Decentralized Collision Avoidance Framework for Unmanned Aerial Vehicles

Andrew Cunningham
Rensselaer Polytechnic Institute
Troy, New York
cunnia3@rpi.edu

Victoria Wu
University of Missouri - Kansas City
Kansas City, Missouri
vcw7c5@mail.umkc.edu

Saad Biaz
Auburn University
Auburn, Alabama
biazsaa@auburn.edu

David Jones
Auburn University
Auburn, Alabama
dhj0001@auburn.edu

***Abstract*—Collision avoidance for unmanned aerial vehicles (UAVs) is a vital issue when using multiple drones. Decentralized collision avoidance algorithms allow for greater range and independence when compared to a centralized approach; however, it also shifts the workload of detecting and resolving conflicts from a more powerful ground station onto individual agents, requiring both a computationally efficient algorithm and supporting framework. This paper presents a lightweight and modular decentralized collision avoidance framework for implementation on a low cost microcomputer. An existing distributed artificial potential field algorithm is modified for use with this framework. Finally, the effectiveness of this framework is verified in simulation and tested using real UAVs.**

I. Introduction

As Unmanned Aerial Vehicles (UAVs) become more widely utilized in both civilian and military endeavours, the need for an effective collision and avoidance resolution algorithm becomes more pressing. Existing collision avoidance algorithms can be classified into two different types - centralized and decentralized. In a centralized collision avoidance algorithm, a central control unit will handle all collision avoidance computations. While centralized algorithms can find globally optimal solutions, they are often impractical because they require the presence of a ground station, which may limit the range that UAVs can travel. Decentralized algorithms provide an alternative in which all collision avoidance computations are run locally on each UAV.

The primary disadvantage associated with decentralized collision avoidance algorithms is that they usually produce suboptimal solutions. This results from the fact that decentralized algorithms typically operate on incomplete information and only avoid collisions locally. However, this situation reflects reality because UAVs will rarely have perfect information about their environment and neighbors. Decentralized collision avoidance algorithms also offer several advantages. One key advantage that decentralized algorithms offer is that they are robust, so no single failure will disable the entire system. In addition, many decentralized algorithms are scalable due to the fact that the local interactions do not change much as more UAVs are added.

Decentralized collision avoidance algorithms are well suited for the inherent mobility and range of UAVs, but require a computationally efficient algorithm and surrounding framework. In this paper, we present a lightweight and modular distributed collision avoidance framework for use in UAVs. In section II, we briefly define our problem statement, as well as the constraints imposed. In section III, we give a general overview of existing distributed collision avoidance algorithms that have similar constraints to our problem statement. In section IV, we describe our implementation of a modified existing algorithm as well as the architecture of our framework. In section V, we discuss the results of testing our framework with and without our modified algorithm. We conclude in section VI, and address future work in VII.

II. Problem Statement

Our main intent is to ultimately develop a distributed, lightweight collision resolution framework that is 1. ready for use with a low cost processor, and 2. able to modularly support various distributed collision resolution algorithms.

Our framework operates in conjunction with an existing ground control station and simulator developed in [1]. The ground control station is responsible for sending desired goal waypoints to simulated and real UAVs while receiving telemetry updates to track progress. The simulator is responsible for generating telemetry messages for simulated planes, and maintaining the state of all simulated planes.

Certain requirements are set to account for real world conditions. One of the main requirements is that the framework must be lightweight. The size of the drone used limits the weight of the payload, which in turn limits the type and power of processor that can be used. Because our aim is to implement a distributed framework, another limitation is that all processing for collision avoidance must be done locally. These two limitations combined require that any framework developed, as well as any algorithm used, must be as lightweight as possible to ensure effective real time performance.

In order to simplify the problem, we assume cooperative agents, as defined in [2]. Cooperative agents share flight information either passively by broadcasting, or actively by responding to a query. Non cooperative agents, on the other hand, do not actively share flight information and are considered obstacles. We constrain our problem to detecting and resolving conflicts with solely cooperative agents; in our case, UAVs passively broadcast their telemetry information consisting of speed, bearing, and current and destination coordinates to other UAVs. To further simplify our problem, certain constraints are placed on the UAVs' performance. The UAVs are constrained to a fixed speed and altitude, limiting the UAVs' movement to turning right or left.

In order to test this framework, we modify and decentralize an existing APF algorithm implemented in [3].

III. Literature Review

A framework for categorizing collision avoidance systems is presented in [4]. In this model, a collision avoidance system is conceptually broken down into different modules (Figure 1). A collision avoidance algorithm can be designed by including the conflict detection and escape trajectory selection, or conflict resolution, modules. In this framework, a collision is defined as an instance in which two or more UAVs violate a minimum separation distance between each other, and a conflict occurs when two or more UAVs will enter a collision sometime in the future. In order to avoid collisions, it is necessary to first detect the corresponding conflict. Therefore, conflict detection is a prerequisite of conflict resolution.

