2013

A Fuzzy Logic Approach to Collision Avoidance in Smart UAVs

Michelle Hromatka
College of Saint Benedict/Saint John's University

Follow this and additional works at: https://digitalcommons.csbsju.edu/honors_theses

Part of the Computer Sciences Commons

Recommended Citation
https://digitalcommons.csbsju.edu/honors_theses/13

This Thesis is brought to you for free and open access by DigitalCommons@CSB/SJU. It has been accepted for inclusion in Honors Theses, 1963-2015 by an authorized administrator of DigitalCommons@CSB/SJU. For more information, please contact digitalcommons@csbsju.edu.
A Fuzzy Logic Approach to Collision Avoidance in Smart UAVs

In Partial Fulfillment
of the Requirements for Distinction
in the Department of Computer Science
by
Michelle Hromatka

April 2013
Project Title:
A Fuzzy Logic Approach to Collision Avoidance in Smart UAVs

Approved by:

Advisor, Lynn Ziegler
Professor of Computer Science

J. Andrew Holey
Professor of Computer Science

James A. Schnepf
Associate Professor of Computer Science

James A. Schnepf
Chair, Department of Computer Science

Anthony Cunningham
Director, Honors Thesis Program
Abstract

In recent years, there has been a rise in attention given to the issue of collision avoidance techniques for unmanned aerial vehicles (UAVs). Many approaches have been proposed but very few have been tested in high density, fixed speed test conditions. This approach successfully uses fuzzy logic to determine an appropriate avoidance maneuver after a possible collision is detected. Initially, fuzzy logic was also used to detect possible collisions, the reasons this level was removed from the final results are detailed within. This approach was chosen for its adaptability, ease of implementation, and robustness in dealing with questionable sensing techniques. The methods used are explained in detail and the simulated results are presented after testing on two different field sizes, each with four levels of plane density. Overall, this approach was able to reduce collisions by a minimum of 98% in all cases.
Contents
1 Introduction

Collision avoidance techniques for unmanned aerial vehicles (UAVs) have recently become a popular area of study in the field of intelligent robotics. In order for these vehicles to be used in areas such as surveillance or exploration, the vehicles must be able to autonomously react to static and dynamic obstacles while maintaining an efficient path. Various approaches that have been used include path planning, potential fields, geometric methods and fuzzy logic. We first outline the problem in section two, then move on to describe the various approaches to solving the problem, including our chosen approach of using fuzzy logic to avoid detected collisions, in section three. Section four includes our detailed approach and section 5 explains our results with respect to the base case and demonstrates the ability of our approach to avoid collisions in even the most dense situations.

2 Problem Statement

Many algorithms exist to solve the general problem of UAV collision avoidance, but few of these algorithms have been tested in similar test conditions. In order to synthesize results and compare many algorithms in a single simulation, Holt et al. developed a simulator for comparison of various algorithms [1]. By using the simulator, different algorithms can be tested using the same metrics and flight constraints on the same platform. All algorithms are tested in two dimensions, with UAVs traveling at a constant speed of 25 miles per hour and a constant altitude. The turning radius of the UAV was modeled after an EasyStar RC aircraft and limited to 22.5 degrees per second. Telemetry, or position, updates are retrieved once per second and new waypoints directing the UAV can be sent once per second. Environmental factors are excluded from this simulation.

The simulator was designed to test algorithms in real-time. Regardless of how long calculations take, the simulator runs in real-time and new telemetry updates will be generated once per second. If no new waypoint is generated in time by the collision avoidance algorithm, the UAV will continue its course until a new waypoint is received. Each randomly generated course, which is a series of waypoint or goal positions per UAV, will run for 10 minutes which was determined to be an appropriate amount of time to measure an algorithm’s effectiveness and efficiency.
Two metrics were chosen to determine effectiveness. A near miss is defined as one UAV traveling within a radius of 12 meters of another UAV. This is meant to represent a collision, as 12 meters represents approximately the distance travelled in one timestep and is the most accurate measure the simulator allows. A near miss should be avoided at all costs. Similarly, a conflict is defined as one UAV traveling within a radius of 24 meters of another UAV, or approximately two timesteps. This represents the danger zone and should be avoided, but is not as critical as a near miss. Both are shown in Figure 1.

![Diagram showing near miss and conflict](image)

Figure 1: Near Miss and Conflict

A separate metric was chosen to provide a measure of efficiency. This is determined by the number of goal positions, or waypoints that a UAV achieves. An overly cautious algorithm may avoid near misses very well, but the overall number of waypoints reached will be decreased. A delicate balance of effectiveness and efficiency for each algorithm must be achieved. Based on the number of near misses and waypoints reached during the simulation time, an algorithm's efficiency and effectiveness can be determined. Note: for the purposes of this paper, parameters were optimized for low collisions (high effectiveness). As with any algorithm, parameters can be adjusted for better efficiency if some collisions are an acceptable sacrifice for better efficiency.

Previous research has tested Mixed Integer Linear Programming (MILP), Dynamic Sparse A* Algorithm, and Artificial Potential Fields in this sim-
ulation [?]. Each algorithm was run in eight different field situations: a 500m square field and a 1000m square field, each tested with 4, 8, 16 and 32 planes. This will allow for easy comparison of methods presented here to those previously tested.

3 Literature Review

The literature review is split into the following sections: path planning, potential fields, proportional navigation, fuzzy logic, and hybrid approaches. Within these sections, the previously tested Mixed Integer Linear Programming (MILP), Artificial Potential Fields (APF), A* Grid Based Algorithm will be reviewed and explored for strengths and weaknesses. As mentioned in Holt, the general goal of this research is to compare various algorithms using the same simulator and metrics [?]. The next section will provide a comprehensive literature review of fuzzy logic as it has been used in collision avoidance along with the justification of this approach.

3.1 Path Planning

Path planning methods are used in collision avoidance to find an optimal obstacle free path. Common implementations include Mixed Integer Linear Programming (MILP) and A* search which are detailed below. This approach tries to plan a collision free path for each plane in the field space, which can quite clearly be computationally expensive for multiple UAV situations. There are some other path planning techniques, but these are mainly used to increase efficiency in the chosen path, not for finding a collision-free path.

3.1.1 MILP

Schouwenaars et. al successfully used MILP for a solution to single and multi-vehicle path planning algorithm with both static and moving obstacles [?]. MILP uses linear programming with a mix of integers in order to find an optimal solution for the field space. There are multiple commercial implementations of MILP solvers that can be purchased in order to solve the system. One of the major advantages of MILP solvers is the ability to limit inputs and outputs; this is useful in UAV collision avoidance where range of motion is clearly defined, such as a maximum turning radius of 22.5 degrees.
The most basic implementation of this algorithm is the fixed arrival time approach, which finds an optimal path once for the entire field. As mentioned above, this method is often limited by heavy computational load; the solver is not guaranteed to come up with a solution, especially for many UAVs in a relatively small space, before collisions occur [?]. This issue is amplified by dynamic, versus static, obstacles. In order to compensate, there are several other approaches that can be used.

A receding horizon is thus a more common implementation of the MILP approach. This strategy limits the solution to a specified number of time steps into the future, only solving the system for, say, 7 time steps into the future, as opposed to solving the entire system, thus an optimal path is found multiple times throughout the simulation. However, the receding horizon is not infallible. A horizon time limit should not be so short that the vehicles will not have sufficient time to react and avoid collisions, nor should the time be so long that heavy computational load is still an issue. There are further optimization techniques that have been used to counter the issues that come with truncating the horizon time [?].

According to the simulation from Holt et. al., MILP was tested using a licensed online MILP solver in high density testing conditions. The receding horizon approach and other optimization techniques were used, however, the system still failed for more than 8 vehicles in a 500m square airspace [?]. The optimization techniques used were solver time limit, specifying the max amount of time before reverting to a sub-optimal solution, and the use of a "loiter cycle" which the UAV enters until the solver can come up with a solution. The latter solution is not ideal when evaluating an algorithm’s efficiency.

3.1.2 A* Search Algorithm

Another path planning algorithm is that of A* search, or the branch and bound method. In this technique, the field space is divided into a grid and only adjacent squares are considered as a next point along the path. This consideration of other nodes is called branching, estimating the best path from the current square is known as bounding. When a square is branched, the cost is computed based on a heuristic cost function. Each square with its corresponding cost is inserted into a tree structure, typically a heap, which stores the minimum element as the root. As the heap is updated from node to node, the lowest cost path is always at the top of the heap. If the path that
the algorithm is considering is no longer the lowest cost, a lower cost option is readily available, and that path is followed until either a lower cost path is found or the goal point is reached [?]. One of the major issues with the A* search approach is determining an appropriate cost function. As with other path planning techniques, computation time is an issue for real-time collision avoidance. There are some ways to lower computation time, especially if the optimum solution is not required but a nearly optimal path is acceptable.

Other literature has used the A* search to determine the best path for a humanoid robot to place its footsteps in an obstacle littered space [?]. In this case, the foot placement options were limited to 15 locations from each previous footnote.

Figure 2: The degrees of freedom allowed for the humanoid robot’s right foot [?]

For their path, specified at 18 footsteps, the unconstrained A* search tree would contain approximately $10^{21}$ nodes. By limiting the options that A* considers, the final tree contained around 830,000 nodes, significantly less than the predicted $10^{21}$ nodes. By limiting the degrees of freedom allowed in the bounding step, the computation time is decreased dramatically, however, this approach is still too computationally demanding for high density systems. As seen in [?] and [?], this algorithm was tested in situations of varying plane density using the maximum turning radius as a limitation on the degrees of freedom allowed. In each of these cases, A* performed well for low density airspaces, but still yielded many near-misses for dense systems.
3.2 Potential Fields

3.2.1 Artificial Potential Fields

The potential field path planning approach can be split into two different categories: global and local. The global solution requires a priori knowledge of the field and location of the obstacles within that field. The idea of this approach is that artificial negative charges are applied to the obstacles and an artificial positive charge is applied to the goal point. The UAV is then driven to the goal point by the forces equal to the negative gradient of the field $\nabla$.

Figure 3: The gradient along which the UAV is directed $\nabla$.

Similar to other global path planning algorithms, global potential field path planning has very high computation costs, especially with robots that have many degrees of freedom, which makes it unsuitable for real-time path planning. The local potential field approach is much more suitable in real-time solution planning. Real-time, local solutions allow for both static and dynamic obstacles, making solutions much more reactive and realistic. The local potential field approach is set up similarly to the global approach, the path for the robot determined by the sum of forces acting on the robot $\nabla$. Potential field path planning methods are computationally efficient for finding a local obstacle-free path, however this method is not without its faults. As $\nabla$ clearly asserts, there is a well-documented problem with the potential field approach: local minima. When a robot experiences equal and opposite
forces acting upon it, the robot can become stuck in a local minimum. In purely dynamic obstacle avoidance environments, the local minimum problem is likely to have resolved itself by the next timestep, however the interval between timesteps might be enough that a possible collision becomes unavoidable.

### 3.2.2 Total Field

Several implementations of UAV collision avoidance have used real world applications in addition to simulation studies. One approach uses magnetic fields and magnetic sensors to detect nearby UAVs. The goal of this approach is to have decentralized collision avoidance, meaning a UAV only has the knowledge that its sensors detect. A gradient is computed based on objects sensed, and the UAV will be directed to travel in opposition to the gradient. The results were found through simulation, but some real world magnetic sensor readings and reaction tests were conducted to see if the movements seen in the simulator were realistic. These methods worked well for collision avoidance, but yielded sub optimal paths.

For the purposes of this paper, all algorithms are tested in simulation, but the research goal of developing a decentralized method of collision avoidance is seen as advantageous. As Sigurd states, the problem of collision avoidance rapidly becomes intractable using centralized methods as the number of vehicles increases, making a decentralized approach desirable where each vehicle is responsible for its own trajectory planning.

### 3.3 Proportional Navigation

Through the course of literature review, several other types or variations of algorithms were considered. One such algorithm, Proportional Navigation, is widely used in missile guidance problems, and it can be used in collision avoidance by reversing the principle to guide a UAV away from, instead of towards, an obstacle. This principle is based on the Line of Sight (LOS) rate between two moving obstacles, $\lambda$ in $\lambda$. If the LOS between two objects is not changing, then the two objects are on a collision path.

This research proposed a collision avoidance law based on conventional Proportional Navigation (PN) guidance law. A collision avoidance vector is defined and the heading of the aircraft is directed toward this vector. The approach uses the collision cone geometric method to determine when to enter
the collision avoidance mode [?]. The total approach is called Proportional Navigation-based Collision Avoid Guidance (PNCAG). Several conditions for collision avoidance must be met including a range between aircraft and obstacle greater than safety distance, the direction of relative velocity must be outside of the collision cone, and the obstacle must be located behind the direction of the relative velocity vector. These techniques have shown promising results in both efficiency and effectiveness.

3.4 Fuzzy Logic

A common technique used in ground robot collision avoidance is that of fuzzy logic. Fuzzy logic is a form of many valued logic which instead of outputting a 0 or 1, true or false value, outputs a degree of truth or degree of membership within a membership set. For example, if height:short is defined as 50 - 70 inches, and height:tall is defined as 65 - 85 inches, a person could belong to both sets height:short and height:tall because fuzzy logic allows values to be a member of two sets at once [?]. A mathematical algorithm then takes the “fuzzified” value and maps this to a single crisp output. This form of logic is more intuitive and is a better representation of the way humans think. A person is not always exclusively short or exclusively tall, rather someone is “kind of tall”. Similarly, fuzzy logic can be applied to collision avoidance with output membership sets such as “very left” or “slightly right”. Using
a fuzzy logic system also allows for sensor fusion, when data from multiple sensors needs to be combined.

For these reasons, fuzzy logic research is frequently used and has been shown to work in mobile ground robots where sensing techniques are generally inexpensive and inaccurate. Sensor fusion has been accomplished using fuzzy logic in a sense-act approach based on multiple sensors distributed around a 2-D ground robot [7]. This approach was used in situations containing only one robot and stationary obstacles. The mobile ground robot was run through a path containing obstacles where the smoothness of the path was evaluated. Advantages of the fuzzy logic approach are intuitive linguistic terms and a smaller computational load than APF and many other approaches. Fuzzy logic is also useful for unknown and semi-unstructured environments. Sense-act is uninterrupted motion with short development time and incorporates flexible, easy to adjust parameters.

One example of using fuzzy logic in UAV collision avoidance was found. In this scenario, a pre-planned path for the UAV was used until a collision was detected. The static obstacles were of varying shape and size and the avoidance technique sensed these two factors, then found an avoidance maneuver that deviated from the pre-planned path. Again, obstacles were static and changes in speed were allowed in determining an appropriate avoidance maneuver. The UAV was able to successfully avoid collisions with one or multiple obstacles in all cases presented [7]. More information on this approach is detailed in the Methods section, as Dong et al.'s methods from the basis of our fuzzy logic collision avoidance system.

3.5 Hybrid Approach

3.5.1 Electrostatic Potential Fields/Fuzzy Logic

Many approaches to UAV collision avoidance use a combination of two or more methods. An example of one of these hybrid approaches used fuzzy logic to control an autonomous mobile robot (AMR) while travelling along a pre-planned path generated by a total field path planning technique. A 2-layer fuzzy logic (FL) interface was combined with an electrostatic potential field path planner (EPF) to provide real-time mobile robot navigation in a 2-D dynamic environment [7]. The first layer of fuzzy logic provided a sensor fusion system to combine data from several sonar sensors returning two fuzzy variables, direction and distance in four directions: front, back, left,
and right. The second layer provided an obstacle detection module that outputs the position and degree of possibility which a collision may occur. This information is combined in motion control module with the path planner output. If an obstacle blocks the pre-planned EPF path, collision avoidance maneuvers affecting heading and speed are used to avoid the detected collision.

This EPF/FL approach combined planned and reactive behavior into one algorithm. EPF path planning reacts slowly to unknown moving obstacles. The EPF/FL approach allowed EPF to plan the path and allowed the fuzzy inference system to implement the path while avoiding collisions with all obstacles. Some discussion was given into tuning the membership functions, however, no analytical method guarantees optimal selection for fuzzy logic inference systems. Several approaches have been used to tune membership functions, such as neural nets, genetic algorithms, and neuro-fuzzy methods, but none of these approaches guarantee an optimal solution. Collision free paths generated by the EPF method necessarily lead the UAV toward the goal, local minima are not generated within the field and stagnation points within the field do not exist. The path planned by EPF also generates an optimal minimum occupancy path toward the goal.

There are several limitations when applying much of this research to UAV collision avoidance. First, many of the obstacles in this research are static, which is not directly applicable to UAV collision avoidance with multiple UAVs. Mobile ground robots have fewer limitations of freedom of motion than do UAVs. Changes in speed, reversing direction and a 360° turning radius are components of many 2-D ground robots, all three of which are limited or not allowed in this UAV collision avoidance research.

### 3.5.2 Path Planning / Reactive Avoidance

Another hybrid approach used a combination of path planning and reactive collision avoidance. This algorithm calculated a Dubins path for each plane, the shortest path between two points within a vehicles turning constraints, then considered only the most imminent predicted collision for each aircraft. The LOS rate between the two planes is then increased. When a UAV detects multiple planes that are on a collision course with itself, it will select the most imminent collision to avoid that has a predicted "zero effort miss distance" of less than the minimum desired separation. The LOS is increased by applying a lateral acceleration to increase the rate of rotation.
of the LOS connecting these two UAVs. The UAV takes the sharpest turn available until the collision course is avoided.

This research used a very different simulation to test this algorithm. Planes initial positions were chosen to be evenly spaced around a circle that dictated the outside of the test space. A random goal point was assigned to each plane somewhere on a smaller concentric circle. This setup eliminates the chance that random plane positions will be chosen in the test space that are actually inside or very near to the near miss zone of another plane. The PNCAG’s simulation setup, however, does not take into account multiple waypoints. Efficiency of the algorithm was measured by deviation from initial Dubins path.

PNCAG was found to be very efficient as well as effective in reducing the number of near-misses. With an outer circle radius of 500 meters, 20 to 60 UAVs were simulated, giving comparable scenarios to current research goals. The efficiencies achieved were very good (around 90 percent or higher), however, the research cannot be completely compared to results here because multiple waypoints were not achieved (the values were averaged over 10 runs with a single waypoint goal here) and the simulation starting points and goals were chosen in an advantageous manner, as described before.

