for each waypoint. The waypoints also have a drag and drop option, so that you can move around waypoints with ease. Furthermore, you can edit an existing file by loading the file using the LoadFile button. This tab provides a nice window that shows what the file will look like once it is outputted, so you can check this window to make sure everything is formatted correctly. After you have completed the mission planning, click on the SaveFile button to output the file. Refer to Figure C.1 to understand all of the features.

The Flight Visualization tab provides a visual representation of the current mission being executed in ROS. As you can see from Figure C.2, the active UAVs are depicted by airplane icons. You can see the current telemetry information per UAV by selecting the UAV with the drop-down box on the upper-right. This is shown in Figure C.2. Furthermore, there are little additives that can help visualize the current mission, like enable/disable flight paths, enable/disable waypoints, auto center the map to the current UAVs, auto zoom the



```
#Auburn University ATTRACT Project - au_uav_ros Sub-project
#Manually generated course file - GUI Built

#Starting waypoints
#Plane ID   Latitude      Longitude   Altitude
0                         32.603519  -85.484426  200.000000
1                         32.604160  -85.484158  200.000000
```

Options

- Random Mission
- [✓ Manual Mission]
- Path File
- [✓ Course File]

Path Visible:

- [✓] 🔴 Plane #0
- [✓] 🟠 Plane #1
- [✓] 🟡 Plane #2

Manual Mission

Plane ID	2
Altitude	200 m

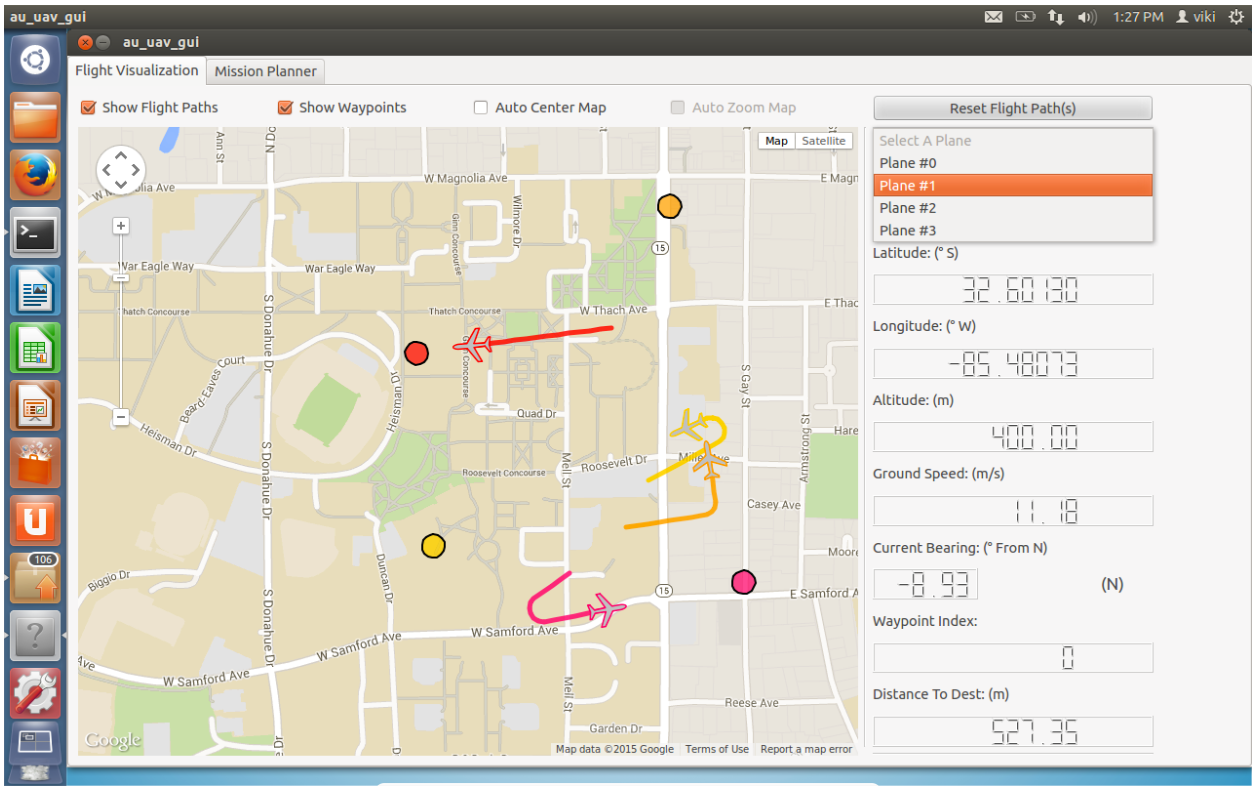
Random Mission

Waypoints	50
Planes	5
Field Size:	500 m2
Altitude Min:	400 m
Altitude Max:	400 m

Figure C.1: The Mission Planner tab on the GUI

map to the current UAVs. These four features can help you while running a simulation so that you are not worrying about changing the resolution of the screen to visualize all of the UAVs.