A. Conflict detection and awareness

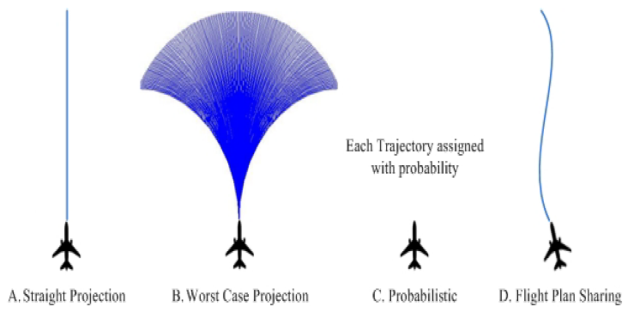

Fig. 2. The four main types of state projection [4]

1) State Projection: State projection refers to the method used to predict the future position of a UAV to detect conflicts. Figure 2 provides a visual representation of each type of state projection and [4] provides a description of each:

Nominal or straight projection traces the UAV's current trajectory and does not account for changing flight plans. A conflict is detected if a UAV will collide with another UAV given that they both continue on their current headings. However, straight projection is not reliable in cases where flight plans change frequently or for long projection times due to the fact that changed headings render previous predictions incorrect.

Worst-case projection traces all possible routes that a UAV could take and detects a conflict if any of these flight paths intersects with the flight path of another UAV. Projection distance must be limited in this approach to prevent oversensitivity in conflict detection and to minimize computational time devoted to prediction.

Probabilistic projection is similar to worst-case projection in that it considers multiple variations on the current trajectory, but it adds the additional layer of considering the likelihood that the specific variation will occur.

Flight plan sharing allows for a more direct detection of conflicts as it involves having multiple UAVs sharing portions of their planned trajectories with other nearby UAVs. This method allows for accurate conflict detection, but presents the difficulty of actually transmitting these plans.

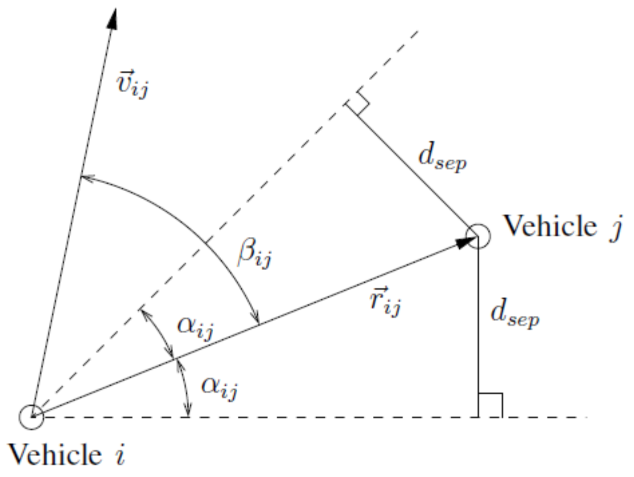

Fig. 3. The collision cone. A conflict is detected when velocity vector $\vec{v_{ij}}$ lies between the dotted lines [5]

2) Collision Cone: The collision cone is a pairwise conflict detection system in which a system of two moving agents is converted into a system with a single agent and a static obstacle [6]. Collision cones can be used to determine if a pair of agents are in conflict, assuming that the agents will continue on their current trajectories.

A conflict is detected if the relative velocity vector lies within the collision cone. Mathematically, a minimum seperation distance, d_{sep}, will be violated sometime in the future if $\beta < \alpha$ where:

$$\beta = \angle\vec{v} - \angle\vec{r},$$
$$\alpha = \arcsin\left(\frac{d_{sep}}{||\vec{r}||}\right)$$
$$\vec{r} = \text{relative position vector}$$
$$\vec{v} = \text{relative velocity vector}$$

B. Conflict Resolution

1) Predefined: In a predefined or rule based collision avoidance algorithm, certain rules define how a plane should maneuver given a specific situation. One such implementation is detailed in [7]. Potential collisions are categorized according to the angle of the collision as seen in Figure 4; depending on the type of collision, each plane will execute a predefined escape maneuver. This decision is executed independently, with each plane detecting the collision from its point of view.

This approach is relatively simple and easy to implement, taking up only as much processing power is needed to determine the type of collision. However, this approach is inherently static, and thus may not guarantee collision avoidance for special cases. In addition, because the maneuvers are arbitrarily predefined with no room for modifications, this approach may