3.5.3 Passive Pre-Planning / Geometric Reactive

Another hybrid approach used a passive approach to collision detection and avoidance [?]. The research split the collision avoidance process into three steps, all defined by two plane’s relative bearing angle: detection, avoidance, and an optimal return to the pre-planned path. After calculating the bearing angles between the two planes that are at risk of colliding, a geometric approach is used to determine the necessary turning angle necessary to avoid a collision. The minimum separation distance between two UAVs can also be specified. Results of this method were promising, but only two scenarios (overtaking and frontal collision) were presented in this research. This research is more applicable as an addition to algorithms that may have limited effectiveness in these two special cases.
4 Methods

Methods here build upon research described previously: work done by Dong et al. [?] and Tsourveloudis [?]. The two-tiered fuzzy logic system used in Tsourveloudis was combined here with the fuzzy logic collision avoidance used by Dong et al. to develop an efficient and effective algorithm that could handle high-density flight situations.

4.1 Collision Detection System

Similar to Tsourveloudis’ multi level fuzzy system, the first level fuzzy logic controller served as a method to detect possible collisions. Considering only the closest UAV, the first tier used fuzzy logic to output a danger value denoting the likelihood of a collision between the current UAV and the closest UAV. The danger value was determined by inputs of distance to collision, \(d_{col}\), and overlap distance, \(d_{ovlp}\). Both variables are defined below and a graphical representation of these two inputs can be seen in Figure 5.

\[
d_{col} = \min(d_A, d_B) \tag{1}
\]

\[
d_{ovlp} = d_A - d_B \tag{2}
\]

![Figure 5: Common Variables](image-url)
Distance to collision represents an urgency component in detecting a collision. In low density situation, the closest UAV may be many hundreds of meters away from the current UAV in which case either one or both of the UAVs may have changed course prior to reaching the determined collision point. Overlap distance represents a timing factor; if one UAV will reach the collision point outside of the minimum separation distance between two planes, a collision will not occur despite the fact that the paths of two UAVs will intersect. Overlap distance will preserve the efficiency of the algorithm and only avoid a collision point if a collision will actually occur.

Only the closest UAV is considered in each time step. Using the closest UAV is a simplification, but several improvements upon this technique are discussed in the Future Work section. In some cases, distance to collision and overlap distance will not detect a possible collision. For example, two planes flying parallel could be inside each other’s near miss zone, but the planes do not have intersecting paths and the danger value returned by the fuzzy logic would incorrectly be low. This is also discussed in the Future Work along with proposed changes. For these two reasons, the first tier fuzzy engine was left out of the final results, in favor of a simple if statement determining whether the closest plane is within a certain danger radius of the current UAV. The danger radius chosen here was 100 meters. Improvement of these techniques is beyond the scope of this research, but highly recommended for future work as including this level of collision detection has the potential for vast improvements on efficiency.

### 4.2 Collision Avoidance System

The second tier used a fuzzy logic engine to determine a change in heading for the UAV. This tier was originally only to be entered for high danger values output from the first fuzzy system. As stated above this was changed so that this level is entered only if the closest UAV is within a danger radius surrounding the current UAV. Many variations of the second tier fuzzy engine were designed and implemented. Inputs of distance between planes, bearing angle, output from the first fuzzy system, and distance to next waypoint were all considered as possible inputs.

Dong et al. explored fuzzy logic collision avoidance techniques between UAVs and static obstacles. Our approach aims to extend the fuzzy logic approach used by Dong et al. to also be applicable to dynamic collision avoidance. As a frame of reference, that approach was duplicated here and
results can be found below. Dong et al.’s approach, while very useful for static obstacle avoidance, was not found to be an effective approach for dynamic collision avoidance between a group of UAVs in limited airspace.

As a matter of implementation, several changes were made to Dong et al.’s methods. First, the simulator used here does not allow for a change in speed, so no fuzzy output for speed was considered. Second, Dong et al.’s approach limited the turning radius to ±45° but the simulator here sets the limit to ±22.5°.

Several techniques were used to improve the effectiveness of fuzzy logic in dynamic collision avoidance. In order to eliminate oscillations in many cases, the bearing of the opposite plane was added to the general approach described by Dong et al. The fuzzy sets for bearing angles and distances vary slightly as well. The output of change in heading is limited to five fuzzy set possibilities, simplifying rule choices. The final design of the fuzzy logic approach is outlined below. Figures ??, ??, and ?? show the ranges and implementations for each fuzzy input and output set. The bearing angle set is identical for the bearing angle of the current plane, $\theta_1$, and that of the opposite plane, $\theta_2$.

![Figure 6: Distance Fuzzy Sets, the linguistic terms used to describe from left to right: Very Close [VC], Close [C], Far [F], Very Far [VF].](image)

The range of values were chosen based on the limitations of the aircraft in the simulator. Turns are limited to ±22.5°, corresponding to the center of the outer fuzzy sets, ensuring that the fuzzy engine never returns a change
Figure 7: Bearing Angle Fuzzy Sets, the linguistic terms used to describe from left to right: Very Negative [VN], Negative [N], Low Negative [LN], Low Positive [LP], Positive [P], Very Positive [VP]

in heading greater than the maximum value allowed

The rules that correspond to the previous pictures are outlined in the following tables. Tables ?? - ?? correspond to the fuzzy sets of VC, C, F, VF for Distance Between Planes, respectively. \( \theta_1 \) corresponds to the fuzzy sets for the current plane’s bearing angle. \( \theta_2 \) corresponds to the fuzzy sets for the opposite plane’s bearing angle, or in other words the plane under consideration for a possible collision. These tables can be read by finding the membership sets that inputs belong to, then using this information to determine an appropriate collision avoidance maneuver. For a very simple example where there is no overlap in membership sets, for a plane distance that maps to the set Very Close, with a \( \theta_1 \) of Low Negative and a \( \theta_2 \) of Positive, the output would map to a Very Left turn.
Table 1: Fuzzy Engine Rule Set - Plane Distance Very Close [VC]

<table>
<thead>
<tr>
<th>$\theta_1$</th>
<th>VN</th>
<th>N</th>
<th>LN</th>
<th>LP</th>
<th>P</th>
<th>VP</th>
</tr>
</thead>
<tbody>
<tr>
<td>VN</td>
<td>R</td>
<td>R</td>
<td>R</td>
<td>R</td>
<td>VR</td>
<td>VR</td>
</tr>
<tr>
<td>N</td>
<td>R</td>
<td>R</td>
<td>VR</td>
<td>VR</td>
<td>VR</td>
<td>VR</td>
</tr>
<tr>
<td>LN</td>
<td>VR</td>
<td>VR</td>
<td>VR</td>
<td>VR</td>
<td>VL</td>
<td>VL</td>
</tr>
<tr>
<td>LP</td>
<td>VR</td>
<td>VR</td>
<td>VR</td>
<td>VL</td>
<td>VL</td>
<td>VL</td>
</tr>
<tr>
<td>P</td>
<td>VL</td>
<td>VL</td>
<td>VL</td>
<td>VL</td>
<td>L</td>
<td>L</td>
</tr>
<tr>
<td>VP</td>
<td>VL</td>
<td>VL</td>
<td>VL</td>
<td>VL</td>
<td>L</td>
<td>L</td>
</tr>
</tbody>
</table>

Table 2: Fuzzy Engine Rule Set - Plane Distance Close [C]

<table>
<thead>
<tr>
<th>$\theta_1$</th>
<th>VN</th>
<th>N</th>
<th>LN</th>
<th>LP</th>
<th>P</th>
<th>VP</th>
</tr>
</thead>
<tbody>
<tr>
<td>VN</td>
<td>NC</td>
<td>NC</td>
<td>R</td>
<td>R</td>
<td>R</td>
<td>R</td>
</tr>
<tr>
<td>N</td>
<td>NC</td>
<td>R</td>
<td>VR</td>
<td>VR</td>
<td>VR</td>
<td>VR</td>
</tr>
<tr>
<td>LN</td>
<td>VR</td>
<td>VR</td>
<td>VR</td>
<td>VR</td>
<td>VL</td>
<td>VL</td>
</tr>
<tr>
<td>LP</td>
<td>VR</td>
<td>VR</td>
<td>VR</td>
<td>VL</td>
<td>VL</td>
<td>VL</td>
</tr>
<tr>
<td>P</td>
<td>VL</td>
<td>VL</td>
<td>VL</td>
<td>VL</td>
<td>L</td>
<td>NC</td>
</tr>
<tr>
<td>VP</td>
<td>L</td>
<td>L</td>
<td>L</td>
<td>L</td>
<td>NC</td>
<td>NC</td>
</tr>
</tbody>
</table>

Table 3: Fuzzy Engine Rule Set - Plane Distance Far [F]

<table>
<thead>
<tr>
<th>$\theta_1$</th>
<th>VN</th>
<th>N</th>
<th>LN</th>
<th>LP</th>
<th>P</th>
<th>VP</th>
</tr>
</thead>
<tbody>
<tr>
<td>VN</td>
<td>NC</td>
<td>NC</td>
<td>R</td>
<td>R</td>
<td>R</td>
<td>R</td>
</tr>
<tr>
<td>N</td>
<td>NC</td>
<td>R</td>
<td>VR</td>
<td>VR</td>
<td>VR</td>
<td>VR</td>
</tr>
<tr>
<td>LN</td>
<td>VR</td>
<td>VR</td>
<td>VR</td>
<td>VR</td>
<td>VL</td>
<td>VL</td>
</tr>
<tr>
<td>LP</td>
<td>VR</td>
<td>VR</td>
<td>VR</td>
<td>VL</td>
<td>VL</td>
<td>VL</td>
</tr>
<tr>
<td>P</td>
<td>VL</td>
<td>VL</td>
<td>VL</td>
<td>VL</td>
<td>L</td>
<td>NC</td>
</tr>
<tr>
<td>VP</td>
<td>L</td>
<td>L</td>
<td>L</td>
<td>L</td>
<td>NC</td>
<td>NC</td>
</tr>
</tbody>
</table>
Figure 8: Change In Heading Fuzzy Sets, the linguistic terms used to describe from left to right: Very Left [VL], Left [L], No Change [NC], Right [R], Very Right [VR].

<table>
<thead>
<tr>
<th>$\theta_1$</th>
<th>VN</th>
<th>N</th>
<th>LN</th>
<th>LP</th>
<th>P</th>
<th>VP</th>
</tr>
</thead>
<tbody>
<tr>
<td>VN</td>
<td>NC</td>
<td>NC</td>
<td>R</td>
<td>R</td>
<td>R</td>
<td>R</td>
</tr>
<tr>
<td>N</td>
<td>NC</td>
<td>R</td>
<td>VR</td>
<td>VR</td>
<td>VR</td>
<td>VR</td>
</tr>
<tr>
<td>LN</td>
<td>VR</td>
<td>VR</td>
<td>VR</td>
<td>VL</td>
<td>VL</td>
<td>VL</td>
</tr>
<tr>
<td>LP</td>
<td>VR</td>
<td>VR</td>
<td>VL</td>
<td>VL</td>
<td>VL</td>
<td>VL</td>
</tr>
<tr>
<td>P</td>
<td>VL</td>
<td>VL</td>
<td>VL</td>
<td>L</td>
<td>NC</td>
<td>NC</td>
</tr>
<tr>
<td>VP</td>
<td>L</td>
<td>L</td>
<td>L</td>
<td>L</td>
<td>NC</td>
<td>NC</td>
</tr>
</tbody>
</table>

5 Results

All results were simulated using the ROS based simulator system as described in [?]. In the simulator, a random course file is generated which includes 50 waypoints for each plane, including random start points. For the field sizes of 500m square and 1000m square, each with 4, 8, 16 and 32 planes, three random course files were generated. A note on the simulator: although a plane may experienced a near-miss, the plane continues to fly throughout the simulation. This is to keep the flight situation the same throughout the simulation run time. If UAVs left the simulation after experiencing a
near-miss, the simulator would favor an algorithm that stabilized quicker than others. A collision avoidance algorithm may operate very well with 24 UAVs in a 500m airspace, so as soon as enough planes have near-misses, the simulation would stabilize. The purpose of this research is not to find the breaking point of an avoidance algorithm, but to test how well an avoidance algorithms can handle high-density flight situations.

Each course was run in the simulator without collision avoidance for every scenario as a frame of reference. Results of Dong et al.'s methods and the methods presented here are compared to this base case. Tables 5 and 6 display the base case results.

As mentioned previously, Dong et al. made great progress on collision avoidance of UAVs with static obstacles. Their methods are repeated here for collision avoidance between UAVs in limited airspace. Results of Dong et al.'s methods applied to dynamic collision avoidance are displayed below in Tables 7 and 8.

This approach improved upon the base case in most categories and field sizes, but can be optimized further. Changes mentioned previously, such as adding the bearing angle of the opposite plane as a fuzzy input, were made to the methods of Dong et al. and the results are detailed below. Results from methods used in this research are displayed in Tables 9 and 10.

<p>| Table 5: No Collision Avoidance Results - 500m Field Size |</p>
<table>
<thead>
<tr>
<th>No.</th>
<th>Waypoints</th>
<th>Conflicts</th>
<th>Near Misses</th>
<th>Distance Ratio</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>84.0</td>
<td>20.0</td>
<td>3.33</td>
<td>1.09</td>
</tr>
<tr>
<td>8</td>
<td>143.0</td>
<td>123.3</td>
<td>29.0</td>
<td>1.10</td>
</tr>
<tr>
<td>16</td>
<td>311.7</td>
<td>528.7</td>
<td>136.7</td>
<td>1.11</td>
</tr>
<tr>
<td>32</td>
<td>666.7</td>
<td>2448</td>
<td>596.7</td>
<td>1.11</td>
</tr>
</tbody>
</table>

5.1 Efficiency

Two efficiency metrics were used to compare algorithms: the number of waypoints reached, and the distance ratio, $\alpha$. The distance ratio is the ratio
### Table 6: No Collision Avoidance Results - 1000m Field Size

<table>
<thead>
<tr>
<th>No.</th>
<th>Waypoints</th>
<th>Conflicts</th>
<th>Near Misses</th>
<th>Distance Ratio</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>39.7</td>
<td>10.0</td>
<td>1.0</td>
<td>1.04</td>
</tr>
<tr>
<td>8</td>
<td>92.0</td>
<td>43.3</td>
<td>12.3</td>
<td>1.05</td>
</tr>
<tr>
<td>16</td>
<td>185.7</td>
<td>179.7</td>
<td>44.7</td>
<td>1.05</td>
</tr>
<tr>
<td>32</td>
<td>378.7</td>
<td>665.0</td>
<td>159.3</td>
<td>1.05</td>
</tr>
</tbody>
</table>

### Table 7: Dong et al. Methods - 500m Field Size

<table>
<thead>
<tr>
<th>No.</th>
<th>Waypoints</th>
<th>Conflicts</th>
<th>Near Misses</th>
<th>Distance Ratio</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>53.3</td>
<td>8.0</td>
<td>3.3</td>
<td>1.81</td>
</tr>
<tr>
<td>8</td>
<td>71.7</td>
<td>58.3</td>
<td>14.3</td>
<td>2.50</td>
</tr>
<tr>
<td>16</td>
<td>72.3</td>
<td>225.0</td>
<td>54.0</td>
<td>4.72</td>
</tr>
<tr>
<td>32</td>
<td>56.7</td>
<td>686.3</td>
<td>172.7</td>
<td>7.30</td>
</tr>
</tbody>
</table>

### Table 8: Dong et al. Methods - 1000m Field Size

<table>
<thead>
<tr>
<th>No.</th>
<th>Waypoints</th>
<th>Conflicts</th>
<th>Near Misses</th>
<th>Distance Ratio</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>32.0</td>
<td>7.0</td>
<td>1.3</td>
<td>1.36</td>
</tr>
<tr>
<td>8</td>
<td>59.3</td>
<td>10.0</td>
<td>4.0</td>
<td>1.52</td>
</tr>
<tr>
<td>16</td>
<td>107.7</td>
<td>355.0</td>
<td>27.3</td>
<td>1.86</td>
</tr>
<tr>
<td>32</td>
<td>110.3</td>
<td>428.7</td>
<td>111.3</td>
<td>2.77</td>
</tr>
</tbody>
</table>
### Table 9: Final Results - 500m Field Size

<table>
<thead>
<tr>
<th>No.</th>
<th>Waypoints</th>
<th>Conflicts</th>
<th>Near Misses</th>
<th>Distance Ratio</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>62.7</td>
<td>0.0</td>
<td>0.0</td>
<td>1.67</td>
</tr>
<tr>
<td>8</td>
<td>82.3</td>
<td>0.0</td>
<td>0.0</td>
<td>1.08</td>
</tr>
<tr>
<td>16</td>
<td>76.0</td>
<td>14.7</td>
<td>2.3</td>
<td>4.27</td>
</tr>
<tr>
<td>32</td>
<td>50.0</td>
<td>64.0</td>
<td>12.3</td>
<td>7.55</td>
</tr>
</tbody>
</table>

### Table 10: Final Results - 1000m Field Size

<table>
<thead>
<tr>
<th>No.</th>
<th>Waypoints</th>
<th>Conflicts</th>
<th>Near Misses</th>
<th>Distance Ratio</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>39.7</td>
<td>0.0</td>
<td>0.0</td>
<td>1.08</td>
</tr>
<tr>
<td>8</td>
<td>80.3</td>
<td>0.0</td>
<td>0.0</td>
<td>1.17</td>
</tr>
<tr>
<td>16</td>
<td>128.7</td>
<td>1.7</td>
<td>0.3</td>
<td>1.51</td>
</tr>
<tr>
<td>32</td>
<td>160.3</td>
<td>21.0</td>
<td>3.0</td>
<td>2.07</td>
</tr>
</tbody>
</table>
of the distances the UAV travelled to reach the waypoints over the minimum, straight-line travel distance.

\[ \alpha = \frac{d_{\text{actual}}}{d_{\text{min}}} \] (3)

The distance ratio only takes into account the distance travelled up to the last waypoint reached, thus this distance is not counted in \( \alpha \) when the simulation ends. This is also true in calculating \( d_{\text{min}} \). This metric shows how much the plane has deviated from a near-optimal path. The number of waypoints reached is also a metric on efficiency and helps to balance the limitations of the distance ratio metric. The planes attempt to reach all waypoints in order within the specified time limit. A plane may have a near perfect distance ratio, say it reaches one waypoint on a straight line path, but then fails to reach another waypoint during the simulation. Although for this individual UAV the distance ratio is near perfect, the number of waypoints reached will be significantly lower. Using waypoints reached counters the distance travelled that the distance ratio fails to take into account as the UAVs travel at fixed speed.

As seen in the base case no avoidance results, the base case distance ratio is not a perfect 1.0, as \( d_{\text{min}} \) is the straight distance between waypoints, not the minimum flight path which would allow for the maximum turning radius. As expected, the distance ratio in high-density flight situations using a collision avoidance system is significantly higher than in the related no avoidance case. Similarly, the number of waypoints reached in the avoidance simulation results are significantly lower than in the no avoidance simulation results.