In order to launch a mission, you need to move your cursor to the menu-bar at the top of the screen and select *File* → *LoadMission*. This will prompt you with a file selection GUI where you can select a course file to launch. This is shown in Figure C.3. All in all, the best way to get familiar with the GUI is to use it.



au_uav_gui

☑ Show Flight Paths ☑ Show Waypoints ☐ Auto Center Map ☐ Auto Zoom Map

Latitude: (° S) 32.60130

Longitude: (° W) -85.48073

Altitude: (m) 400.00

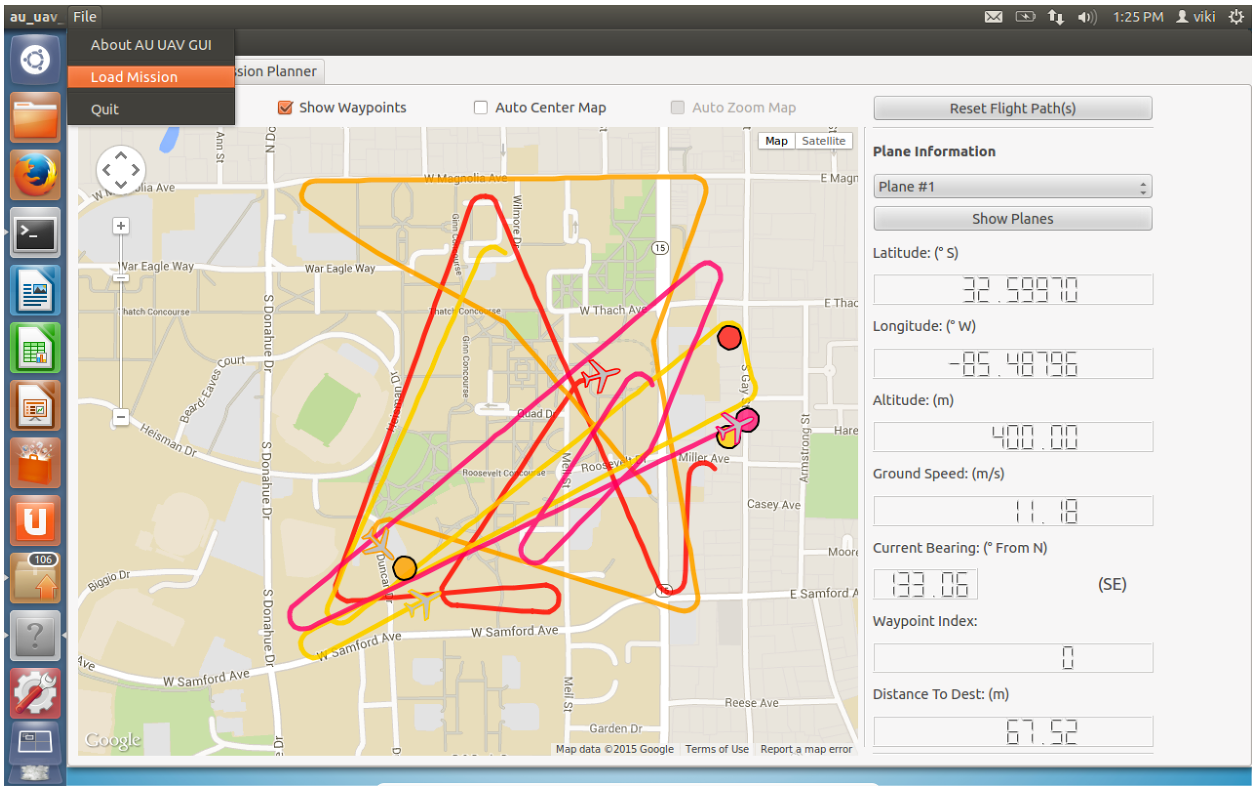
Ground Speed: (m/s) 11.18

Current Bearing: (° From N) -8.93 (N)

Waypoint Index: 0

Distance To Dest: (m) 527.35

Figure C.2: The Flight Visualization tab on the GUI



[✓ Show Waypoints] Auto Center Map Auto Zoom Map

Plane Information

Plane #1

Latitude: (° S) 32.59970

Longitude: (° W) -85.48796

Altitude: (m) 400.00

Ground Speed: (m/s) 11.18

Current Bearing: (° From N) 133.06 (SE)

Waypoint Index: 0

Distance To Dest: (m) 67.52

Figure C.3: The Flight Visualization tab on the GUI