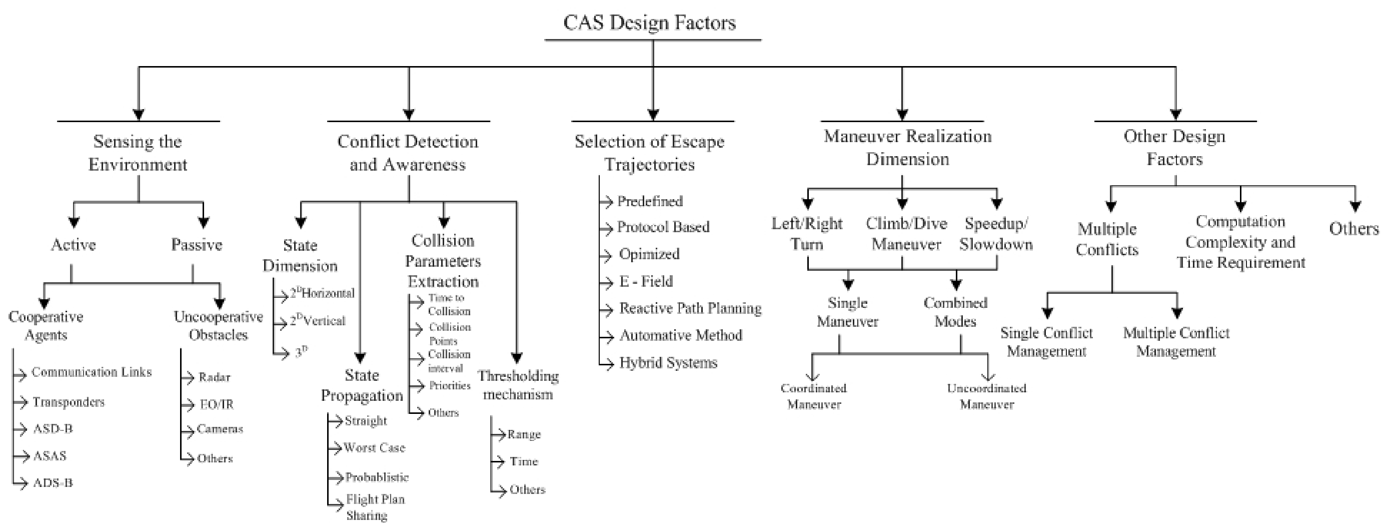

Fig. 1. Illustrates the main CAS design factors divisions with its subdivisions [4]

result in an inefficient escape maneuver when a more efficient one is available.

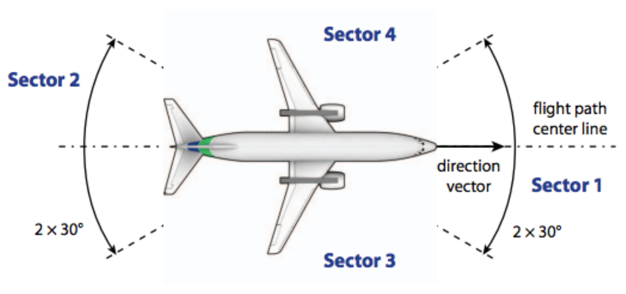

Fig. 4. Categorizing collisions based on angle. [7]

2) Protocol based: A negotiation based approach uses a predefined protocol to compromise and negotiate to achieve a mutually beneficial collision avoidance path. This approach assumes cooperative agents. The goal of such an approach is to find conflict-free flight paths for agents that are mutually beneficial.

One implementation of a protocol based collision avoidance algorithm is discussed in [8]. The algorithm consists of four main steps - detection, negotiation set generation, path optimization, and execution. For the purposes of explaining the steps, a pair of conflicting agents is assumed.

Detection - Based on the flight plan data received from other agents, an agent can determine if a conflict is about to occur. Once a conflict is detected, the process of generating a negotiation set begins.

Negotiation Set Generation - Each agent generates a set of possible maneuvers it could take. In the implementation described in [8], the author suggests six possible alternate trajectories - left/right turn, climb/descend, and speed up/slow down. Each possible trajectory is then marked with a utility value representing the desirability of the trajectory to the agent. The utility of a trajectory is determined by a utility function, defined as the difference between the cost of the trajectory and the cost of a fallback trajectory. Figure 5 lists a number of possible cost functions [9]. The cost function encodes the preferences of each agent, and is taken into account during negotiations. The agent marks each possible trajectory with its calculated utility value and sends them over to the other agent.

$$\begin{aligned}
D &= \text{total distance traveled}\\
\Delta A &= \text{total altitude changes during flight plan}\\
\Delta H &= \text{total heading changes during flight plan}\\
Cost &= D\\
Cost &= D + \Delta A\\
Cost &= D + \Delta H\\
Cost &= D + \Delta A + \Delta H
\end{aligned}$$

Fig. 5. Cost functions for generating utility values. [9]

Once an agent receives a set of possible paths from a conflicting agent, it will pair them up with its own set to make a new set of all possible pairs of paths both agents could take. The resulting set is the preliminary negotiation set, and is made of up of deals, or pairs of possible paths. All deals in the preliminary negotiation set are then validated, with those generating further conflicts removed; the remaining pairs of non conflict generating paths make up the negotiation set.

After generating the negotiation set, the next step is optimization. The best pair of paths must be found, taking into account the utility values associated with each deal. One possible way to find the best deal is to use the monotonic concession protocol (MCP), a incremental bargaining process described in [9] and [10]. If no deal can be found, then a fallback deal must be executed. One option for the fallback deal is assuming the worst case scenario and finding a guaranteed safe trajectory no matter what the other agent does.

The number of agents involved in the negotiation affects how and when these four steps are executed. For pairs of conflicting agents, each agent can either execute all four steps independently, or arbitrarily assign a master and slave to prevent repetition of work (Figure 6). For conflicts involving more than two agents, the same pair-wise algorithm can be applied iteratively, or the master slave approach can be expanded to replace the master with a coordinator responsible for finding collision free paths for a party of planes.