5.2 Effectiveness

Effectiveness is a measure of conflict-reduction and near-miss reduction. Again, a near-miss is defined as one UAV traveling within a radius of 12 meters of another UAV. A note on these metrics: for this purpose, the main goal was to achieve zero collisions in each flight situation. Depending on the situation, waypoints may be given higher priority. By changing several of the rules in the collision avoidance fuzzy system to output less extreme changes in heading, more waypoints are achieved but the number of collisions increases.

(* denotes areas where Dong et al. performed worse than the base case.)
### Table 11: Dong et al. Methods - 500m Field Size

<table>
<thead>
<tr>
<th>No. Planes</th>
<th>Waypoints Reached</th>
<th>Distance Ratio</th>
<th>Near-Miss Reduction</th>
<th>Conflict Reduction</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>53.3</td>
<td>1.81</td>
<td>0.0%*</td>
<td>60.0%</td>
</tr>
<tr>
<td>8</td>
<td>71.7</td>
<td>2.50</td>
<td>28.5%</td>
<td>52.7%</td>
</tr>
<tr>
<td>16</td>
<td>72.3</td>
<td>4.72</td>
<td>60.5%</td>
<td>57.4%</td>
</tr>
<tr>
<td>32</td>
<td>172.7</td>
<td>7.30</td>
<td>71.1%</td>
<td>72.0%</td>
</tr>
</tbody>
</table>

### Table 12: Dong et al. Methods - 1000m Field Size

<table>
<thead>
<tr>
<th>No. Planes</th>
<th>Waypoints Reached</th>
<th>Distance Ratio</th>
<th>Near-Miss Reduction</th>
<th>Conflict Reduction</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>32.0</td>
<td>1.36</td>
<td>0.0%*</td>
<td>30.0%</td>
</tr>
<tr>
<td>8</td>
<td>59.3</td>
<td>1.52</td>
<td>67.50%</td>
<td>76.92%</td>
</tr>
<tr>
<td>16</td>
<td>107.7</td>
<td>1.86</td>
<td>38.93%</td>
<td>0.0%*</td>
</tr>
<tr>
<td>32</td>
<td>110.3</td>
<td>2.77</td>
<td>30.3%</td>
<td>35.5%</td>
</tr>
</tbody>
</table>
This approach improved upon the base case in most categories and field sizes, but can be optimized further. Changes mentioned previously, such as adding the bearing angle of the opposite plane as a fuzzy input, were made to the methods of Dong et al. and the results are detailed below. These are the results from the fuzzy rules and fuzzy set dimensions described in graphs and tables previously.

### Table 13: Final Results - 500m Field Size

<table>
<thead>
<tr>
<th>No. Planes</th>
<th>Waypoints Reached</th>
<th>Distance Ratio</th>
<th>Near-Miss Reduction</th>
<th>Conflict Reduction</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>62.7</td>
<td>1.67</td>
<td>100.0%</td>
<td>100.0%</td>
</tr>
<tr>
<td>8</td>
<td>82.3</td>
<td>1.08</td>
<td>100.0%</td>
<td>100.0%</td>
</tr>
<tr>
<td>16</td>
<td>76.0</td>
<td>4.27</td>
<td>98.32%</td>
<td>97.59%</td>
</tr>
<tr>
<td>32</td>
<td>50.0</td>
<td>7.55</td>
<td>97.94%</td>
<td>97.39%</td>
</tr>
</tbody>
</table>

### Table 14: Final Results - 1000m Field Size

<table>
<thead>
<tr>
<th>No. Planes</th>
<th>Waypoints Reached</th>
<th>Distance Ratio</th>
<th>Near-Miss Reduction</th>
<th>Conflict Reduction</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>39.7</td>
<td>1.08</td>
<td>100.0%</td>
<td>100.0%</td>
</tr>
<tr>
<td>8</td>
<td>80.3</td>
<td>1.17</td>
<td>100.0%</td>
<td>100.0%</td>
</tr>
<tr>
<td>16</td>
<td>128.7</td>
<td>1.51</td>
<td>99.33%</td>
<td>99.05%</td>
</tr>
<tr>
<td>32</td>
<td>160.3</td>
<td>2.07</td>
<td>98.12%</td>
<td>99.55%</td>
</tr>
</tbody>
</table>

### 6 Conclusions

This paper presents a solution to high-density UAV collision avoidance using fuzzy logic. While we still experienced near-misses in the higher density cases, the overall reduction in near-misses as compared to the base case was significant. In every
field situation, we were able to decrease the number of near-misses compared to the base case by a minimum of 97%, 100% in the lower density cases. By using a simple determination mechanism to see if a UAV is in a collision situation, a fuzzy inference system is used to determine the appropriate change in heading based on the bearing angles of each plane and the distance between both planes. Results were generated by using a ROS based simulator which sends position updates for each UAV once per second. Results were evaluated based on effectiveness in avoiding collisions as well as efficiency in path taken and waypoints reached.

The future work section details the areas of improvement that future research can take. Fuzzy logic systems should be considered as a valid solution to UAV collision avoidance, especially with a sensing system that is prone to noise or inaccuracies.

7 Future Work

One major advantage of a fuzzy logic system used in collision avoidance is the ability to deal with uncertainty of inputs and outputs. In simulation, inputs are known exactly, but this is not always the case in real-world applications. Variance in data due to weather or noise could affect the certainty of data. Ranges of fuzzy sets will be able to handle this uncertainty of input measurement values as well as other system disturbances. A real-world implementation is part of intended future work of this research.

Other future work includes a better approach for determining the most dangerous UAV, or determining the UAV with which a collision is most imminent. As stated above, only the closest UAV to the current UAV was considered. Several geometric approaches exist that would be better suited to determining the most dangerous UAV, instead of the closest UAV. Exploring the use of a different determination mechanism would likely increase both efficiency and effectiveness. Limitations of our current approach include increased risk of collisions with planes flying relatively parallel or heading directly towards one another. One such approach that eliminates these two limitations is presented by Ghose et al. by using their Zero Effort Miss metric [?].

Future work will also include further optimization of fuzzy sets and rules. Time limited the number of test cases, and the sets and rules presented here are manually tuned. Other optimization techniques for fuzzy logic engines will be explored and implemented in the future.
8 Source Code

This code was designed for use in conjunction with the simulator designed by Holt et al. which can be found at https://github.com/holtjma/AU-UAV-ROS. The Fuzzy Lite library developed by Juan Rada-Vilela which was used to design the fuzzy logic system can be found at http://code.google.com/p/fuzzylite/. All of the code may be found on our github, https://github.com/JeffreyWest/FuzzyLogic.

8.1 Fuzzy Logic System

This code is the basis for the fuzzy logic system used to determine an appropriate collision avoidance maneuver. Further details about the use of the FuzzyLite library can be found at http://code.google.com/p/fuzzylite/. The first fuzzy logic system (oursAndTheirs2) is a combination of Dong et al.’s methods

```cpp
/*
* Fuzzy logic controllers for detecting collisions and
* for determining appropriate collision avoidance maneuver
*
* @authors Michelle Hromatka, Jeffrey West
*/

#include "ros/ros.h"
#include "fuzzylite/FuzzyLite.h"
#include "AU_UAV_ROS/FuzzyLogicController.h"
#include <string>
#include <sstream>
#include <fstream>

double AU_UAV_ROS::FuzzyLogicController::oursandtheirs2(double distance, double ourAngle, double theirAngle)
{
    fl::FuzzyOperator& op = fl::FuzzyOperator::DefaultFuzzyOperator();
    fl::FuzzyEngine engine("Heading - Change", op);
    fl::InputLVar* distanceBetweenPlanes = new fl::InputLVar("PlaneDist");
    distanceBetweenPlanes->addTerm(new fl::ShoulderTerm("VERYCLOSE", 24.0, 50.0, true));
    distanceBetweenPlanes->addTerm(new fl::TriangularTerm("CLOSE", 24.0, 65.0));
    distanceBetweenPlanes->addTerm(new fl::TriangularTerm("FAR", 35.0, 76.0));
    distanceBetweenPlanes->addTerm(new fl::ShoulderTerm("VERYFAR", 50.0, 76.0, false));
    engine.addInputLVar(distanceBetweenPlanes);

    fl::InputLVar* ourBearingAngle = new fl::InputLVar("OurBearingAngle");
    ourBearingAngle->addTerm(new fl::ShoulderTerm("VERYNEG", -90.0, -45.0, true));
    ourBearingAngle->addTerm(new fl::TriangularTerm("NEG", -89, -44.0));
    ourBearingAngle->addTerm(new fl::TriangularTerm("LOWNEG", -45.0, 2.0));
    ourBearingAngle->addTerm(new fl::TriangularTerm("LOWPOS", 1.0, 45.0));
    ourBearingAngle->addTerm(new fl::TriangularTerm("POS", 44, 89.0));
    ourBearingAngle->addTerm(new fl::ShoulderTerm("VERYPOS", 45, 90.0, false));
    engine.addInputLVar(ourBearingAngle);
```
fl::InputLVar* theirBearingAngle = new fl::InputLVar("TheirBearingAngle");
theirBearingAngle->addTerm(new fl::ShoulderTerm("VERYNEG", -90.0, -45.0, true));
theirBearingAngle->addTerm(new fl::TriangularTerm("NEG", -89, -44.0));
theirBearingAngle->addTerm(new fl::TriangularTerm("LOWNEG", -45.0, 2.0));
theirBearingAngle->addTerm(new fl::TriangularTerm("LOWPOS", 1.0, 45.0));
theirBearingAngle->addTerm(new fl::TriangularTerm("POS", 44, 89.0));
theirBearingAngle->addTerm(new fl::ShoulderTerm("VERYPOS", 45, 90.0, false));
engine.addInputLVar(theirBearingAngle);

fl::OutputLVar* changeHeading = new fl::OutputLVar("ChangeInHeading");
changeHeading->addTerm(new fl::TriangularTerm("VERYLEFT", -45, 0.0));
changeHeading->addTerm(new fl::TriangularTerm("LEFT", -22.5, 0.0));
changeHeading->addTerm(new fl::TriangularTerm("NOCHANGE", -22.5, 22.5));
changeHeading->addTerm(new fl::TriangularTerm("RIGHT", 0.0, 22.5));
changeHeading->addTerm(new fl::TriangularTerm("VERYRIGHT", 0.0, 45));
engine.addOutputLVar(changeHeading);