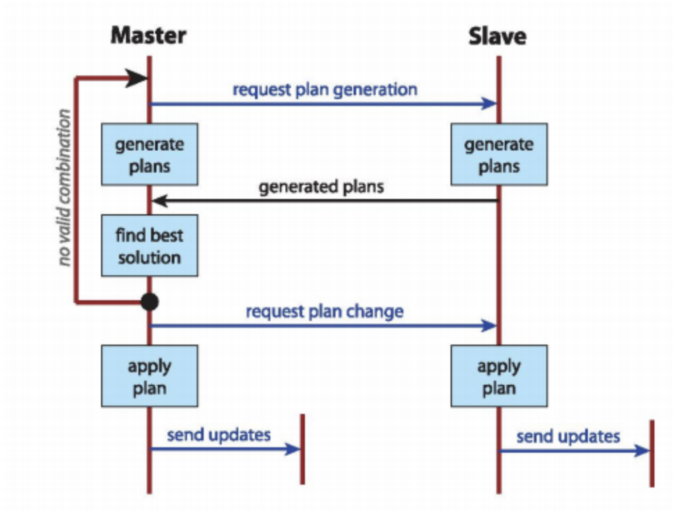

Fig. 6. Protocol based negotiations. [8]

This approach's main benefit is that it negotiates a mutually beneficial path, and the measure of the benefit can be set by each agent. Each agent is able to inject its own preferences into the negotiating process. As a result, this approach can produce more efficient results and is more adaptable than a static, rule based method or a worst case method. However, the chattiness of the protocol requires more bandwidth for communications, and the algorithm itself requires more processing power depending on the number of agents involved.

3) Optimized: The optimization approach to selecting escape trajectories involves formulating the problem in terms of constraints and cost functions. Typically, the constraints are coupled with kinematic models of the planes and the cost functions are derived from a variety of metrics. An escape trajectory can then be found by minimizing the cost function subject to the aforementioned constraints utilizing any optimization technique. Commonly used optimization techniques for this purpose include genetic algorithms, expert systems and fuzzy control.

Genetic algorithms search the solution space of possible paths using techniques inspired by evolution. Genetic algorithms generally follow a set process, described as follows: The first step in most genetic algorithms is to initialize a population, or a random set of initial solutions. The algorithm then enters a loop in which it will iteratively mutate the most successful members of the population and continue to do so until a suitably fit solution is found [3].

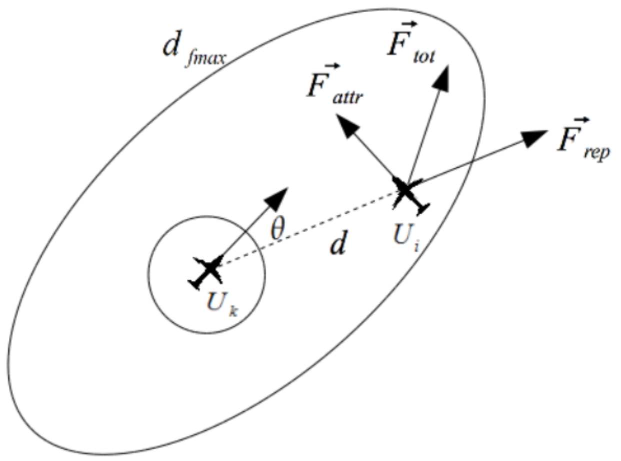

Fig. 7. Calculating virtual forces acting on a UAV [13]

Expert Systems make use of rule bases to decide how to categorize and resolve conflicts. However, these systems require a large number of rules to recognize all type of conflicts and the resolutions to the types of conflict. Once the rule base is established, its use of rules makes it relatively easy to understand. [11]

Fuzzy control systems are modelled on fuzzy logic and follow a certain process. In the first step, fuzzy values are created from numeric or crisp input. The fuzzification of crisp input entails assigning descriptive tags to inputs that meet certain requirements. For example, an altitude of 500-1000 meters could be classified as high. A fuzzy control system can then apply a fuzzy rule base to this input to create a fuzzy output. Finally, this output can be defuzzified to produce a flight plan. [12]

4) E Field: The E-Field or APF (artificial potential field) approach to trajectory calculation involves modelling UAVs and their waypoints as charged particles. UAVs are assigned the same charge so that they repel each other and waypoints are assigned an opposite charge so that their corresponding UAVs are attracted to them. UAVs can then implicitly plan paths by summing attractive and repulsive forces to create the next desired heading.

Several factors need to be considered when designing an APF system. One consideration is field shape. While in reality, charged particles can affect other charged particles over any distance, there is little reason to implement such a field for UAVs in the context of collision avoidance. If the APFs were to extend to a great distance, the contribution of force to the affected UAV would be negligible and would only use more computation time. Therefore, it is often the case that limit functions are created to define a finite field shape around a UAV. Another design consideration that should be taken into account is the field function. Once a UAV enters another UAV's APF, the field function determines the magnitude (and possibly direction) of the repulsive force [14].

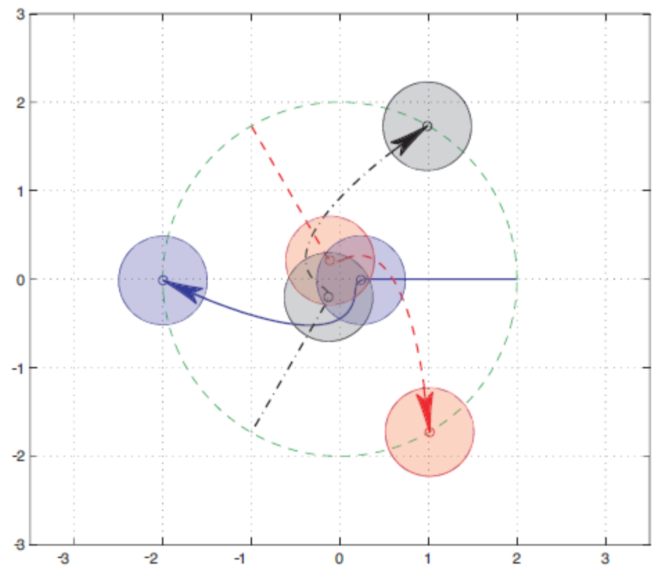

Fig. 8. Gyroscopic forces allow three agents to spin free of their initial configuration and reach their antipodal destinations [17]

There are several issues that arise when APFs are used for collision avoidance on unicycle UAVs. If only APFs are utilized, then there is no method for resolving a head on collision between two UAVs with waypoints located behind each other. This results from the fact that there exists no force in this configuration that would cause the UAVs to turn away from each other. Another issue is looping. Looping occurs when a UAV circles its goal waypoint and is unable to reach it due to turning constraints. In a pure APF approach, the attractive force of a waypoint can provide the centripetal force needed to initiate looping. It is possible to resolve both of these issues by explicitly identifying the situation and then applying the appropriate maneuver(s) to resolve the issue [13].

APFs are a popular research topic and have been utilized in many different ways to implement collision avoidance. [13], [15] and [16] all make use of APFs to govern swarming behavior. Swarming, in this context, refers to the grouping of UAVs in a geometric fashion in response to a specific situation. Specifically, [13] made use of flocking behaviours to limit competition between UAVs that had similar destinations. [17] presents gyroscopic forces as another extension to APFs for collision avoidance. In [17], repulsive forces are replaced with a gyroscopic force that activates whenever two agents come within a certain distance of each other.

IV. Implementation

A. APF Algorithm

A modified APF algorithm was chosen to implement decentralized collision avoidance. In this setup, each UAV is subject to attractive and repulsive forces which lead the UAV to its desired waypoint while avoiding collisions. The attractive force was modeled to have a constant magnitude with a direction that is toward the planes goal waypoint. Repulsive forces, on the other hand, were calculated using a statically assigned field shape and field function on each plane. Statically assigned field shapes and functions enable each plane to calculate virtual forces using only the location and bearing of other planes. The field shape was defined as an ovoid and the field function was defined using bivariate normal functions where these functions and parameters were developed and optimized by [3]. A modular approach to APF assignment was taken, allowing users to define and make use of different field shapes and/or field functions. The correctness of the algorithm was then verified through simulation before it was ported to the hardware framework.

B. Hardware

The hardware used in this study consisted of the Bixler hobby plane and several control/communication devices used to automate its flight. At the center of the control system is the ArduPilot, which enables stable flight between designated waypoints. These waypoints are generated by our algorithm on a Raspberry Pi, which is a low cost microcomputer. Finally, we include XBee modules to enable wireless communication between UAVs and ground stations. We connect the Raspberry Pi, ArduPilot and XBee together using USB cables and communicate between the systems using the mavlink protocol. Mavlink allows for sending and receiving c-structs over serial communication lines. See [18] for a more detailed overview of the mavlink protocol and the custom messages used in our system.

C. Architecture

The decentralized collision avoidance framework developed in this work was implemented using ROS, or Robot Operating System. In ROS, processes are organized as nodes, and ROS provides methods for communication between nodes. Our decentralized collision avoidance framework makes use of three nodes: the ardu, xbee and mover nodes. The ardu and xbee nodes are responsible for handling communications between the ArduPilot and XBee devices respectively, while the mover node is responsible for forwarding waypoints to the ArduPilot and running the collision avoidance algorithm.

These nodes communicate with each other by making use of topics which ROS provides access to. Topics have subscribers and publishers; subscribers read data from the topic and publishers post data to the topic. Figure 9 provides a graphical representation of how the nodes communicate with each other using topics. The four topics used for communicating between nodes are the my_mav_telem, gcs_commands, all_telemetry and ca_commands topics.