fl::RuleBlock* block = new fl::RuleBlock();
//PlaneDist is VERYCLOSE
block->addRule(new fl::MamdaniRule("if OurBearingAngle is VERYNEG and TheirBearingAngle" &
" is VERYNEG and PlaneDist is VERYCLOSE then ChangeInHeading is RIGHT", engine));
block->addRule(new fl::MamdaniRule("if OurBearingAngle is VERYNEG and TheirBearingAngle" &
" is LOWNEG and PlaneDist is VERYCLOSE then ChangeInHeading is RIGHT", engine));
block->addRule(new fl::MamdaniRule("if OurBearingAngle is VERYNEG and TheirBearingAngle" &
" is LOWPOS and PlaneDist is VERYCLOSE then ChangeInHeading is RIGHT", engine));
block->addRule(new fl::MamdaniRule("if OurBearingAngle is VERYNEG and TheirBearingAngle" &
" is POS and PlaneDist is VERYCLOSE then ChangeInHeading is RIGHT", engine));
block->addRule(new fl::MamdaniRule("if OurBearingAngle is VERYNEG and TheirBearingAngle" &
" is VERYPOS and PlaneDist is VERYCLOSE then ChangeInHeading is VERYRIGHT", engine));
block->addRule(new fl::MamdaniRule("if OurBearingAngle is NEG and TheirBearingAngle" &
" is VERYNEG and PlaneDist is VERYCLOSE then ChangeInHeading is RIGHT", engine));
block->addRule(new fl::MamdaniRule("if OurBearingAngle is NEG and TheirBearingAngle" &
" is NEG and PlaneDist is VERYCLOSE then ChangeInHeading is RIGHT", engine));
block->addRule(new fl::MamdaniRule("if OurBearingAngle is NEG and TheirBearingAngle" &
" is LOWNEG and PlaneDist is VERYCLOSE then ChangeInHeading is VERYRIGHT", engine));
block->addRule(new fl::MamdaniRule("if OurBearingAngle is NEG and TheirBearingAngle" &
" is LOWPOS and PlaneDist is VERYCLOSE then ChangeInHeading is VERYRIGHT", engine));
block->addRule(new fl::MamdaniRule("if OurBearingAngle is NEG and TheirBearingAngle" &
" is POS and PlaneDist is VERYCLOSE then ChangeInHeading is VERYRIGHT", engine));
block->addRule(new fl::MamdaniRule("if OurBearingAngle is LOWNEG and TheirBearingAngle" &
" is VERYNEG and PlaneDist is VERYCLOSE then ChangeInHeading is RIGHT", engine));
block->addRule(new fl::MamdaniRule("if OurBearingAngle is LOWNEG and TheirBearingAngle" &
" is LOWNEG and PlaneDist is VERYCLOSE then ChangeInHeading is VERYRIGHT", engine));
block->addRule(new fl::MamdaniRule("if OurBearingAngle is LOWNEG and TheirBearingAngle" &
" is LOWPOS and PlaneDist is VERYCLOSE then ChangeInHeading is VERYRIGHT", engine));
block->addRule(new fl::MamdaniRule("if OurBearingAngle is LOWNEG and TheirBearingAngle" &
" is POS and PlaneDist is VERYCLOSE then ChangeInHeading is VERYLEFT", engine));
block->addRule(new fl::MamdaniRule("if OurBearingAngle is LOWNEG and TheirBearingAngle" &
"is VERYPOS and PlaneDist is VERYCLOSE then ChangeInHeading is VERYLEFT", engine));
block->addRule(new fl::MamdaniRule("if OurBearingAngle is LOWPOS and TheirBearingAngle" &
  "is VERYNEG and PlaneDist is VERYCLOSE then ChangeInHeading is LEFT", engine));
block->addRule(new fl::MamdaniRule("if OurBearingAngle is LOWPOS and TheirBearingAngle" &
  "is NEG and PlaneDist is VERYCLOSE then ChangeInHeading is LEFT", engine));
block->addRule(new fl::MamdaniRule("if OurBearingAngle is LOWPOS and TheirBearingAngle" &
  "is LOWNEG and PlaneDist is VERYCLOSE then ChangeInHeading is LEFT", engine));
block->addRule(new fl::MamdaniRule("if OurBearingAngle is LOWPOS and TheirBearingAngle" &
  "is POS and PlaneDist is VERYCLOSE then ChangeInHeading is LEFT", engine));
block->addRule(new fl::MamdaniRule("if OurBearingAngle is LOWPOS and TheirBearingAngle" &
  "is VERYPOS and PlaneDist is VERYCLOSE then ChangeInHeading is LEFT", engine));
block->addRule(new fl::MamdaniRule("if OurBearingAngle is POS and TheirBearingAngle" &
  "is VERYNEG and PlaneDist is VERYCLOSE then ChangeInHeading is LEFT", engine));
block->addRule(new fl::MamdaniRule("if OurBearingAngle is POS and TheirBearingAngle" &
  "is NEG and PlaneDist is VERYCLOSE then ChangeInHeading is LEFT", engine));
block->addRule(new fl::MamdaniRule("if OurBearingAngle is POS and TheirBearingAngle" &
  "is LOWNEG and PlaneDist is VERYCLOSE then ChangeInHeading is LEFT", engine));
block->addRule(new fl::MamdaniRule("if OurBearingAngle is POS and TheirBearingAngle" &
  "is LOWPOS and PlaneDist is VERYCLOSE then ChangeInHeading is LEFT", engine));
block->addRule(new fl::MamdaniRule("if OurBearingAngle is POS and TheirBearingAngle" &
  "is POS and PlaneDist is VERYCLOSE then ChangeInHeading is LEFT", engine));
block->addRule(new fl::MamdaniRule("if OurBearingAngle is POS and TheirBearingAngle" &
  "is VERYPOS and PlaneDist is VERYCLOSE then ChangeInHeading is LEFT", engine));
block->addRule(new fl::MamdaniRule("if OurBearingAngle is POS and TheirBearingAngle" &
  "is VERYNEG and PlaneDist is CLOSE then ChangeInHeading is NOCHANGE", engine));
block->addRule(new fl::MamdaniRule("if OurBearingAngle is POS and TheirBearingAngle" &
  "is NEG and PlaneDist is CLOSE then ChangeInHeading is NOCHANGE", engine));
block->addRule(new fl::MamdaniRule("if OurBearingAngle is POS and TheirBearingAngle" &
  "is LOWNEG and PlaneDist is CLOSE then ChangeInHeading is RIGHT", engine));
block->addRule(new fl::MamdaniRule("if OurBearingAngle is POS and TheirBearingAngle" &
  "is LOWPOS and PlaneDist is CLOSE then ChangeInHeading is RIGHT", engine));
block->addRule(new fl::MamdaniRule("if OurBearingAngle is POS and TheirBearingAngle" &
  "is POS and PlaneDist is CLOSE then ChangeInHeading is RIGHT", engine));
block->addRule(new fl::MamdaniRule("if OurBearingAngle is POS and TheirBearingAngle" &
  "is VERYPOS and PlaneDist is CLOSE then ChangeInHeading is NOCHANGE", engine));
block->addRule(new fl::MamdaniRule("if OurBearingAngle is NEG and TheirBearingAngle is LOWNEG and PlaneDist is CLOSE then ChangeInHeading is VERYRIGHT", engine));
block->addRule(new fl::MamdaniRule("if OurBearingAngle is POS and TheirBearingAngle is LOWNEG and PlaneDist is CLOSE then ChangeInHeading is LEFT", engine));
block->addRule(new fl::MamdaniRule("if OurBearingAngle is POS and TheirBearingAngle is LOWPOS and PlaneDist is CLOSE then ChangeInHeading is VERYRIGHT", engine));
block->addRule(new fl::MamdaniRule("if OurBearingAngle is LOWNEG and TheirBearingAngle is POS and PlaneDist is CLOSE then ChangeInHeading is VERYLEFT", engine));
block->addRule(new fl::MamdaniRule("if OurBearingAngle is LOWNEG and TheirBearingAngle is LOWPOS and PlaneDist is CLOSE then ChangeInHeading is VERYLEFT", engine));
block->addRule(new fl::MamdaniRule("if OurBearingAngle is LOWNEG and TheirBearingAngle is LOWNEG and PlaneDist is CLOSE then ChangeInHeading is VERYRIGHT", engine));
block->addRule(new fl::MamdaniRule("if OurBearingAngle is LOWNEG and TheirBearingAngle is LOWNEG and PlaneDist is CLOSE then ChangeInHeading is VERYLEFT", engine));
block->addRule(new fl::MamdaniRule("if OurBearingAngle is LOWPOS and TheirBearingAngle is POS and PlaneDist is CLOSE then ChangeInHeading is VERYLEFT", engine));
block->addRule(new fl::MamdaniRule("if OurBearingAngle is LOWPOS and TheirBearingAngle is LOWPOS and PlaneDist is CLOSE then ChangeInHeading is VERYLEFT", engine));
block->addRule(new fl::MamdaniRule("if OurBearingAngle is LOWPOS and TheirBearingAngle is LOWPOS and PlaneDist is CLOSE then ChangeInHeading is LEFT", engine));
block->addRule(new fl::MamdaniRule("if OurBearingAngle is LOWPOS and TheirBearingAngle is LOWPOS and PlaneDist is CLOSE then ChangeInHeading is NOCHANGE", engine));
block->addRule(new fl::MamdaniRule("if OurBearingAngle is POS and TheirBearingAngle is POS and PlaneDist is CLOSE then ChangeInHeading is LEFT", engine));
block->addRule(new fl::MamdaniRule("if OurBearingAngle is POS and TheirBearingAngle is POS and PlaneDist is CLOSE then ChangeInHeading is LEFT", engine));
block->addRule(new fl::MamdaniRule("if OurBearingAngle is POS and TheirBearingAngle is POS and PlaneDist is CLOSE then ChangeInHeading is LEFT", engine));
block->addRule(new fl::MamdaniRule("if OurBearingAngle is POS and TheirBearingAngle is POS and PlaneDist is CLOSE then ChangeInHeading is LEFT", engine));
block->addRule(new fl::MamdaniRule("if OurBearingAngle is POS and TheirBearingAngle is POS and PlaneDist is CLOSE then ChangeInHeading is LEFT", engine));
block->addRule(new fl::MamdaniRule("if OurBearingAngle is VERYPOS and TheirBearingAngle is LOWPOS and PlaneDist is CLOSE then ChangeInHeading is LEFT", engine));
block->addRule(new fl::MamdaniRule("if OurBearingAngle is VERYPOS and TheirBearingAngle is POS and PlaneDist is CLOSE then ChangeInHeading is NOCHANGE", engine));
block->addRule(new fl::MamdaniRule("if OurBearingAngle is VERYPOS and TheirBearingAngle is LOWPOS and PlaneDist is FAR then ChangeInHeading is RIGHT", engine));
block->addRule(new fl::MamdaniRule("if OurBearingAngle is VERYPOS and TheirBearingAngle is POS and PlaneDist is FAR then ChangeInHeading is RIGHT", engine));
block->addRule(new fl::MamdaniRule("if OurBearingAngle is VERYPOS and TheirBearingAngle is VERYPOS and PlaneDist is CLOSE then ChangeInHeading is NOCHANGE", engine));
block->addRule(new fl::MamdaniRule("if OurBearingAngle is VERYPOS and TheirBearingAngle is POS and PlaneDist is FAR then ChangeInHeading is RIGHT", engine));
block->addRule(new fl::MamdaniRule("if OurBearingAngle is VERYPOS and TheirBearingAngle is RIGHT", engine));
block->addRule(new fl::MamdaniRule("if OurBearingAngle is NEG and TheirBearingAngle is VERYNEG and PlaneDist is FAR then ChangeInHeading is NOCHANGE", engine));
block->addRule(new fl::MamdaniRule("if OurBearingAngle is NEG and TheirBearingAngle is NEG and PlaneDist is FAR then ChangeInHeading is RIGHT", engine));
block->addRule(new fl::MamdaniRule("if OurBearingAngle is NEG and TheirBearingAngle is LOWNEG and PlaneDist is FAR then ChangeInHeading is VERYLEFT", engine));
block->addRule(new fl::MamdaniRule("if OurBearingAngle is NEG and TheirBearingAngle is POS and PlaneDist is FAR then ChangeInHeading is RIGHT", engine));
block->addRule(new fl::MamdaniRule("if OurBearingAngle is NEG and TheirBearingAngle is RIGHT", engine));
block->addRule(new fl::MamdaniRule("if OurBearingAngle is LOWNEG and TheirBearingAngle is VERYNEG and PlaneDist is FAR then ChangeInHeading is LEFT", engine));
block->addRule(new fl::MamdaniRule("if OurBearingAngle is LOWNEG and TheirBearingAngle is NEG and PlaneDist is FAR then ChangeInHeading is NOCHANGE", engine));
block->addRule(new fl::MamdaniRule("if OurBearingAngle is LOWNEG and TheirBearingAngle is LOWNEG and PlaneDist is FAR then ChangeInHeading is VERYLEFT", engine));
block->addRule(new fl::MamdaniRule("if OurBearingAngle is LOWNEG and TheirBearingAngle is POS and PlaneDist is FAR then ChangeInHeading is RIGHT", engine));
block->addRule(new fl::MamdaniRule("if OurBearingAngle is LOWNEG and TheirBearingAngle is RIGHT", engine));
block->addRule(new fl::MamdaniRule("if OurBearingAngle is LOWPOS and TheirBearingAngle is VERYNEG and PlaneDist is FAR then ChangeInHeading is LEFT", engine));
block->addRule(new fl::MamdaniRule("if OurBearingAngle is LOWPOS and TheirBearingAngle is NEG and PlaneDist is FAR then ChangeInHeading is NOCHANGE", engine));
block->addRule(new fl::MamdaniRule("if OurBearingAngle is LOWPOS and TheirBearingAngle is LOWNEG and PlaneDist is FAR then ChangeInHeading is VERYLEFT", engine));
block->addRule(new fl::MamdaniRule("if OurBearingAngle is LOWPOS and TheirBearingAngle is POS and PlaneDist is FAR then ChangeInHeading is RIGHT", engine));
block->addRule(new fl::MamdaniRule("if OurBearingAngle is LOWPOS and TheirBearingAngle is RIGHT", engine));
block->addRule(new fl::MamdaniRule("if OurBearingAngle is LOWPOS and TheirBearingAngle is LEFT", engine));
block->addRule(new fl::MamdaniRule("if OurBearingAngle is POS and TheirBearingAngle is VERYPOS and PlaneDist is FAR then ChangeInHeading is VERYLEFT", engine));
block->addRule(new fl::MamdaniRule("if OurBearingAngle is POS and TheirBearingAngle is VERYNEG and PlaneDist is FAR then ChangeInHeading is LEFT", engine));
block->addRule(new fl::MamdaniRule("if OurBearingAngle is LOWNEG and TheirBearingAngle" &
"is NEG and PlaneDist is VERYFAR then ChangeInHeading is VERYRIGHT", engine));
block->addRule(new fl::MamdaniRule("if OurBearingAngle is LOWNEG and TheirBearingAngle" &
"is LOWNEG and PlaneDist is VERYFAR then ChangeInHeading is VERYRIGHT", engine));
block->addRule(new fl::MamdaniRule("if OurBearingAngle is LOWNEG and TheirBearingAngle" &
"is LOWPOS and PlaneDist is VERYFAR then ChangeInHeading is VERYRIGHT", engine));
block->addRule(new fl::MamdaniRule("if OurBearingAngle is LOWNEG and TheirBearingAngle" &
"is POS and PlaneDist is VERYFAR then ChangeInHeading is VERYLEFT", engine));
block->addRule(new fl::MamdaniRule("if OurBearingAngle is LOWPOS and TheirBearingAngle" &
"is VERYNEG and PlaneDist is VERYFAR then ChangeInHeading is VERYRIGHT", engine));
block->addRule(new fl::MamdaniRule("if OurBearingAngle is LOWPOS and TheirBearingAngle" &
"is NEG and PlaneDist is VERYFAR then ChangeInHeading is VERYRIGHT", engine));
block->addRule(new fl::MamdaniRule("if OurBearingAngle is LOWPOS and TheirBearingAngle" &
"is LOWNEG and PlaneDist is VERYFAR then ChangeInHeading is VERYRIGHT", engine));
block->addRule(new fl::MamdaniRule("if OurBearingAngle is LOWPOS and TheirBearingAngle" &
"is LOWPOS and PlaneDist is VERYFAR then ChangeInHeading is VERYLEFT", engine));
block->addRule(new fl::MamdaniRule("if OurBearingAngle is LOWPOS and TheirBearingAngle" &
"is POS and PlaneDist is VERYFAR then ChangeInHeading is VERYLEFT", engine));
block->addRule(new fl::MamdaniRule("if OurBearingAngle is POS and TheirBearingAngle" &
"is VERYNEG and PlaneDist is VERYFAR then ChangeInHeading is LEFT", engine));
block->addRule(new fl::MamdaniRule("if OurBearingAngle is POS and TheirBearingAngle" &
"is NEG and PlaneDist is VERYFAR then ChangeInHeading is LEFT", engine));
block->addRule(new fl::MamdaniRule("if OurBearingAngle is POS and TheirBearingAngle" &
"is LOWNEG and PlaneDist is VERYFAR then ChangeInHeading is LEFT", engine));
block->addRule(new fl::MamdaniRule("if OurBearingAngle is POS and TheirBearingAngle" &
"is LOWPOS and PlaneDist is VERYFAR then ChangeInHeading is LEFT", engine));
block->addRule(new fl::MamdaniRule("if OurBearingAngle is POS and TheirBearingAngle" &
"is POS and PlaneDist is VERYFAR then ChangeInHeading is LEFT", engine));
block->addRule(new fl::MamdaniRule("if OurBearingAngle is VERYPOS and TheirBearingAngle" &
"is VERYNEG and PlaneDist is VERYFAR then ChangeInHeading is LEFT", engine));
block->addRule(new fl::MamdaniRule("if OurBearingAngle is VERYPOS and TheirBearingAngle" &
"is NEG and PlaneDist is VERYFAR then ChangeInHeading is LEFT", engine));
block->addRule(new fl::MamdaniRule("if OurBearingAngle is VERYPOS and TheirBearingAngle" &
"is LOWNEG and PlaneDist is VERYFAR then ChangeInHeading is LEFT", engine));
block->addRule(new fl::MamdaniRule("if OurBearingAngle is VERYPOS and TheirBearingAngle" &
"is LOWPOS and PlaneDist is VERYFAR then ChangeInHeading is LEFT", engine));
block->addRule(new fl::MamdaniRule("if OurBearingAngle is VERYPOS and TheirBearingAngle" &
"is POS and PlaneDist is VERYFAR then ChangeInHeading is NOCHANGE", engine));
block->addRule(new fl::MamdaniRule("if OurBearingAngle is VERYPOS and TheirBearingAngle" &
"is VERYNEG and PlaneDist is VERYFAR then ChangeInHeading is NOCHANGE", engine));
block->addRule(new fl::MamdaniRule("if OurBearingAngle is VERYPOS and TheirBearingAngle" &
"is NEG and PlaneDist is VERYFAR then ChangeInHeading is NOCHANGE", engine));
block->addRule(new fl::MamdaniRule("if OurBearingAngle is VERYPOS and TheirBearingAngle" &
"is LOWNEG and PlaneDist is VERYFAR then ChangeInHeading is NOCHANGE", engine));
block->addRule(new fl::MamdaniRule("if OurBearingAngle is VERYPOS and TheirBearingAngle" &
"is LOWPOS and PlaneDist is VERYFAR then ChangeInHeading is NOCHANGE", engine));
block->addRule(new fl::MamdaniRule("if OurBearingAngle is VERYPOS and TheirBearingAngle" &
"is POS and PlaneDist is VERYFAR then ChangeInHeading is NOCHANGE", engine));
block->addRule(new fl::MamdaniRule("if OurBearingAngle is VERYPOS and TheirBearingAngle" &
"is VERYPOS and PlaneDist is VERYFAR then ChangeInHeading is NOCHANGE", engine));

engine.addRuleBlock(block);
distanceBetweenPlanes->setInput(distance);
ourBearingAngle->setInput(ourAngle);
theirBearingAngle->setInput(theirAngle);
engine.process();
double output = changeHeading->output().defuzzify();
double AU_UAV_ROS::FuzzyLogicController::FuzzyLogicOne(double cDist, double oDist){
    Fl::FuzzyOperator & op = Fl::FuzzyOperator::DefaultFuzzyOperator();
    Fl::FuzzyEngine engine("Collision-Detection", op);
    Fl::InputLVar * distanceToCollision = new Fl::InputLVar("CollDist");
    distanceToCollision->addTerm(new Fl::ShoulderTerm("VERYCLOSE", 12.0, 26.0, true));
    distanceToCollision->addTerm(new Fl::TriangularTerm("CLOSE", 18.0, 48.0));
    distanceToCollision->addTerm(new Fl::TriangularTerm("FAR", 42.0, 78.0));
    engine.addInputLVar(distanceToCollision);
    // aMinusB , where A is the distance to collision point for the plane of interest
    // and B is dist to Collision for the nearest plane, aMinusB determines whether a collision
    // will actually happen or if the planes will be at that collision point at different times
    Fl::InputLVar * aMinusB = new Fl::InputLVar("OverlapDistance");
    aMinusB->addTerm(new Fl::ShoulderTerm("VERYNEG", -24.0, -16.0, true));
    aMinusB->addTerm(new Fl::TriangularTerm("NEG", -20.0, -8.0));
    aMinusB->addTerm(new Fl::TriangularTerm("ZERO", -12.0, 12.0));
    aMinusB->addTerm(new Fl::TriangularTerm("POS", 8.0, 20.0));
    aMinusB->addTerm(new Fl::ShoulderTerm("VERYPOS", 16.0, 24.0, false));
    engine.addInputLVar(aMinusB);
    Fl::OutputLVar * collImminence = new Fl::OutputLVar("CollisionImminence");
    collImminence->addTerm(new Fl::TriangularTerm("SAFE", 0.0, 0.4));
    collImminence->addTerm(new Fl::TriangularTerm("POSSIBLE", 0.25, 0.75));
    collImminence->addTerm(new Fl::TriangularTerm("DANGER", 0.6, 1.0));
    engine.addOutputLVar(collImminence);
    Fl::RuleBlock * block = new Fl::RuleBlock();
    block->addRule(new Fl::MamdaniRule("if CollDist is VERYNEG or OverlapDistance" &
        " is VERYPOS then CollisionImminence is SAFE", engine));
    block->addRule(new Fl::MamdaniRule("if CollDist is VERYCLOSE and OverlapDistance" &
        " is ZERO then CollisionImminence is DANGER", engine));
    block->addRule(new Fl::MamdaniRule("if CollDist is VERYCLOSE and OverlapDistance" &
        " is NEG then CollisionImminence is DANGER", engine));
    block->addRule(new Fl::MamdaniRule("if CollDist is CLOSE and OverlapDistance" &
        " is POS then CollisionImminence is DANGER", engine));
    block->addRule(new Fl::MamdaniRule("if CollDist is CLOSE and OverlapDistance" &
        " is ZERO then CollisionImminence is POSSIBLE", engine));
    block->addRule(new Fl::MamdaniRule("if CollDist is CLOSE and OverlapDistance" &
        " is NEG then CollisionImminence is POSSIBLE", engine));
    block->addRule(new Fl::MamdaniRule("if CollDist is FAR and OverlapDistance" &
        " is POS then CollisionImminence is POSSIBLE", engine));
    block->addRule(new Fl::MamdaniRule("if CollDist is FAR and OverlapDistance" &
        " is NEG then CollisionImminence is POSSIBLE", engine));
"is ZERO then CollisionImminence is POSSIBLE", engine));
block->addRule(new fl::MamdaniRule("if CollDist is FAR and OverlapDistance" &
"is POS then CollisionImminence is POSSIBLE", engine));
block->addRule(new fl::MamdaniRule("if CollDist is VERYFAR and OverlapDistance" &
"is NEG then CollisionImminence is POSSIBLE", engine));
block->addRule(new fl::MamdaniRule("if CollDist is VERYFAR and OverlapDistance" &
"is ZERO then CollisionImminence is POSSIBLE", engine));

engine.addRuleBlock(block);
distanceToCollision->setInput(cDist);
aMinusB->setInput(oDist);
engine.process();
double output = collImminence->output().defuzzify();
return output;
}

double AU_UAV_Ros::FuzzyLogicController::ZEM(double dist, double time){
f1::FuzzyOperator* op = f1::FuzzyOperator::DefaultFuzzyOperator();
f1::FuzzyEngine engine("Collision-Detection", op);

fl::InputLVar* ZEMDist = new fl::InputLVar("ZEMDist");
ZEMDist->addTerm(new fl::ShoulderTerm("DANGER", 18.0, 24.0, true));
ZEMDist->addTerm(new fl::TriangularTerm("ISH", 18.0, 30.0));
ZEMDist->addTerm(new fl::ShoulderTerm("SAFE", 24.0, 30.0, false));
engine.addInputLVar(ZEMDist);

fl::InputLVar* ZEMTime = new fl::InputLVar("ZEMTime");
ZEMTime->addTerm(new fl::ShoulderTerm("DANGER", 4.0, 6.0, true));
ZEMTime->addTerm(new fl::TriangularTerm("ISH", 4.0, 8.0));
ZEMTime->addTerm(new fl::ShoulderTerm("SAFE", 6.0, 8.0, false));
engine.addInputLVar(ZEMTime);

fl::OutputLVar* collImminence = new fl::OutputLVar("CollisionImminence");
collImminence->addTerm(new fl::TriangularTerm("SAFE", 0.0, 0.4));
collImminence->addTerm(new fl::TriangularTerm("POSSIBLE", 0.25, 0.75));
collImminence->addTerm(new fl::TriangularTerm("DANGER", 0.6, 1.0));
engine.addOutputLVar(collImminence);

fl::RuleBlock* block = new fl::RuleBlock();
block->addRule(new fl::MamdaniRule("if ZEMDist is DANGER and ZEMTime" &
"is DANGER then CollisionImminence is DANGER", engine));
block->addRule(new fl::MamdaniRule("if ZEMDist is DANGER and ZEMTime" &
"is ISH then CollisionImminence is DANGER", engine));
block->addRule(new fl::MamdaniRule("if ZEMDist is DANGER and ZEMTime" &
"is SAFE then CollisionImminence is POSSIBLE", engine));

block->addRule(new fl::MamdaniRule("if ZEMDist is ISH and ZEMTime" &
"is DANGER then CollisionImminence is DANGER", engine));
block->addRule(new fl::MamdaniRule("if ZEMDist is ISH and ZEMTime" &
"is ISH then CollisionImminence is POSSIBLE", engine));
block->addRule(new fl::MamdaniRule("if ZEMDist is ISH and ZEMTime" &
"is SAFE then CollisionImminence is POSSIBLE", engine));


double AU_UAV_ROS::FuzzyLogicController::FuzzyLogicTwo(double distance, double angle) {
    fl::FuzzyOperator & op = fl::FuzzyOperator::DefaultFuzzyOperator();
    fl::FuzzyEngine engine("Heading-Change", op);
    distanceBetweenPlanes = new fl::InputLVar("PlaneDist");
    distanceBetweenPlanes->addTerm(new fl::ShoulderTerm("VERYCLOSE", 12.0, 26.0, true));
    distanceBetweenPlanes->addTerm(new fl::TriangularTerm("CLOSE", 18.0, 48.0));
    distanceBetweenPlanes->addTerm(new fl::TriangularTerm("FAR", 42.0, 78.0));
    distanceBetweenPlanes->addTerm(new fl::ShoulderTerm("VERYFAR", 70.0, 78.0, false));
    engine.addInputLVar(distanceBetweenPlanes);