1) Ardu: The ardu node is responsible for handling communication between the Raspberry Pi and the ArduPilot system. This entails receiving GPS information from the ArduPilot and forwarding waypoint commands to the ArduPilot. The ardu node accomplishes this by simultaneously listening to the ArduPilot's serial line and waiting for a message to be posted to ca_commands. When the ardu node receives a telemetry update from the ArduPilot, it will publish this information to

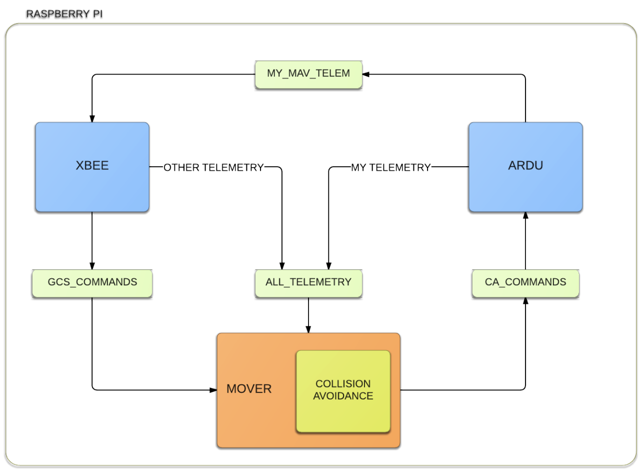

Fig. 9. Our framework architecture

both the all_telemetry topic (which is used by the mover node) and the my_mav_telemetry topic (which is used to send this telemetry to other UAVs).

While the ardu node is listening for a telemetry update from the ArduPilot, it will also be waiting for a command to be published on the ca_commands topic. If a command is published on this topic, the ardu node will take the posted information, pack it into a mavlink message and then send it to the ArduPilot system using serial communication. The ArduPilot will then set the new destination of the UAV to reflect the command.

2) Xbee: The xbee node is responsible for handling communication between the Raspberry Pi and the XBee module. The XBee module communicates with other XBee modules by either receiving messages from a ground station and other UAVs or by sending out telemetry information to other systems. Much like the ardu node, the xbee node simultaneously listens to the XBee module for a message and writes to the XBee once a message is posted to the my_mav_telemetry topic.

The xbee node can currently read and decode two types of mavlink messages: a telemetry update and a command. If a telemetry update is read from the XBee module, the message is decoded and posted to the all_telemetry topic (which is used by the mover node). If a command is read from the XBee module, it is decoded and posted to the gcs_commands topic (which is also used by the mover node).

In addition to reading from the XBee module, the xbee node also waits for a message to be posted to the my_mav_telem topic. Once a message is posted to this topic, the xbee node will take the information, pack it to a mavlink message and perform a serial write to the XBee so that the telemetry information is sent out by the XBee module.

3) Mover: The mover node is responsible for generating waypoints to send to the ArduPilot. This node operates by taking in telemetry messages and ground station control commands, then periodically forwarding waypoints to ca_commands so that they can be sent to the ArduPilot. Input to this node is taken from either the gcs_commands topic or the all_telemetry topic. Messages from the gcs_commands topic specify a goal waypoint that the UAV should eventually aim to reach while running collision avoidance. Messages from the all_telemetry topic, on the other hand, contain GPS updates from all UAVs (including the plane running this framework). Depending on the state that the mover node is in (see section IV-D1), the forwarded waypoint will be either a collision avoidance waypoint or the goal waypoint. The waypoint is forwarded to the ardu node through the ca_commands topic at a preset rate so that the ArduPilot is not flooded with commands.

4) Collision Avoidance: One of the requirements of our system is that it must be modular, such that users can implement their own distributed collision avoidance algorithms within our framework. This was accomplished by encapsulating the collision avoidance logic in the Collision Avoidance class (CA) inside the mover node. Every time a telemetry update is received by mover, the avoid function in CA containing the user specified collision avoidance algorithm is executed. Using the incoming telemetry information, the avoid function generates a collision avoidance waypoint which is then forwarded to the ArduPilot by the mover node.

Encapsulating the collision avoidance logic in its own class provides two main benefits - users can implement their collision avoidance algorithm and any other needed variables in a self contained manner, and different collision avoidance algorithms can be swapped easily.

D. Ease of Testing

For ease of testing, we introduce two features implemented in our framework: state based command forwarding and a pseudo ground control station.

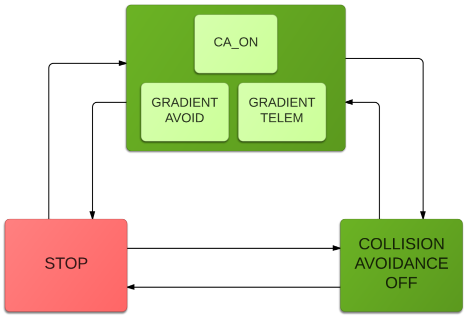

Fig. 10. State based command forwarding

1) State Based Command Forwarding: The mover node operates in three main states that dictate what type of command waypoint is sent to the ArduPilot, illustrated in Figure 10. In the stop state (ST_RED), no commands are published by mover and sent to the ArduPilot. This state is in place to ensure manual remote control mode of the ArduPilot will not be overridden by any unnecessary commands from mover. In the collision avoidance off (CA_OFF) state, the mover node simply forwards goal waypoints to the ArduPilot, without using any collision avoidance. In this mode, the behavior of the plane is the same as if there were only an ArduPilot receiving commands from the ground control station, and no Raspberry Pi in the middle. In the collision avoidance on state (CA_ON), the mover node forwards collision avoidance waypoints generated by the CA algorithm.

Within the CA_ON state, there are two further substates used to simplify testing. The tests substitute the goal waypoint instead of a generated collision avoidance waypoint at different levels, or gradients, of the telemetry callback. The first is the gradient telemetry state (GRADIENT_TELEM); when a telemetry message is received, the goal waypoint is added to the collision avoidance queue instead of a collision avoidance waypoint. The second state is the gradient avoid state(GRADIENT_AVOID); it is similar to the original CA_ON state in that it calls CA's avoid function, but the goal waypoint is substituted from within CA's avoid function. The purpose behind these substates is to allow for further testing of the core framework separate from the collision avoidance algorithm.

State changes are triggered when ground station commands with specially defined latitude and longitude are sent. This feature works in conjunction with the gcs_routing node, explained in the next section.

2) Pseudo Ground Control Station: The gcs_routing node offers a method for integrating a pseudo ground control station for testing our framework. This node operates much like the xbee node, but adds in the ability to send user-specified commands and telemetry to UAVs running our framework. The user can choose to publish a specific message to either the my_mav_telem topic or the gcs_commands topic to send out a telemetry message or a waypoint command. The gcs_routing node responds to messages posted on these two topics by packing the messages into the correct mavlink format and sending the message through XBee to other UAVs. When switching execution states, the user publishes a message to the gcs_commands topic with specific latitudes and longitudes that correspond to the desired state.

The gcs_routing node can also be run with the ardu node and/or the mover node to further expand testing capabilities. If the ardu node were to be run, the ArduPilot would have to be tethered to the ground control station, leaving the connected ArduPilot with a lower altitude. However, this ArduPilot could then simulate another plane by automatically generating telemetry updates, and can then be positioned so that the plane in the air should avoid it. Additionally, the mover node could be run on the ground station to ensure that the grounded ArduPilot would also avoid the collision as if it were in the air.

V. Results

We tested our framework by running communications tests on the ground and by flying the Bixler with the aforementioned hardware. In the communications test, we were able to successfully receive and send all messages that would come into and out of the Raspberry Pi. The xbee node can successfully receive and decode messages that were sent by other XBees and can forward messages to the appropriate topics within the framework. In addition, the ardu node can successfully read telemetry information from the ArduPilot and can send commands to the ArduPilot such that the ArduPilot updates its destination to be this command. Finally, the mover node correctly follows the state based behaviour that was previously outlined.

The flight tests we conducted made use of one airborne UAV and one UAV tethered to a ground station so that state-based testing could be used. The tethered UAV served the purpose of generating telemetry updates automatically. In addition, the tethered UAV allowed us to visualize the location of the airborne UAV relative to the location of the tethered UAV. In conducting the flight tests, we cycled through each of the flight states to ensure each of them were working correctly.

In the CA_OFF state (mode that forwards goal waypoint) the Bixler was able to successfully fly to its goal waypoint. This verified that the framework does communicate correctly with the ground control station and that the communication is done in such a way that it does not disrupt the functionality of the ArduPilot. In addition, we were able to successfully switch between flight states in mid flight, which allowed for the activation of our algorithm by moving to the CA_ON state.

Our APF algorithm was tested in the air by means of a two step process. The first step taken was that our APF algorithm would be activated when there were no other simulated or real UAVs in the airspace. This test verified that our algorithm would correctly direct the UAV to its goal waypoint by making use of attractive forces. When run on the airborne UAV, the algorithm correctly directed the UAV to the appropriate goal waypoint. The second test performed involved sending out telemetry information for both real and simulated planes so that the UAV in the air would experience repulsive forces. When run on the airborne UAV, the algorithm successfully generated avoidance waypoints that directed the airborne UAV away from both the tethered UAV and the simulated UAVs.

In the ST_RED state, we were able to successfully regain manual control of the plane and land it safely. Finally, the flight behaviour of the Bixler was only slightly changed by the weight of the additional hardware, and the Bixler was still able to move to the correct destination without needing to adjust control parameters.

VI. Conclusion

Decentralized collision avoidance algorithms are especially suited to UAVs' inherent mobility and range, as they reduce the dependency on a ground control station that centralized algorithms require. However, decentralized collision avoidance algorithms require an efficient surrounding framework.

A lightweight, modular framework for testing different distributed collision avoidance algorithms was presented in this paper. Our framework was implemented using an existing, modified APF algorithm on a commercially available drone outfitted with an XBee, Raspberry Pi, and ArduPilot. The functionality of our framework, both with and without collision avoidance, was verified through integration testing as well as flight testing.

VII. Future Work

A. Simulating GPS data

Throughout our testing period, the imprecision and drift of real GPS data made it very difficult to accurately test our algorithm within our framework. In addition, the long setup time and travel required for performing test flights made frequent flight tests impractical. This problem can be addressed in two possible ways - either through some sort of visualization of real time GPS data, similar to how the existing simulator works, or faking more reliable GPS data. Future work could include using the ArduPilot mission planner's simulated GPS capabilities to more efficiently test distributed collision avoidance algorithms without having to physically fly the planes or bring them outside.