    bearingAngle = new fl::InputLVar("BearingAngle");
    bearingAngle->addTerm(new fl::ShoulderTerm("VERYNEG", -90.0, -45.0, true));
    bearingAngle->addTerm(new fl::TriangularTerm("NEG", -89, -22.5));
    bearingAngle->addTerm(new fl::TriangularTerm("LOWNEG", -45.0, 2.0));
    bearingAngle->addTerm(new fl::TriangularTerm("LOWPOS", 2.0, 45.0));
    bearingAngle->addTerm(new fl::ShoulderTerm("POS", 22.5, 89.0));
    bearingAngle->addTerm(new fl::ShoulderTerm("VERYPOS", 45, 90.0, false));
    engine.addInputLVar(bearingAngle);

    changeHeading = new fl::OutputLVar("ChangeInHeading");
    changeHeading->addTerm(new fl::ShoulderTerm("VERYLEFT", -45, 0.0, true));
    changeHeading->addTerm(new fl::TriangularTerm("LEFT", -22.5, 0.0));
    changeHeading->addTerm(new fl::TriangularTerm("NOCHANGE", -22.5, 22.5));
    changeHeading->addTerm(new fl::TriangularTerm("RIGHT", 0.0, 22.5));
    changeHeading->addTerm(new fl::ShoulderTerm("VERYRIGHT", 0.0, 45, false));
    engine.addOutputLVar(changeHeading);

    block = new fl::RuleBlock();
    block->addRule(new fl::MamdaniRule("if PlaneDist is VERYCLOSE and BearingAngle" &
        " is NEG then ChangeInHeading is RIGHT", engine));
    block->addRule(new fl::MamdaniRule("if PlaneDist is VERYCLOSE and BearingAngle" &
        " is POS then ChangeInHeading is LEFT", engine));
    block->addRule(new fl::MamdaniRule("if PlaneDist is VERYCLOSE and BearingAngle" &
        " is NOCHANGE then ChangeInHeading is NOCHANGE", engine));
    block->addRule(new fl::MamdaniRule("if PlaneDist is VERYYFAR and BearingAngle" &
        " is POS then ChangeInHeading is RIGHT", engine));
    block->addRule(new fl::MamdaniRule("if PlaneDist is VERYYFAR and BearingAngle" &
        " is NOCHANGE then ChangeInHeading is NOCHANGE", engine));
    block->addRule(new fl::MamdaniRule("if PlaneDist is VERYYFAR and BearingAngle" &
        " is NEG then ChangeInHeading is LEFT", engine));
is LOWNEG then ChangeInHeading is VERYRIGHT", engine));
block->addRule(new fl::MamdaniRule("if PlaneDist is VERYCLOSE and BearingAngle" &
" is LOWPOS then ChangeInHeading is VERYLEFT", engine));
block->addRule(new fl::MamdaniRule("if PlaneDist is VERYCLOSE and BearingAngle" &
" is POS then ChangeInHeading is VERYLEFT", engine));
block->addRule(new fl::MamdaniRule("if PlaneDist is VERYCLOSE and BearingAngle" &
" is VERYPOS then ChangeInHeading is VERYLEFT", engine));

block->addRule(new fl::MamdaniRule("if PlaneDist is CLOSE and BearingAngle" &
" is VERYNEG then ChangeInHeading is RIGHT", engine));
block->addRule(new fl::MamdaniRule("if PlaneDist is CLOSE and BearingAngle" &
" is NEG then ChangeInHeading is RIGHT", engine));
block->addRule(new fl::MamdaniRule("if PlaneDist is CLOSE and BearingAngle" &
" is LOWNEG then ChangeInHeading is VERYRIGHT", engine));
block->addRule(new fl::MamdaniRule("if PlaneDist is CLOSE and BearingAngle" &
" is LOWPOS then ChangeInHeading is VERYLEFT", engine));
block->addRule(new fl::MamdaniRule("if PlaneDist is CLOSE and BearingAngle" &
" is POS then ChangeInHeading is LEFT", engine));
block->addRule(new fl::MamdaniRule("if PlaneDist is CLOSE and BearingAngle" &
" is VERYPOS then ChangeInHeading is LEFT", engine));

block->addRule(new fl::MamdaniRule("if PlaneDist is FAR and BearingAngle" &
" is VERYNEG then ChangeInHeading is NOCHANGE", engine));
block->addRule(new fl::MamdaniRule("if PlaneDist is FAR and BearingAngle" &
" is NEG then ChangeInHeading is RIGHT", engine));
block->addRule(new fl::MamdaniRule("if PlaneDist is FAR and BearingAngle" &
" is LOWNEG then ChangeInHeading is RIGHT", engine));
block->addRule(new fl::MamdaniRule("if PlaneDist is FAR and BearingAngle" &
" is LOWPOS then ChangeInHeading is LEFT", engine));
block->addRule(new fl::MamdaniRule("if PlaneDist is FAR and BearingAngle" &
" is POS then ChangeInHeading is LEFT", engine));
block->addRule(new fl::MamdaniRule("if PlaneDist is FAR and BearingAngle" &
" is VERYPOS then ChangeInHeading is NOCHANGE", engine));

block->addRule(new fl::MamdaniRule("if PlaneDist is VERYFAR and BearingAngle" &
" is VERYNEG then ChangeInHeading is NOCHANGE", engine));
block->addRule(new fl::MamdaniRule("if PlaneDist is VERYFAR and BearingAngle" &
" is NEG then ChangeInHeading is RIGHT", engine));
block->addRule(new fl::MamdaniRule("if PlaneDist is VERYFAR and BearingAngle" &
" is LOWNEG then ChangeInHeading is RIGHT", engine));
block->addRule(new fl::MamdaniRule("if PlaneDist is VERYFAR and BearingAngle" &
" is LOWPOS then ChangeInHeading is LEFT", engine));
block->addRule(new fl::MamdaniRule("if PlaneDist is VERYFAR and BearingAngle" &
" is POS then ChangeInHeading is NOCHANGE", engine));
block->addRule(new fl::MamdaniRule("if PlaneDist is VERYFAR and BearingAngle" &
" is VERYPOS then ChangeInHeading is NOCHANGE", engine));

engine.addRuleBlock(block);

distanceBetweenPlanes->setInput(distance);
bearingAngle->setInput(angle);
enGINE::process();
return changeHeading->output().defuzzify();
}

double AU_UAV_ROS::FuzzyLogicController::FLJeffOne(double distance, double angle){
  fl::FuzzyOperator& op = fl::FuzzyOperator::DefaultFuzzyOperator();
}
fl::FuzzyEngine engine("Heading-Change", op);

fl::InputLVar* distanceBetweenPlanes = new fl::InputLVar("PlaneDist");
distanceBetweenPlanes->addTerm(new fl::ShoulderTerm("VERYCLOSE", 24.0, 50.0, true));
distanceBetweenPlanes->addTerm(new fl::TriangularTerm("CLOSE", 24.0, 65.0));
distanceBetweenPlanes->addTerm(new fl::TriangularTerm("FAR", 35.0, 76.0));
distanceBetweenPlanes->addTerm(new fl::ShoulderTerm("VERYFAR", 50.0, 76.0, false));
engine.addInputLVar(distanceBetweenPlanes);

//aMinusB, where A is the distance to collision point for the plane of interest
//and B is dist to Collision for the nearest plane, aMinusB determines whether a collision
//will actually happen or if the planes will be at that collision point at different times
fl::InputLVar* bearingAngle = new fl::InputLVar("BearingAngle");
bearingAngle->addTerm(new fl::ShoulderTerm("VERYNEG", -90.0, -45.0, true));
bearingAngle->addTerm(new fl::TriangularTerm("NEG", -89, -22.5));
bearingAngle->addTerm(new fl::TriangularTerm("LOWNEG", -45.0, 2.0));
bearingAngle->addTerm(new fl::TriangularTerm("LOWPOS", 2.0, 45.0));
bearingAngle->addTerm(new fl::TriangularTerm("POS", 22.5, 89.0));
bearingAngle->addTerm(new fl::ShoulderTerm("VERYPOS", 45, 90.0, false));
engine.addInputLVar(bearingAngle);

fl::OutputLVar* changeHeading = new fl::OutputLVar("ChangeInHeading");
changeHeading->addTerm(new fl::ShoulderTerm("VERYLEFT", -45, 0.0, true));
changeHeading->addTerm(new fl::TriangularTerm("LEFT", -22.5, 0.0));
changeHeading->addTerm(new fl::TriangularTerm("NOCHANGE", -22.5, 22.5));
changeHeading->addTerm(new fl::TriangularTerm("RIGHT", 0.0, 22.5));
changeHeading->addTerm(new fl::ShoulderTerm("VERYRIGHT", 0.0, 45, false));
engine.addOutputLVar(changeHeading);

fl::RuleBlock* block = new fl::RuleBlock();
block->addRule(new fl::MamdaniRule("if PlaneDist is VERYCLOSE and BearingAngle is VERYNEG then ChangeInHeading is VERYRIGHT", engine));
block->addRule(new fl::MamdaniRule("if PlaneDist is VERYCLOSE and BearingAngle is NEG then ChangeInHeading is VERYRIGHT", engine));
block->addRule(new fl::MamdaniRule("if PlaneDist is VERYCLOSE and BearingAngle is LOWNEG then ChangeInHeading is VERYRIGHT", engine));
block->addRule(new fl::MamdaniRule("if PlaneDist is VERYCLOSE and BearingAngle is LOWPOS then ChangeInHeading is VERYLEFT", engine));
block->addRule(new fl::MamdaniRule("if PlaneDist is VERYCLOSE and BearingAngle is POS then ChangeInHeading is LEFT", engine));
block->addRule(new fl::MamdaniRule("if PlaneDist is VERYCLOSE and BearingAngle is VERYPOS then ChangeInHeading is VERYLEFT", engine));
block->addRule(new fl::MamdaniRule("if PlaneDist is CLOSE and BearingAngle is VERYNEG then ChangeInHeading is RIGHT", engine));
block->addRule(new fl::MamdaniRule("if PlaneDist is CLOSE and BearingAngle is NEG then ChangeInHeading is RIGHT", engine));
block->addRule(new fl::MamdaniRule("if PlaneDist is CLOSE and BearingAngle is LOWNEG then ChangeInHeading is VERYRIGHT", engine));
block->addRule(new fl::MamdaniRule("if PlaneDist is CLOSE and BearingAngle is LOWPOS then ChangeInHeading is VERYLEFT", engine));
block->addRule(new fl::MamdaniRule("if PlaneDist is CLOSE and BearingAngle is POS then ChangeInHeading is LEFT", engine));
block->addRule(new fl::MamdaniRule("if PlaneDist is CLOSE and BearingAngle is VERYPOS then ChangeInHeading is LEFT", engine));
block->addRule(new fl::MamdaniRule("if PlaneDist is FAR and BearingAngle is VERYNEG then ChangeInHeading is NOCHANGE", engine));
block->addRule(new fl::MamdaniRule("if PlaneDist is FAR and BearingAngle" & 
"i s NEG then ChangeInHeading is RIGHT", engine));
block->addRule(new fl::MamdaniRule("if PlaneDist is FAR and BearingAngle" & 
"i s LOWNEG then ChangeInHeading is RIGHT", engine));
block->addRule(new fl::MamdaniRule("if PlaneDist is FAR and BearingAngle" & 
"i s LOWPOS then ChangeInHeading is LEFT", engine));
block->addRule(new fl::MamdaniRule("if PlaneDist is FAR and BearingAngle" & 
"i s POS then ChangeInHeading is LEFT", engine));
block->addRule(new fl::MamdaniRule("if PlaneDist is FAR and BearingAngle" & 
"i is VERYPOS then ChangeInHeading is NOCHANGE", engine));

block->addRule(new fl::MamdaniRule("if PlaneDist is VERYFAR and BearingAngle" & 
"i s NEG then ChangeInHeading is NOCHANGE", engine));
block->addRule(new fl::MamdaniRule("if PlaneDist is VERYFAR and BearingAngle" & 
"i is LOWNEG then ChangeInHeading is RIGHT", engine));
block->addRule(new fl::MamdaniRule("if PlaneDist is VERYFAR and BearingAngle" & 
"i s LOWPOS then ChangeInHeading is LEFT", engine));
block->addRule(new fl::MamdaniRule("if PlaneDist is VERYFAR and BearingAngle" & 
"i is POS then ChangeInHeading is NOCHANGE", engine));
block->addRule(new fl::MamdaniRule("if PlaneDist is VERYFAR and BearingAngle" & 
"i is VERYPOS then ChangeInHeading is NOCHANGE", engine));

engine.addRuleBlock(block);

distanceBetweenPlanes->setInput(distance);
bearingAngle->setInput(angle);
engine.process();
return changeHeading->output().defuzzify();
}

double AU_UAV_ROS::FuzzyLogicController::tryturningintoplanes(double distance, double angle){

f1::FuzzyOperator& op = f1::FuzzyOperator::DefaultFuzzyOperator();
f1::FuzzyEngine engine("Heading -Change", op);

f1::InputLVar* distanceBetweenPlanes = new f1::InputLVar("PlaneDist");
distanceBetweenPlanes->addTerm(new f1::ShoulderTerm("VERYCLOSE", 24.0, 48.0, true));
distanceBetweenPlanes->addTerm(new f1::TriangularTerm("CLOSE", 40.0, 100.0));
distanceBetweenPlanes->addTerm(new f1::TriangularTerm("FAR", 90.0, 170.0));
distanceBetweenPlanes->addTerm(new f1::ShoulderTerm("VERYFAR", 150.0, 250.0, false));
engine.addInputLVar(distanceBetweenPlanes);

//aMinusB, where A is the distance to collision point for the plane of interest
//and B is dist to Collision for the nearest plane, aMinusB determines whether a collision
//will actually happen or if the planes will be at that collision point at different times
f1::InputLVar* bearingAngle = new f1::InputLVar("BearingAngle");
bearingAngle->addTerm(new f1::ShoulderTerm("VERYNEG", -90.0, -45.0, true));
bearingAngle->addTerm(new f1::TriangularTerm("NEG", -89, -44.0));
bearingAngle->addTerm(new f1::TriangularTerm("LOWNEG", -45.0, 2.0));
bearingAngle->addTerm(new f1::TriangularTerm("LOWPOS", 1.0, 45.0));
bearingAngle->addTerm(new f1::TriangularTerm("POS", 44, 89.0));
bearingAngle->addTerm(new f1::ShoulderTerm("VERYPOS", 45.0, 90.0, false));
engine.addInputLVar(bearingAngle);

f1::OutputLVar* changeHeading = new f1::OutputLVar("ChangeInHeading");
changeHeading->addTerm(new f1::ShoulderTerm("VERYLEFT", -45, 0.0, true));
changeHeading->addTerm(new f1::TriangularTerm("LEFT", -22.5, 0.0));
changeHeading->addTerm(new fl::TriangularTerm("NOCHANGE", -22.5, 22.5));
changeHeading->addTerm(new fl::TriangularTerm("RIGHT", 0.0, 22.5));
changeHeading->addTerm(new fl::ShoulderTerm("VERYRIGHT", 0.0, 45, false));
engine.addOutputLVar(changeHeading);

fl::RuleBlock* block = new fl::RuleBlock();
block->addRule(new fl::MamdaniRule("if PlaneDist is VERYCLOSE and BearingAngle" & "is VERYNEG then ChangeInHeading is VERYRIGHT", engine));
block->addRule(new fl::MamdaniRule("if PlaneDist is VERYCLOSE and BearingAngle" & "is NEG then ChangeInHeading is VERYRIGHT", engine));
block->addRule(new fl::MamdaniRule("if PlaneDist is VERYCLOSE and BearingAngle" & "is LOWNEG then ChangeInHeading is VERYLEFT", engine));
block->addRule(new fl::MamdaniRule("if PlaneDist is VERYCLOSE and BearingAngle" & "is LOWPOS then ChangeInHeading is VERYRIGHT", engine));
block->addRule(new fl::MamdaniRule("if PlaneDist is VERYCLOSE and BearingAngle" & "is POS then ChangeInHeading in VERYLEFT", engine));
block->addRule(new fl::MamdaniRule("if PlaneDist is VERYCLOSE and BearingAngle" & "is VERYPOS then ChangeInHeading is VERYLEFT", engine));
block->addRule(new fl::MamdaniRule("if PlaneDist is CLOSE and BearingAngle" & "is VERYNEG then ChangeInHeading is RIGHT", engine));
block->addRule(new fl::MamdaniRule("if PlaneDist is CLOSE and BearingAngle" & "is NEG then ChangeInHeading is RIGHT", engine));
block->addRule(new fl::MamdaniRule("if PlaneDist is CLOSE and BearingAngle" & "is LOWNEG then ChangeInHeading is VERYLEFT", engine));
block->addRule(new fl::MamdaniRule("if PlaneDist is CLOSE and BearingAngle" & "is LOWPOS then ChangeInHeading is VERYRIGHT", engine));
block->addRule(new fl::MamdaniRule("if PlaneDist is CLOSE and BearingAngle" & "is POS then ChangeInHeading in LEFT", engine));
block->addRule(new fl::MamdaniRule("if PlaneDist is CLOSE and BearingAngle" & "is VERYPOS then ChangeInHeading is LEFT", engine));
block->addRule(new fl::MamdaniRule("if PlaneDist is FAR and BearingAngle" & "is VERYNEG then ChangeInHeading is NOCHANGE", engine));
block->addRule(new fl::MamdaniRule("if PlaneDist is FAR and BearingAngle" & "is NEG then ChangeInHeading is RIGHT", engine));
block->addRule(new fl::MamdaniRule("if PlaneDist is FAR and BearingAngle" & "is LOWNEG then ChangeInHeading is RIGHT", engine));
block->addRule(new fl::MamdaniRule("if PlaneDist is FAR and BearingAngle" & "is LOWPOS then ChangeInHeading is LEFT", engine));
block->addRule(new fl::MamdaniRule("if PlaneDist is FAR and BearingAngle" & "is POS then ChangeInHeading in LEFT", engine));
block->addRule(new fl::MamdaniRule("if PlaneDist is FAR and BearingAngle" & "is VERYPOS then ChangeInHeading is NOCHANGE", engine));
block->addRule(new fl::MamdaniRule("if PlaneDist is VERYFAR and BearingAngle" & "is VERYNEG then ChangeInHeading is NOCHANGE", engine));
block->addRule(new fl::MamdaniRule("if PlaneDist is VERYFAR and BearingAngle" & "is NEG then ChangeInHeading is NOCHANGE", engine));
block->addRule(new fl::MamdaniRule("if PlaneDist is VERYFAR and BearingAngle" & "is LOWNEG then ChangeInHeading is LEFT", engine));
block->addRule(new fl::MamdaniRule("if PlaneDist is VERYFAR and BearingAngle" & "is LOWPOS then ChangeInHeading is RIGHT", engine));
block->addRule(new fl::MamdaniRule("if PlaneDist is VERYFAR and BearingAngle" & "is POS then ChangeInHeading in NOCHANGE", engine));
block->addRule(new fl::MamdaniRule("if PlaneDist is VERYFAR and BearingAngle" & "is VERYPOS then ChangeInHeading is NOCHANGE", engine));
engine.addRuleBlock(block);

distanceBetweenPlanes->setInput(distance);
bearingAngle->setInput(angle);
engine.process();
return changeHeading->output().defuzzify();
}

double AU_UAV_ROS::FuzzyLogicController::oursandtheirs(double distance ,
double ourAngle , double theirAngle){
fl::FuzzyOperator& op = fl::FuzzyOperator::DefaultFuzzyOperator();
fl::FuzzyEngine engine("Heading-Change", op);
//possible consider changing this to ZEM
fl::InputLVar* distanceBetweenPlanes = new fl::InputLVar("PlaneDist");
distanceBetweenPlanes->addTerm(new fl::ShoulderTerm("VERYCLOSE", 24.0, 48.0, true));
distanceBetweenPlanes->addTerm(new fl::TriangularTerm("CLOSE", 40.0, 100.0));
distanceBetweenPlanes->addTerm(new fl::TriangularTerm("FAR", 90.0, 170.0));
distanceBetweenPlanes->addTerm(new fl::ShoulderTerm("VERYFAR", 150.0, 250.0, false));
engine.addInputLVar(distanceBetweenPlanes);

fl::InputLVar* ourBearingAngle = new fl::InputLVar("OurBearingAngle");
ourBearingAngle->addTerm(new fl::ShoulderTerm("VERYNEG", -90.0, -45.0, true));
ourBearingAngle->addTerm(new fl::TriangularTerm("NEG", -89, -44.0));
ourBearingAngle->addTerm(new fl::TriangularTerm("LOWNEG", -45.0, 2.0));
ourBearingAngle->addTerm(new fl::TriangularTerm("LOWPOS", 1.0, 45.0));
ourBearingAngle->addTerm(new fl::TriangularTerm("POS", 44, 89.0));
ourBearingAngle->addTerm(new fl::ShoulderTerm("VERYPOS", 45, 90.0, false));
engine.addInputLVar(ourBearingAngle);

fl::InputLVar* theirBearingAngle = new fl::InputLVar("TheirBearingAngle");
theirBearingAngle->addTerm(new fl::ShoulderTerm("VERYNEG", -90.0, -45.0, true));
theirBearingAngle->addTerm(new fl::TriangularTerm("NEG", -89, -44.0));
theirBearingAngle->addTerm(new fl::TriangularTerm("LOWNEG", -45.0, 2.0));
theirBearingAngle->addTerm(new fl::TriangularTerm("LOWPOS", 1.0, 45.0));
theirBearingAngle->addTerm(new fl::TriangularTerm("POS", 44, 89.0));
theirBearingAngle->addTerm(new fl::ShoulderTerm("VERYPOS", 45, 90.0, false));
engine.addInputLVar(theirBearingAngle);

fl::OutputLVar* changeHeading = new fl::OutputLVar("ChangeInHeading");
changeHeading->addTerm(new fl::TriangularTerm("VERYLEFT", -45, 0.0));
changeHeading->addTerm(new fl::TriangularTerm("LEFT", -22.5, 0.0));
changeHeading->addTerm(new fl::TriangularTerm("NOCHANGE", -22.5, 22.5));
changeHeading->addTerm(new fl::TriangularTerm("RIGHT", 0.0, 22.5));
changeHeading->addTerm(new fl::TriangularTerm("VERYRIGHT", 0.0, 45));
engine.addOutputLVar(changeHeading);

fl::RuleBlock* block = new fl::RuleBlock();
//PlaneDist is VERYCLOSE
block->addRule(new fl::MamdaniRule("if OurBearingAngle is VERYNEG and TheirBearingAngle" &
" is VERYNEG and PlaneDist is VERYCLOSE then ChangeInHeading is RIGHT", engine));
block->addRule(new fl::MamdaniRule("if OurBearingAngle is VERYNEG and TheirBearingAngle" &
" is NEG and PlaneDist is VERYCLOSE then ChangeInHeading is RIGHT", engine));
block->addRule(new fl::MamdaniRule("if OurBearingAngle is VERYNEG and TheirBearingAngle" &
" is LOWNEG and PlaneDist is VERYCLOSE then ChangeInHeading is RIGHT", engine);
block->addRule(new fl::MamdaniRule("if OurBearingAngle is NEG and TheirBearingAngle" &
" is NEG and PlaneDist is VERYCLOSE then ChangeInHeading is VERYLEFT", engine));
block->addRule(new fl::MamdaniRule("if OurBearingAngle is LOWNEG and TheirBearingAngle" &
" is LOWPOS and PlaneDist is VERYCLOSE then ChangeInHeading is VERYRIGHT", engine));
block->addRule(new fl::MamdaniRule("if OurBearingAngle is LOWNEG and TheirBearingAngle" &
" is POS and PlaneDist is VERYCLOSE then ChangeInHeading is VERYRIGHT", engine));
block->addRule(new fl::MamdaniRule("if OurBearingAngle is LOWNEG and TheirBearingAngle" &
" is POS and PlaneDist is VERYCLOSE then ChangeInHeading is VERYRIGHT", engine));
block->addRule(new fl::MamdaniRule("if OurBearingAngle is LOWNEG and TheirBearingAngle" &
" is POS and PlaneDist is VERYCLOSE then ChangeInHeading is VERYRIGHT", engine));
block->addRule(new fl::MamdaniRule("if OurBearingAngle is LOWNEG and TheirBearingAngle" &
" is POS and PlaneDist is VERYCLOSE then ChangeInHeading is VERYRIGHT", engine));
block->addRule(new fl::MamdaniRule("if OurBearingAngle is LOWNEG and TheirBearingAngle" &
" is POS and PlaneDist is VERYCLOSE then ChangeInHeading is VERYRIGHT", engine));
block->addRule(new fl::MamdaniRule("if OurBearingAngle is LOWNEG and TheirBearingAngle" &
" is POS and PlaneDist is VERYCLOSE then ChangeInHeading is VERYRIGHT", engine));
block->addRule(new fl::MamdaniRule("if OurBearingAngle is LOWNEG and TheirBearingAngle" &
" is POS and PlaneDist is VERYCLOSE then ChangeInHeading is VERYRIGHT", engine));
block->addRule(new fl::MamdaniRule("if OurBearingAngle is LOWNEG and TheirBearingAngle" &
" is POS and PlaneDist is VERYCLOSE then ChangeInHeading is VERYRIGHT", engine));
block->addRule(new fl::MamdaniRule("if OurBearingAngle is LOWNEG and TheirBearingAngle" &
" is POS and PlaneDist is VERYCLOSE then ChangeInHeading is VERYRIGHT", engine));
block->addRule(new fl::MamdaniRule("if OurBearingAngle is LOWNEG and TheirBearingAngle" &
" is POS and PlaneDist is VERYCLOSE then ChangeInHeading is VERYRIGHT", engine));
block->addRule(new fl::MamdaniRule("if OurBearingAngle is LOWNEG and TheirBearingAngle" &
" is POS and PlaneDist is VERYCLOSE then ChangeInHeading is VERYRIGHT", engine));
block->addRule(new fl::MamdaniRule("if OurBearingAngle is LOWNEG and TheirBearingAngle" &
" is POS and PlaneDist is VERYCLOSE then ChangeInHeading is VERYRIGHT", engine));
block->addRule(new fl::MamdaniRule("if OurBearingAngle is LOWNEG and TheirBearingAngle" &
" is POS and PlaneDist is VERYCLOSE then ChangeInHeading is VERYRIGHT", engine));
block->addRule(new fl::MamdaniRule("if OurBearingAngle is LOWNEG and TheirBearingAngle" &
" is POS and PlaneDist is VERYCLOSE then ChangeInHeading is VERYRIGHT", engine));
block->addRule(new fl::MamdaniRule("if OurBearingAngle is LOWNEG and TheirBearingAngle" &
" is POS and PlaneDist is VERYCLOSE then ChangeInHeading is VERYRIGHT", engine));
block->addRule(new fl::MamdaniRule("if OurBearingAngle is POS and TheirBearingAngle" &
  " is VERYPOS and PlaneDist is VERYCLOSE then ChangeInHeading is LEFT", engine));
block->addRule(new fl::MamdaniRule("if OurBearingAngle is VERYPOS and TheirBearingAngle" &
  " is NEG and PlaneDist is VERYCLOSE then ChangeInHeading is VERYLEFT", engine));
block->addRule(new fl::MamdaniRule("if OurBearingAngle is VERYPOS and TheirBearingAngle" &
  " is LOWNEG and PlaneDist is VERYCLOSE then ChangeInHeading is RIGHT", engine));
block->addRule(new fl::MamdaniRule("if OurBearingAngle is LOWPOS and TheirBearingAngle" &
  " is VERYNEG and PlaneDist is CLOSE then ChangeInHeading in NOCHANGE", engine));
block->addRule(new fl::MamdaniRule("if OurBearingAngle is LOWNEG and TheirBearingAngle" &
  " is LOWPOS and PlaneDist is CLOSE then ChangeInHeading is LEFT", engine));
block->addRule(new fl::MamdaniRule("if OurBearingAngle is LOWPOS and TheirBearingAngle" &
  " is VERYNEG and PlaneDist is CLOSE then ChangeInHeading is RIGHT", engine));
block->addRule(new fl::MamdaniRule("if OurBearingAngle is NEG and TheirBearingAngle" &
  " is VERYNEG and PlaneDist is CLOSE then ChangeInHeading is NOCHANGE", engine));
block->addRule(new fl::MamdaniRule("if OurBearingAngle is NEG and TheirBearingAngle" &
  " is LOWNEG and PlaneDist is CLOSE then ChangeInHeading is LEFT", engine));
block->addRule(new fl::MamdaniRule("if OurBearingAngle is LOWNEG and TheirBearingAngle" &
  " is NEG and PlaneDist is CLOSE then ChangeInHeading is RIGHT", engine));
block->addRule(new fl::MamdaniRule("if OurBearingAngle is LOWNEG and TheirBearingAngle" &
  " is VERYNEG and PlaneDist is CLOSE then ChangeInHeading is RIGHT", engine));
block->addRule(new fl::MamdaniRule("if OurBearingAngle is VERYNEG and TheirBearingAngle" &
  " is NEG and PlaneDist is CLOSE then ChangeInHeading is RIGHT", engine));
block->addRule(new fl::MamdaniRule("if OurBearingAngle is LOWNEG and TheirBearingAngle" &
  " is LOWNEG and PlaneDist is CLOSE then ChangeInHeading is VERYRIGHT", engine));
block->addRule(new fl::MamdaniRule("if OurBearingAngle is LOWPOS and TheirBearingAngle" &
  " is LOWNEG and PlaneDist is CLOSE then ChangeInHeading is VERYRIGHT", engine));
block->addRule(new fl::MamdaniRule("if OurBearingAngle is LOWPOS and TheirBearingAngle" &
  " is POS and PlaneDist is CLOSE then ChangeInHeading is LEFT", engine));
block->addRule(new fl::MamdaniRule("if OurBearingAngle is POS and TheirBearingAngle" &
  " is VERYPOS and PlaneDist is VERYCLOSE then ChangeInHeading is LEFT", engine));
block->addRule(new fl::MamdaniRule("if OurBearingAngle is VERYPOS and TheirBearingAngle" &
  " is NEG and PlaneDist is VERYCLOSE then ChangeInHeading is VERYLEFT", engine));
block->addRule(new fl::MamdaniRule("if OurBearingAngle is VERYPOS and TheirBearingAngle" &
  " is LOWNEG and PlaneDist is VERYCLOSE then ChangeInHeading is RIGHT", engine));
block->addRule(new fl::MamdaniRule("if OurBearingAngle is LOWPOS and TheirBearingAngle" &
  " is VERYNEG and PlaneDist is CLOSE then ChangeInHeading is NOCHANGE", engine));
block->addRule(new fl::MamdaniRule("if OurBearingAngle is LOWNEG and TheirBearingAngle" &
  " is LOWNEG and PlaneDist is CLOSE then ChangeInHeading is VERYRIGHT", engine));
block->addRule(new fl::MamdaniRule("if OurBearingAngle is LOWPOS and TheirBearingAngle" &
  " is POS and PlaneDist is CLOSE then ChangeInHeading is LEFT", engine));
block->addRule(new fl::MamdaniRule("if OurBearingAngle is POS and TheirBearingAngle" &
  " is VERYPOS and PlaneDist is VERYCLOSE then ChangeInHeading is LEFT", engine));
block->addRule(new fl::MamdaniRule("if OurBearingAngle is LOWPOS and TheirBearingAngle" &
  " is VERYNEG and PlaneDist is CLOSE then ChangeInHeading is VERYRIGHT", engine));
block->addRule(new fl::MamdaniRule("if OurBearingAngle is LOWNEG and TheirBearingAngle" &
  " is LOWNEG and PlaneDist is CLOSE then ChangeInHeading is VERYRIGHT", engine));
block->addRule(new fl::MamdaniRule("if OurBearingAngle is LOWPOS and TheirBearingAngle" &
  " is POS and PlaneDist is CLOSE then ChangeInHeading is LEFT", engine));
block->addRule(new fl::MamdaniRule("if OurBearingAngle is POS and TheirBearingAngle" &
  " is VERYPOS and PlaneDist is VERYCLOSE then ChangeInHeading is LEFT", engine));
PlaneDist is FAR
block->addRule(new fl::MamdaniRule("if OurBearingAngle is LOWPOS and TheirBearingAngle" &
         "is NEG and PlaneDist is CLOSE then ChangeInHeading is VERYRIGHT", engine));
block->addRule(new fl::MamdaniRule("if OurBearingAngle is LOWPOS and TheirBearingAngle" &
         "is LOWNEG and PlaneDist is CLOSE then ChangeInHeading is VERYRIGHT", engine));
block->addRule(new fl::MamdaniRule("if OurBearingAngle is LOWPOS and TheirBearingAngle" &
         "is POS and PlaneDist is CLOSE then ChangeInHeading is VERYLEFT", engine));
block->addRule(new fl::MamdaniRule("if OurBearingAngle is LOWPOS and TheirBearingAngle" &
         "is VERYPOS and PlaneDist is CLOSE then ChangeInHeading in NOCHANGE", engine));
block->addRule(new fl::MamdaniRule("if OurBearingAngle is POS and TheirBearingAngle" &
         "is NEG and PlaneDist is CLOSE then ChangeInHeading is VERYLEFT", engine));
block->addRule(new fl::MamdaniRule("if OurBearingAngle is POS and TheirBearingAngle" &
         "is LOWNEG and PlaneDist is CLOSE then ChangeInHeading is VERYLEFT", engine));
block->addRule(new fl::MamdaniRule("if OurBearingAngle is POS and TheirBearingAngle" &
         "is LOWPOS and PlaneDist is CLOSE then ChangeInHeading is LEFT", engine));
block->addRule(new fl::MamdaniRule("if OurBearingAngle is POS and TheirBearingAngle" &
         "is POS and PlaneDist is CLOSE then ChangeInHeading is NOCHANGE", engine));
block->addRule(new fl::MamdaniRule("if OurBearingAngle is POS and TheirBearingAngle" &
         "is VERYPOS and PlaneDist is CLOSE then ChangeInHeading is NOCHANGE", engine));

//PlaneDist is CLOSE
block->addRule(new fl::MamdaniRule("if OurBearingAngle is NOT.ZERO and TheirBearingAngle" &
         "is NEG and PlaneDist is CLOSE then ChangeInHeading is NOCHANGE", engine));
block->addRule(new fl::MamdaniRule("if OurBearingAngle is NOT.ZERO and TheirBearingAngle" &
         "is LOWNEG and PlaneDist is CLOSE then ChangeInHeading is RIGHT", engine));
block->addRule(new fl::MamdaniRule("if OurBearingAngle is NOT.ZERO and TheirBearingAngle" &
         "is LOWPOS and PlaneDist is CLOSE then ChangeInHeading is RIGHT", engine));
block->addRule(new fl::MamdaniRule("if OurBearingAngle is NOT.ZERO and TheirBearingAngle" &
         "is POS and PlaneDist is FAR then ChangeInHeading is NOCHANGE", engine));
block->addRule(new fl::MamdaniRule("if OurBearingAngle is NOT.ZERO and TheirBearingAngle" &
         "is VERYPOS and PlaneDist is FAR then ChangeInHeading is RIGHT", engine));