B. Integration with Ground Control

During flight tests, our framework was mainly tested with our custom gcs_routing station. Full integration of our framework with the existing ground control station and simulator would allow for easier testing, as the simulator could visually display plane location. In addition, distributed collision avoidance algorithms could be tested with a combination of simulated planes and real planes to minimize loss of hardware. As mentioned in the above section, combining the ArduPilot mission planner's simulated GPS with integration into the existing ground control station would also allow use of the existing testing infrastructure.

C. XBee Networking

One limitation that our work was subject to was that the XBees we used to network our devices were set up so that only pairwise connections were reliable enough to be used. This resulted in only using two XBees in any particular setup. Efforts at adding a third XBee and implementing communication through broadcasts resulted in packet loss and packet clustering. If this system is to be implemented on more than two devices, work needs to be done to configure the XBees so that a reliable n to n network can be established.

Acknowledgments

We would like to thank Dr. Biaz and Auburn University for this amazing opportunity and David Jones for his invaluable assistance in flying the UAVs. We would also like to thank the other members of the 2013 National Science Foundation Research Experience for Undergraduates on Smart UAVs for their help in configuring a distributed simulation environment and Hosea Siu Miriam Figueroa from the 2012 program for their contributions to the APF approach to collision avoidance. Lastly, we would like to thank the Department of Defense and the National Science foundation for their generous support of this program.

References

[1] J. Holt, "Comparison of aerial collision avoidance algorithms in a simulated environment," Ph.D. dissertation, Auburn University, 2012.

[2] U. A. V. Roadmap, "Roadmap 2002-2027," *US DoD, December*, 2002.

[3] H. Siu, M. Figueroa, J. Holt, and S. Biaz, "A dynamic swarm approach to artificial potential field collision avoidance technical report# csse12-02."

[4] B. Albaker and N. Rahim, "A survey of collision avoidance approaches for unmanned aerial vehicles," in *Technical Postgraduates (TECHPOS), 2009 International Conference for*. IEEE, 2009, pp. 1–7.

[5] E. Lalish, K. A. Morgansen, and T. Tsukamaki, "Decentralized reactive collision avoidance for multiple unicycle-type vehicles," in *American Control Conference, 2008*. IEEE, 2008, pp. 5055–5061.

[6] Z. A. Daniels, L. A. Wright, J. M. Holt, and S. Biaz, "Collision avoidance of multiple uas using a collision cone-based cost function."

[7] D. Sislak, M. Rehak, M. Pechoucek, D. s. Pavlicek, and M. Uller, "Negotiation-based approach to unmanned aerial vehicles," in *Distributed Intelligent Systems: Collective Intelligence and Its Applications, 2006. DIS 2006. IEEE Workshop on*. IEEE, 2006, pp. 279–284.

[8] D. Šišlák, M. Pěchouček, P. Volf, D. Pavlíček, J. Samek, V. Mařík, and P. Losiewicz, "Agentfly: Towards multi-agent technology in free flight air traffic control," in *Defence Industry Applications of Autonomous Agents and Multi-Agent Systems*. Springer, 2008, pp. 73–96.

[9] S. Wollkind, J. Valasek, and T. R. Ioerger, "Automated conflict resolution for air traffic management using cooperative multiagent negotiation," in *AIAA guidance, navigation, and control conference*, 2004, pp. 16–19.

[10] G. Zlotkin and J. S. Rosenschein, "Negotiation and task sharing among autonomous agents in cooperative domains," in *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence*. Citeseer, 1989, pp. 912–917.

[11] J. K. Kuchar and L. C. Yang, "A review of conflict detection and resolution modeling methods," *Intelligent Transportation Systems, IEEE Transactions on*, vol. 1, no. 4, pp. 179–189, 2000.

[12] M. Hromatka, J. West, J. Holt, and S. Biaz, "A fuzzy logic approach to collision avoidance in smart uavs technical report csse12-05."

[13] J. Ruchti, R. Senkbeil, J. Carroll, J. Dickinson, J. Holt, and S. Biaz, "Uav collision avoidance using artificial potential fields," Technical Report# CSSE11-03. Auburn University, Tech. Rep., 2011.

[14] O. Khatib, "Real-time obstacle avoidance for manipulators and mobile robots," in *Robotics and Automation. Proceedings. 1985 IEEE International Conference on*, vol. 2. IEEE, 1985, pp. 500–505.

[15] T. Balch and M. Hybinette, "Social potentials for scalable multi-robot formations," in *Robotics and Automation, 2000. Proceedings. ICRA'00. IEEE International Conference on*, vol. 1. IEEE, 2000, pp. 73–80.

[16] L. Barnes, W. Alvis, M. Fields, K. Valavanis, and W. Moreno, "Swarm formation control with potential fields formed by bivariate normal functions," in *Control and Automation, 2006. MED'06. 14th Mediterranean Conference on*. IEEE, 2006, pp. 1–7.

[17] D. E. Chang, S. C. Shadden, J. E. Marsden, and R. Olfati-Saber, "Collision avoidance for multiple agent systems," in *Decision and Control, 2003. Proceedings. 42nd IEEE Conference on*, vol. 1. IEEE, 2003, pp. 539–543.

[18] J. H. William Bates, Sad Biaz and D. Jones, "Configuring autopilot for safe flight and ros simulation," Technical Report# CSSE12-04. Auburn University, Tech. Rep.