43
" is LOWNEG and PlaneDist is FAR then ChangeInHeading is VERYRIGHT", engine));
block->addRule(new fl::MamdaniRule("if OurBearingAngle is LOWNEG and TheirBearingAngle" &
 " is LOWPOS and PlaneDist is FAR then ChangeInHeading is VERYRIGHT", engine));
block->addRule(new fl::MamdaniRule("if OurBearingAngle is LOWNEG and TheirBearingAngle" &
 " is POS and PlaneDist is FAR then ChangeInHeading is VERYRIGHT", engine));
block->addRule(new fl::MamdaniRule("if OurBearingAngle is LOWNEG and TheirBearingAngle" &
 " is VERYPOS and PlaneDist is FAR then ChangeInHeading is VERYRIGHT", engine));
block->addRule(new fl::MamdaniRule("if OurBearingAngle is LOWNEG and TheirBearingAngle" &
 " is VERYNEG and PlaneDist is FAR then ChangeInHeading is VERYRIGHT", engine));
block->addRule(new fl::MamdaniRule("if OurBearingAngle is LOWNEG and TheirBearingAngle" &
 " is NEG and PlaneDist is FAR then ChangeInHeading is VERYRIGHT", engine));
block->addRule(new fl::MamdaniRule("if OurBearingAngle is LOWNEG and TheirBearingAngle" &
 " is LOWPOS and PlaneDist is FAR then ChangeInHeading is VERYRIGHT", engine));
block->addRule(new fl::MamdaniRule("if OurBearingAngle is LOWNEG and TheirBearingAngle" &
 " is LOWPOS and PlaneDist is FAR then ChangeInHeading is VERYLEFT", engine));
block->addRule(new fl::MamdaniRule("if OurBearingAngle is LOWNEG and TheirBearingAngle" &
 " is POS and PlaneDist is FAR then ChangeInHeading is VERYLEFT", engine));
block->addRule(new fl::MamdaniRule("if OurBearingAngle is LOWNEG and TheirBearingAngle" &
 " is VERYPOS and PlaneDist is FAR then ChangeInHeading is VERYLEFT", engine));
block->addRule(new fl::MamdaniRule("if OurBearingAngle is LOWNEG and TheirBearingAngle" &
 " is VERYNEG and PlaneDist is FAR then ChangeInHeading is VERYLEFT", engine));
block->addRule(new fl::MamdaniRule("if OurBearingAngle is LOWNEG and TheirBearingAngle" &
 " is NEG and PlaneDist is FAR then ChangeInHeading is VERYLEFT", engine));
block->addRule(new fl::MamdaniRule("if OurBearingAngle is LOWNEG and TheirBearingAngle" &
 " is LOWPOS and PlaneDist is FAR then ChangeInHeading is VERYLEFT", engine));
block->addRule(new fl::MamdaniRule("if OurBearingAngle is LOWNEG and TheirBearingAngle" &
 " is LOWPOS and PlaneDist is FAR then ChangeInHeading is LEFT", engine));
block->addRule(new fl::MamdaniRule("if OurBearingAngle is LOWNEG and TheirBearingAngle" &
 " is POS and PlaneDist is FAR then ChangeInHeading is LEFT", engine));
block->addRule(new fl::MamdaniRule("if OurBearingAngle is LOWNEG and TheirBearingAngle" &
 " is VERYPOS and PlaneDist is FAR then ChangeInHeading is LEFT", engine));
block->addRule(new fl::MamdaniRule("if OurBearingAngle is LOWNEG and TheirBearingAngle" &
 " is VERYNEG and PlaneDist is FAR then ChangeInHeading is NOCHANGE", engine));
block->addRule(new fl::MamdaniRule("if OurBearingAngle is LOWNEG and TheirBearingAngle" &
 " is NEG and PlaneDist is FAR then ChangeInHeading is NOCHANGE", engine));
planeDist is VERYFAR

block->addRule(new fl::MamdaniRule("if OurBearingAngle is VERYPOS and TheirBearingAngle" &
  " is VERYPOS and PlaneDist is FAR then ChangeInHeading is NOCHANGE", engine));

//PlaneDist is VERYFAR
block->addRule(new fl::MamdaniRule("if OurBearingAngle is VERYNEG and TheirBearingAngle" &
  " is VERYNEG and PlaneDist is VERYFAR then ChangeInHeading is NOCHANGE", engine));
block->addRule(new fl::MamdaniRule("if OurBearingAngle is LOWNEG and TheirBearingAngle" &
  " is LOWNEG and PlaneDist is VERYFAR then ChangeInHeading is NOCHANGE", engine));
block->addRule(new fl::MamdaniRule("if OurBearingAngle is LOWNEG and TheirBearingAngle" &
  " is LOWPOS and PlaneDist is VERYFAR then ChangeInHeading is RIGHT", engine));
block->addRule(new fl::MamdaniRule("if OurBearingAngle is LOWNEG and TheirBearingAngle" &
  " is POS and PlaneDist is VERYFAR then ChangeInHeading is RIGHT", engine));
block->addRule(new fl::MamdaniRule("if OurBearingAngle is LOWNEG and TheirBearingAngle" &
  " is VERYPOS and PlaneDist is VERYFAR then ChangeInHeading is RIGHT", engine));

block->addRule(new fl::MamdaniRule("if OurBearingAngle is NEG and TheirBearingAngle" &
  " is NEG and PlaneDist is VERYFAR then ChangeInHeading is RIGHT", engine));
block->addRule(new fl::MamdaniRule("if OurBearingAngle is NEG and TheirBearingAngle" &
  " is LOWNEG and PlaneDist is VERYFAR then ChangeInHeading is VERYRIGHT", engine));
block->addRule(new fl::MamdaniRule("if OurBearingAngle is NEG and TheirBearingAngle" &
  " is LOWPOS and PlaneDist is VERYFAR then ChangeInHeading is VERYRIGHT", engine));
block->addRule(new fl::MamdaniRule("if OurBearingAngle is NEG and TheirBearingAngle" &
  " is POS and PlaneDist is VERYFAR then ChangeInHeading is VERYRIGHT", engine));
block->addRule(new fl::MamdaniRule("if OurBearingAngle is NEG and TheirBearingAngle" &
  " is VERYPOS and PlaneDist is VERYFAR then ChangeInHeading is VERYRIGHT", engine));

block->addRule(new fl::MamdaniRule("if OurBearingAngle is LOWNEG and TheirBearingAngle" &
  " is VERYNEG and PlaneDist is FAR then ChangeInHeading is NOCHANGE", engine));
block->addRule(new fl::MamdaniRule("if OurBearingAngle is LOWNEG and TheirBearingAngle" &
  " is LOWNEG and PlaneDist is FAR then ChangeInHeading is NOCHANGE", engine));
block->addRule(new fl::MamdaniRule("if OurBearingAngle is LOWNEG and TheirBearingAngle" &
  " is LOWPOS and PlaneDist is FAR then ChangeInHeading is NOCHANGE", engine));
block->addRule(new fl::MamdaniRule("if OurBearingAngle is LOWNEG and TheirBearingAngle" &
  " is POS and PlaneDist is FAR then ChangeInHeading is NOCHANGE", engine));
block->addRule(new fl::MamdaniRule("if OurBearingAngle is LOWNEG and TheirBearingAngle" &
  " is VERYPOS and PlaneDist is FAR then ChangeInHeading is NOCHANGE", engine));

block->addRule(new fl::MamdaniRule("if OurBearingAngle is LOWPOS and TheirBearingAngle" &
  " is VERYNEG and PlaneDist is FAR then ChangeInHeading is NOCHANGE", engine));
block->addRule(new fl::MamdaniRule("if OurBearingAngle is LOWPOS and TheirBearingAngle" &
  " is LOWNEG and PlaneDist is FAR then ChangeInHeading is NOCHANGE", engine));
block->addRule(new fl::MamdaniRule("if OurBearingAngle is LOWPOS and TheirBearingAngle" &
  " is LOWPOS and PlaneDist is FAR then ChangeInHeading is NOCHANGE", engine));
block->addRule(new fl::MamdaniRule("if OurBearingAngle is LOWPOS and TheirBearingAngle" &
  " is POS and PlaneDist is FAR then ChangeInHeading is NOCHANGE", engine));
block->addRule(new fl::MamdaniRule("if OurBearingAngle is LOWPOS and TheirBearingAngle" &
  " is VERYPOS and PlaneDist is FAR then ChangeInHeading is NOCHANGE", engine));

block->addRule(new fl::MamdaniRule("if OurBearingAngle is POS and TheirBearingAngle" &
block->addRule(new fl::MamdaniRule("if OurBearingAngle is POS and TheirBearingAngle is VERY NEG and PlaneDist is VERY FAR then ChangeInHeading is VERY LEFT", engine));
block->addRule(new fl::MamdaniRule("if OurBearingAngle is POS and TheirBearingAngle is LOW NEG and PlaneDist is VERY FAR then ChangeInHeading is LEFT", engine));
block->addRule(new fl::MamdaniRule("if OurBearingAngle is POS and TheirBearingAngle is LOW POS and PlaneDist is VERY FAR then ChangeInHeading is LEFT", engine));
block->addRule(new fl::MamdaniRule("if OurBearingAngle is POS and TheirBearingAngle is POS and PlaneDist is VERY FAR then ChangeInHeading is LEFT", engine));
block->addRule(new fl::MamdaniRule("if OurBearingAngle is POS and TheirBearingAngle is VERY POS and PlaneDist is VERY FAR then ChangeInHeading is LEFT", engine));
block->addRule(new fl::MamdaniRule("if OurBearingAngle is POS and TheirBearingAngle is VERY POS and PlaneDist is VERY FAR then ChangeInHeading is LEFT", engine));
block->addRule(new fl::MamdaniRule("if OurBearingAngle is POS and TheirBearingAngle is LOW POS and PlaneDist is VERY FAR then ChangeInHeading is NO CHANGE", engine));
block->addRule(new fl::MamdaniRule("if OurBearingAngle is POS and TheirBearingAngle is POS and PlaneDist is VERY FAR then ChangeInHeading is NO CHANGE", engine));

engine.addRuleBlock(block);
distanceBetweenPlanes->setInput(distance);
ourBearingAngle->setInput(ourAngle);
theirBearingAngle->setInput(theirAngle);
engine.process();
double output = changeHeading->output().defuzzify();
return output;
}

double AU_UAV_ROS::FuzzyLogicController::sevenOutputs(double distance, double ourAngle, double theirAngle){
fl::FuzzyOperator& op = fl::FuzzyOperator::DefaultFuzzyOperator();
fl::FuzzyEngine engine("Heading - Change", op);

//possible consider changing this to ZEM
fl::InputLVar* distanceBetweenPlanes = new fl::InputLVar("PlaneDist");
distanceBetweenPlanes->addTerm(new fl::ShoulderTerm("CLOSE", 24.0, 50.0, true));
distanceBetweenPlanes->addTerm(new fl::TriangularTerm("MEDIUM", 24.0, 65.0));
distanceBetweenPlanes->addTerm(new fl::ShoulderTerm("FAR", 50.0, 76.0, false));
engine.addInputLVar(distanceBetweenPlanes);

fl::InputLVar* ourBearingAngle = new fl::InputLVar("OurBearingAngle");
ourBearingAngle->addTerm(new fl::ShoulderTerm("VERY NEG", -90.0, -60.0, true));
ourBearingAngle->addTerm(new fl::TriangularTerm("NEG", -70.0, -20.0));
ourBearingAngle->addTerm(new fl::ShoulderTerm("VERY POS", 60.0, 90.0, false));
ourBearingAngle->addTerm(new fl::TriangularTerm("POS", 20.0, 70.0));

fl::InputLVar* theirBearingAngle = new fl::InputLVar("TheirBearingAngle");
theirBearingAngle->addTerm(new fl::ShoulderTerm("VERY POS", 60.0, 90.0, false));
theirBearingAngle->addTerm(new fl::TriangularTerm("POS", 20.0, 70.0));
theirBearingAngle -> addTerm(new fl::ShoulderTerm("VERYNEG", -90.0, -60.0, true));
theirBearingAngle -> addTerm(new fl::TriangularTerm("NEG", -70.0, -20.0));
theirBearingAngle -> addTerm(new fl::TriangularTerm("LOWNEG", -21.0, 1.0));
theirBearingAngle -> addTerm(new fl::TriangularTerm("LOWPOS", -1.0, 21.0));
theirBearingAngle -> addTerm(new fl::TriangularTerm("POS", 20.0, 70.0));
theirBearingAngle -> addTerm(new fl::ShoulderTerm("VERYPOS", 60.0, 90.0, false));

engine.addInputLVar(theirBearingAngle);

fl::OutputLVar* changeHeading = new fl::OutputLVar("ChangeInHeading");
changeHeading -> addTerm(new fl::TriangularTerm("VERYLEFT", -32.5, 12.5));
changeHeading -> addTerm(new fl::TriangularTerm("LEFT", -21.25, -1.25));
changeHeading -> addTerm(new fl::TriangularTerm("SLIGHTLEFT", -10.5, 0.0));
changeHeading -> addTerm(new fl::TriangularTerm("NOCHANGE", -10.0, 10.0));
changeHeading -> addTerm(new fl::TriangularTerm("SLIGHTRIGHT", 0.0, 10.5));
changeHeading -> addTerm(new fl::TriangularTerm("RIGHT", 1.25, 21.25));
changeHeading -> addTerm(new fl::TriangularTerm("VERYRIGHT", 12.5, 32.5));

engine.addOutputLVar(changeHeading);

fl::RuleBlock* block = new fl::RuleBlock();
// PlaneDist is CLOSE
block -> addRule(new fl::MamdaniRule("if OurBearingAngle is VERYNEG and TheirBearingAngle is VERYNEG and PlaneDist is CLOSE then ChangeInHeading is SLIGHTRIGHT", engine));
block -> addRule(new fl::MamdaniRule("if OurBearingAngle is VERYNEG and TheirBearingAngle is NEG and PlaneDist is CLOSE then ChangeInHeading is SLIGHTRIGHT", engine));
block -> addRule(new fl::MamdaniRule("if OurBearingAngle is VERYNEG and TheirBearingAngle is LOWNEG and PlaneDist is CLOSE then ChangeInHeading is RIGHT", engine));
block -> addRule(new fl::MamdaniRule("if OurBearingAngle is VERYNEG and TheirBearingAngle is POS and PlaneDist is CLOSE then ChangeInHeading is VERYRIGHT", engine));
block -> addRule(new fl::MamdaniRule("if OurBearingAngle is VERYNEG and TheirBearingAngle is VERYPOS and PlaneDist is CLOSE then ChangeInHeading is VERYRIGHT", engine));
block -> addRule(new fl::MamdaniRule("if OurBearingAngle is NEG and TheirBearingAngle is VERYNEG and PlaneDist is CLOSE then ChangeInHeading is SLIGHTRIGHT", engine));
block -> addRule(new fl::MamdaniRule("if OurBearingAngle is NEG and TheirBearingAngle is NEG and PlaneDist is CLOSE then ChangeInHeading is SLIGHTRIGHT", engine));
block -> addRule(new fl::MamdaniRule("if OurBearingAngle is NEG and TheirBearingAngle is LOWNEG and PlaneDist is CLOSE then ChangeInHeading is RIGHT", engine));
block -> addRule(new fl::MamdaniRule("if OurBearingAngle is NEG and TheirBearingAngle is POS and PlaneDist is CLOSE then ChangeInHeading is VERYRIGHT", engine));
block -> addRule(new fl::MamdaniRule("if OurBearingAngle is LOWNEG and TheirBearingAngle is VERYNEG and PlaneDist is CLOSE then ChangeInHeading is VERYRIGHT", engine));
block -> addRule(new fl::MamdaniRule("if OurBearingAngle is LOWNEG and TheirBearingAngle is POS and PlaneDist is CLOSE then ChangeInHeading is VERYRIGHT", engine));
block -> addRule(new fl::MamdaniRule("if OurBearingAngle is LOWNEG and TheirBearingAngle is VERYPOS and PlaneDist is CLOSE then ChangeInHeading is VERYRIGHT", engine));
"is VERYPOS and PlaneDist is CLOSE then ChangeInHeading is VERYLEFT", engine));
block->addRule(new fl::MamdaniRule("if OurBearingAngle is POS and TheirBearingAngle" &
"is VERYNEG and PlaneDist is CLOSE then ChangeInHeading is VERYLEFT", engine));
block->addRule(new fl::MamdaniRule("if OurBearingAngle is POS and TheirBearingAngle" &
"is LOWPOS and PlaneDist is CLOSE then ChangeInHeading is LEFT", engine));
block->addRule(new fl::MamdaniRule("if OurBearingAngle is POS and TheirBearingAngle" &
"is POS and PlaneDist is CLOSE then ChangeInHeading is LEFT", engine));
block->addRule(new fl::MamdaniRule("if OurBearingAngle is POS and TheirBearingAngle" &
"is VERYPOS and PlaneDist is CLOSE then ChangeInHeading is SLIGHTLEFT", engine));
block->addRule(new fl::MamdaniRule("if OurBearingAngle is POS and TheirBearingAngle" &
"is VERYNEG and PlaneDist is CLOSE then ChangeInHeading is LEFT", engine));
block->addRule(new fl::MamdaniRule("if OurBearingAngle is POS and TheirBearingAngle" &
"is NEG and PlaneDist is CLOSE then ChangeInHeading is VERYLEFT", engine));
block->addRule(new fl::MamdaniRule("if OurBearingAngle is POS and TheirBearingAngle" &
"is LOWNEG and PlaneDist is CLOSE then ChangeInHeading is VERYLEFT", engine));
block->addRule(new fl::MamdaniRule("if OurBearingAngle is POS and TheirBearingAngle" &
"is LOWPOS and PlaneDist is CLOSE then ChangeInHeading is LEFT", engine));
block->addRule(new fl::MamdaniRule("if OurBearingAngle is POS and TheirBearingAngle" &
"is POS and PlaneDist is CLOSE then ChangeInHeading is LEFT", engine));
block->addRule(new fl::MamdaniRule("if OurBearingAngle is POS and TheirBearingAngle" &
"is VERYPOS and PlaneDist is CLOSE then ChangeInHeading is SLIGHTLEFT", engine));
block->addRule(new fl::MamdaniRule("if OurBearingAngle is POS and TheirBearingAngle" &
"is VERYNEG and PlaneDist is MEDIUM then ChangeInHeading is SLIGHTRIGHT", engine));
block->addRule(new fl::MamdaniRule("if OurBearingAngle is POS and TheirBearingAngle" &
"is NEG and PlaneDist is MEDIUM then ChangeInHeading is SLIGHTRIGHT", engine));
block->addRule(new fl::MamdaniRule("if OurBearingAngle is POS and TheirBearingAngle" &
"is LOWNEG and PlaneDist is MEDIUM then ChangeInHeading is RIGHT", engine));
block->addRule(new fl::MamdaniRule("if OurBearingAngle is POS and TheirBearingAngle" &
"is LOWPOS and PlaneDist is MEDIUM then ChangeInHeading is VERYRIGHT", engine));
block->addRule(new fl::MamdaniRule("if OurBearingAngle is POS and TheirBearingAngle" &
"is POS and PlaneDist is MEDIUM then ChangeInHeading is VERYRIGHT", engine));
block->addRule(new fl::MamdaniRule("if OurBearingAngle is NEG and TheirBearingAngle" &
"is VERYNEG and PlaneDist is MEDIUM then ChangeInHeading is VERYRIGHT", engine));
block->addRule(new fl::MamdaniRule("if OurBearingAngle is VERYPOS and TheirBearingAngle is LOWPOS and PlaneDist is MEDIUM then ChangeInHeading in VERYRIGHT", engine));
block->addRule(new fl::MamdaniRule("if OurBearingAngle is VERYPOS and TheirBearingAngle is LOWNEG and PlaneDist is MEDIUM then ChangeInHeading in VERYRIGHT", engine));
block->addRule(new fl::MamdaniRule("if OurBearingAngle is POS and TheirBearingAngle is LOWNEG and PlaneDist is MEDIUM then ChangeInHeading in VERYRIGHT", engine));
block->addRule(new fl::MamdaniRule("if OurBearingAngle is POS and TheirBearingAngle is LOWPOS and PlaneDist is MEDIUM then ChangeInHeading in VERYRIGHT", engine));
block->addRule(new fl::MamdaniRule("if OurBearingAngle is POS and TheirBearingAngle is LOWNEG and PlaneDist is MEDIUM then ChangeInHeading in VERYRIGHT", engine));
block->addRule(new fl::MamdaniRule("if OurBearingAngle is POS and TheirBearingAngle is LOWNEG and PlaneDist is MEDIUM then ChangeInHeading in VERYRIGHT", engine));
block->addRule(new fl::MamdaniRule("if OurBearingAngle is POS and TheirBearingAngle is LOWPOS and PlaneDist is MEDIUM then ChangeInHeading in VERYRIGHT", engine));
block->addRule(new fl::MamdaniRule("if OurBearingAngle is POS and TheirBearingAngle is LOWPOS and PlaneDist is MEDIUM then ChangeInHeading in VERYRIGHT", engine));
block->addRule(new fl::MamdaniRule("if OurBearingAngle is POS and TheirBearingAngle is LOWPOS and PlaneDist is MEDIUM then ChangeInHeading in LEFT", engine));
block->addRule(new fl::MamdaniRule("if OurBearingAngle is POS and TheirBearingAngle is LOWPOS and PlaneDist is MEDIUM then ChangeInHeading in LEFT", engine));
block->addRule(new fl::MamdaniRule("if OurBearingAngle is POS and TheirBearingAngle is LOWPOS and PlaneDist is MEDIUM then ChangeInHeading in SlightLEFT", engine));
block->addRule(new fl::MamdaniRule("if OurBearingAngle is POS and TheirBearingAngle is LOWPOS and PlaneDist is MEDIUM then ChangeInHeading in SlightLEFT", engine));
block->addRule(new fl::MamdaniRule("if OurBearingAngle is POS and TheirBearingAngle is LOWPOS and PlaneDist is MEDIUM then ChangeInHeading in SlightLEFT", engine));
block->addRule(new fl::MamdaniRule("if OurBearingAngle is POS and TheirBearingAngle is LOWPOS and PlaneDist is MEDIUM then ChangeInHeading in SlightLEFT", engine));
block->addRule(new fl::MamdaniRule("if OurBearingAngle is POS and TheirBearingAngle is LOWPOS and PlaneDist is MEDIUM then ChangeInHeading in SlightLEFT", engine));
block->addRule(new fl::MamdaniRule("if OurBearingAngle is POS and TheirBearingAngle is LOWPOS and PlaneDist is MEDIUM then ChangeInHeading in SlightLEFT", engine));
block->addRule(new fl::MamdaniRule("if OurBearingAngle is POS and TheirBearingAngle is LOWPOS and PlaneDist is MEDIUM then ChangeInHeading in SlightLEFT", engine));
" is LOWPOS and PlaneDist is MEDIUM then ChangeInHeading is LEFT", engine));
block->addRule(new fl::MamdaniRule("if OurBearingAngle is VERYPOS and TheirBearingAngle" &
" is POS and PlaneDist is MEDIUM then ChangeInHeading is SLIGHTLEFT", engine));
block->addRule(new fl::MamdaniRule("if OurBearingAngle is VERYPOS and TheirBearingAngle" &
" is VERYPOS and PlaneDist is MEDIUM then ChangeInHeading is SLIGHTLEFT", engine));

//PlaneDist is FAR
block->addRule(new fl::MamdaniRule("if OurBearingAngle is VERYNEG and TheirBearingAngle" &
" is VERYNEG and PlaneDist is FAR then ChangeInHeading is SLIGHTRIGHT", engine));
block->addRule(new fl::MamdaniRule("if OurBearingAngle is VERYNEG and TheirBearingAngle" &
" is NEG and PlaneDist is FAR then ChangeInHeading is SLIGHTRIGHT", engine));
block->addRule(new fl::MamdaniRule("if OurBearingAngle is VERYNEG and TheirBearingAngle" &
" is LOWNEG and PlaneDist is FAR then ChangeInHeading is RIGHT", engine));
block->addRule(new fl::MamdaniRule("if OurBearingAngle is LOWPOS and PlaneDist is FAR then ChangeInHeading" &
" is LOS and PlaneDist is FAR then ChangeInHeading is VERYRIGHT", engine));
block->addRule(new fl::MamdaniRule("if OurBearingAngle is NEG and TheirBearingAngle" &
" is POS and PlaneDist is FAR then ChangeInHeading is VERYRIGHT", engine));
block->addRule(new fl::MamdaniRule("if OurBearingAngle is NEG and TheirBearingAngle" &
" is LOS and PlaneDist is FAR then ChangeInHeading is VERYRIGHT", engine));
block->addRule(new fl::MamdaniRule("if OurBearingAngle is NEG and TheirBearingAngle" &
" is POS and PlaneDist is FAR then ChangeInHeading is VERYRIGHT", engine));
block->addRule(new fl::MamdaniRule("if OurBearingAngle is NEG and TheirBearingAngle" &
" is LOS and PlaneDist is FAR then ChangeInHeading is VERYRIGHT", engine));
block->addRule(new fl::MamdaniRule("if OurBearingAngle is NEG and TheirBearingAngle" &
" is POS and PlaneDist is FAR then ChangeInHeading is VERYRIGHT", engine));
block->addRule(new fl::MamdaniRule("if OurBearingAngle is LOWNEG and TheirBearingAngle" &
" is POS and PlaneDist is FAR then ChangeInHeading is SLIGHTLEFT", engine));
block->addRule(new fl::MamdaniRule("if OurBearingAngle is LOWNEG and TheirBearingAngle" &
" is NEG and PlaneDist is FAR then ChangeInHeading is SLIGHTLEFT", engine));
block->addRule(new fl::MamdaniRule("if OurBearingAngle is LOWNEG and TheirBearingAngle" &
" is NEG and PlaneDist is FAR then ChangeInHeading is RIGHT", engine));
block->addRule(new fl::MamdaniRule("if OurBearingAngle is LOWNEG and TheirBearingAngle" &
" is LOWNEG and PlaneDist is FAR then ChangeInHeading is RIGHT", engine));
block->addRule(new fl::MamdaniRule("if OurBearingAngle is LOWNEG and TheirBearingAngle" &
" is LOS and PlaneDist is FAR then ChangeInHeading is LEFT", engine));
block->addRule(new fl::MamdaniRule("if OurBearingAngle is LOWNEG and TheirBearingAngle" &
" is NEG and PlaneDist is FAR then ChangeInHeading is LEFT", engine));
block->addRule(new fl::MamdaniRule("if OurBearingAngle is LOWNEG and TheirBearingAngle" &
" is LOS and PlaneDist is FAR then ChangeInHeading is LEFT", engine));
block->addRule(new fl::MamdaniRule("if OurBearingAngle is LOWNEG and TheirBearingAngle" &
" is POS and PlaneDist is FAR then ChangeInHeading is LEFT", engine));
block->addRule(new fl::MamdaniRule("if OurBearingAngle is LOWNEG and TheirBearingAngle" &
" is NEG and PlaneDist is FAR then ChangeInHeading is LEFT", engine));
block->addRule(new fl::MamdaniRule("if OurBearingAngle is LOWNEG and TheirBearingAngle" &
" is LOS and PlaneDist is FAR then ChangeInHeading is LEFT", engine));
block->addRule(new fl::MamdaniRule("if OurBearingAngle is LOWNEG and TheirBearingAngle" &
" is POS and PlaneDist is FAR then ChangeInHeading is LEFT", engine));
block->addRule(new fl::MamdaniRule("if OurBearingAngle is LOWNEG and TheirBearingAngle" &
" is NEG and PlaneDist is FAR then ChangeInHeading is LEFT", engine));
block->addRule(new fl::MamdaniRule("if OurBearingAngle is LOWNEG and TheirBearingAngle" &
" is LOS and PlaneDist is FAR then ChangeInHeading is LEFT", engine));
block->addRule(new fl::MamdaniRule("if OurBearingAngle is LOWNEG and TheirBearingAngle" &
" is POS and PlaneDist is FAR then ChangeInHeading is LEFT", engine));
block->addRule(new fl::MamdaniRule("if OurBearingAngle is LOWNEG and TheirBearingAngle" &
" is NEG and PlaneDist is FAR then ChangeInHeading is LEFT", engine));
block->addRule(new fl::MamdaniRule("if OurBearingAngle is LOWNEG and TheirBearingAngle" &
" is LOS and PlaneDist is FAR then ChangeInHeading is LEFT", engine));
block->addRule(new fl::MamdaniRule("if OurBearingAngle is LOWNEG and TheirBearingAngle" &
" is POS and PlaneDist is FAR then ChangeInHeading is LEFT", engine));
block->addRule(new fl::MamdaniRule("if OurBearingAngle is LOWNEG and TheirBearingAngle" &
" is NEG and PlaneDist is FAR then ChangeInHeading is LEFT", engine));
block->addRule(new fl::MamdaniRule("if OurBearingAngle is LOWNEG and TheirBearingAngle" &
" is LOS and PlaneDist is FAR then ChangeInHeading is LEFT", engine));
block->addRule(new fl::MamdaniRule("if OurBearingAngle is LOWNEG and TheirBearingAngle" &
" is POS and PlaneDist is FAR then ChangeInHeading is LEFT", engine));
block->addRule(new fl::MamdaniRule("if OurBearingAngle is LOWNEG and TheirBearingAngle" &
" is NEG and PlaneDist is FAR then ChangeInHeading is LEFT", engine));
block->addRule(new fl::MamdaniRule("if OurBearingAngle is LOWNEG and TheirBearingAngle" &
" is LOS and PlaneDist is FAR then ChangeInHeading is LEFT", engine));
block->addRule(new fl::MamdaniRule("if OurBearingAngle is LOWNEG and TheirBearingAngle" &
" is POS and PlaneDist is FAR then ChangeInHeading is LEFT", engine));
block->addRule(new fl::MamdaniRule("if OurBearingAngle is POS and TheirBearingAngle is VERYNEG and PlaneDist is FAR then ChangeInHeading is LEFT", engine));
block->addRule(new fl::MamdaniRule("if OurBearingAngle is POS and TheirBearingAngle is VERYNEG and PlaneDist is FAR then ChangeInHeading is LEFT", engine));
block->addRule(new fl::MamdaniRule("if OurBearingAngle is POS and TheirBearingAngle is LOWNEG and PlaneDist is FAR then ChangeInHeading is VERYLEFT", engine));
block->addRule(new fl::MamdaniRule("if OurBearingAngle is POS and TheirBearingAngle is LOWPOS and PlaneDist is FAR then ChangeInHeading is LEFT", engine));
block->addRule(new fl::MamdaniRule("if OurBearingAngle is POS and TheirBearingAngle is POS and PlaneDist is FAR then ChangeInHeading is SLIGHTLEFT", engine));
block->addRule(new fl::MamdaniRule("if OurBearingAngle is POS and TheirBearingAngle is VERYPOS and PlaneDist is FAR then ChangeInHeading is SLIGHTLEFT", engine));
block->addRule(new fl::MamdaniRule("if OurBearingAngle is VERYPOS and TheirBearingAngle is VERYNEG and PlaneDist is FAR then ChangeInHeading is LEFT", engine));
block->addRule(new fl::MamdaniRule("if OurBearingAngle is VERYPOS and TheirBearingAngle is VERYNEG and PlaneDist is FAR then ChangeInHeading is LEFT", engine));
block->addRule(new fl::MamdaniRule("if OurBearingAngle is VERYPOS and TheirBearingAngle is LOWNEG and PlaneDist is FAR then ChangeInHeading is VERYLEFT", engine));
block->addRule(new fl::MamdaniRule("if OurBearingAngle is VERYPOS and TheirBearingAngle is LOWPOS and PlaneDist is FAR then ChangeInHeading is LEFT", engine));
block->addRule(new fl::MamdaniRule("if OurBearingAngle is VERYPOS and TheirBearingAngle is POS and PlaneDist is FAR then ChangeInHeading is SLIGHTLEFT", engine));
block->addRule(new fl::MamdaniRule("if OurBearingAngle is VERYPOS and TheirBearingAngle is VERYPOS and PlaneDist is FAR then ChangeInHeading is SLIGHTLEFT", engine));
engine.addRuleBlock(block);

distanceBetweenPlanes->setInput(distance);
ourBearingAngle->setInput(ourAngle);
theirBearingAngle->setInput(theirAngle);
engine.process();
double output = changeHeading->output().defuzzify();
return output;
}

double AU_UAV_ROS::FuzzyLogicController::smallDistance(double ourAngle, double theirAngle){
fl::FuzzyOperator& op = fl::FuzzyOperator::DefaultFuzzyOperator();
fl::FuzzyEngine engine("Heading-Change", op);
fl::InputLVar* ourBearingAngle = new fl::InputLVar("OurBearingAngle");
ourBearingAngle->addTerm(new fl::ShoulderTerm("VERYNEG", -90.0, -60.0, true));
ourBearingAngle->addTerm(new fl::TriangularTerm("NEG", -70, -20.0));
ourBearingAngle->addTerm(new fl::TriangularTerm("LOWNEG", -21.0, 1.0));
ourBearingAngle->addTerm(new fl::TriangularTerm("LOWPOS", -1.0, 21.0));
ourBearingAngle->addTerm(new fl::TriangularTerm("POS", 20, 70.0));
ourBearingAngle->addTerm(new fl::ShoulderTerm("VERYPOS", 60, 90.0, false));
engine.addInputLVar(ourBearingAngle);

fl::InputLVar* theirBearingAngle = new fl::InputLVar("TheirBearingAngle");
theirBearingAngle->addTerm(new fl::ShoulderTerm("VERYNEG", -90.0, -60.0, true));
theirBearingAngle->addTerm(new fl::TriangularTerm("NEG", -70, -20.0));
theirBearingAngle->addTerm(new fl::TriangularTerm("LOWNEG", -21.0, 1.0));
theirBearingAngle->addTerm(new fl::TriangularTerm("LOWPOS", -1.0, 21.0));
theirBearingAngle->addTerm(new fl::TriangularTerm("POS", 20, 70.0));
theirBearingAngle->addTerm(new fl::ShoulderTerm("VERYPOS", 60, 90.0, false));
engine.addInputLVar(theirBearingAngle);

fl::OutputLVar* changeHeading = new fl::OutputLVar("ChangeInHeading");
changeHeading->addTerm(new fl::TriangularTerm("VERYLEFT", -32.5, 12.5));
changeHeading->addTerm(new fl::TriangularTerm("LEFT", -21.25, -1.25));
changeHeading->addTerm(new fl::TriangularTerm("SLIGHTLEFT", -10.5, 0.0));
changeHeading->addTerm(new fl::TriangularTerm("NOCHANGE", -10.0, 10.0));
changeHeading->addTerm(new fl::TriangularTerm("SLIGHTRIGHT", 0.0, 10.5));
changeHeading->addTerm(new fl::TriangularTerm("RIGHT", 1.25, 21.25));
changeHeading->addTerm(new fl::TriangularTerm("VERYRIGHT", 12.5, 32.5));
engine.addOutputLVar(changeHeading);

fl::RuleBlock* block = new fl::RuleBlock();
//PlaneDist is CLOSE
block->addRule(new fl::MamdaniRule("if OurBearingAngle is VERYNEG and TheirBearingAngle" &
  " is VERYNEG then ChangeInHeading is NOCHANGE", engine));
block->addRule(new fl::MamdaniRule("if OurBearingAngle is VERYNEG and TheirBearingAngle" &
  " is NEG then ChangeInHeading is NOCHANGE", engine));
block->addRule(new fl::MamdaniRule("if OurBearingAngle is VERYNEG and TheirBearingAngle" &
  " is LOWNEG then ChangeInHeading is SLIGHTRIGHT", engine));
block->addRule(new fl::MamdaniRule("if OurBearingAngle is VERYNEG and TheirBearingAngle" &
  " is LOWPOS then ChangeInHeading is RIGHT", engine));
block->addRule(new fl::MamdaniRule("if OurBearingAngle is VERYNEG and TheirBearingAngle" &
  " is VERYPOS then ChangeInHeading is RIGHT", engine));
block->addRule(new fl::MamdaniRule("if OurBearingAngle is NEG and TheirBearingAngle" &
  " is VERYNEG then ChangeInHeading is NOCHANGE", engine));
block->addRule(new fl::MamdaniRule("if OurBearingAngle is NEG and TheirBearingAngle" &
  " is NEG then ChangeInHeading is NOCHANGE", engine));
block->addRule(new fl::MamdaniRule("if OurBearingAngle is NEG and TheirBearingAngle" &
  " is LOWNEG then ChangeInHeading is SLIGHTRIGHT", engine));
block->addRule(new fl::MamdaniRule("if OurBearingAngle is NEG and TheirBearingAngle" &
  " is LOWPOS then ChangeInHeading is RIGHT", engine));
block->addRule(new fl::MamdaniRule("if OurBearingAngle is NEG and TheirBearingAngle" &
  " is POS then ChangeInHeading is RIGHT", engine));
block->addRule(new fl::MamdaniRule("if OurBearingAngle is NEG and TheirBearingAngle" &
  " is VERYPOS then ChangeInHeading is RIGHT", engine));
block->addRule(new fl::MamdaniRule("if OurBearingAngle is LOWNEG and TheirBearingAngle" &
  " is VERYNEG then ChangeInHeading is SLIGHTRIGHT", engine));
block->addRule(new fl::MamdaniRule("if OurBearingAngle is LOWNEG and TheirBearingAngle" &
  " is NEG then ChangeInHeading is SLIGHTRIGHT", engine));
block->addRule(new fl::MamdaniRule("if OurBearingAngle is LOWNEG and TheirBearingAngle" &
  " is LOWNEG then ChangeInHeading is VERYRIGHT", engine));
block->addRule(new fl::MamdaniRule("if OurBearingAngle is LOWNEG and TheirBearingAngle" &
  " is LOWPOS then ChangeInHeading is VERYLEFT", engine));
block->addRule(new fl::MamdaniRule("if OurBearingAngle is LOWNEG and TheirBearingAngle" &
  " is VERYPOS then ChangeInHeading is VERYLEFT", engine));
block->addRule(new fl::MamdaniRule("if OurBearingAngle is LOWPOS and TheirBearingAngle" &
  " is VERYNEG then ChangeInHeading is VERYRIGHT", engine));
block->addRule(new fl::MamdaniRule("if OurBearingAngle is LOWPOS and TheirBearingAngle" &
  " is NEG then ChangeInHeading is VERYRIGHT", engine));

52
block->addRule(new fl::MamdaniRule("if OurBearingAngle is LOWPOS and TheirBearingAngle" &
" is LOWNEG then ChangeInHeading is VERYRIGHT", engine));
block->addRule(new fl::MamdaniRule("if OurBearingAngle is LOWPOS and TheirBearingAngle" &
" is LOWPOS then ChangeInHeading is VERYLEFT", engine));
block->addRule(new fl::MamdaniRule("if OurBearingAngle is LOWPOS and TheirBearingAngle" &
" is POS then ChangeInHeading is SLIGHTLEFT", engine));
block->addRule(new fl::MamdaniRule("if OurBearingAngle is LOWPOS and TheirBearingAngle" &
" is VERYPOS then ChangeInHeading is SLIGHTLEFT", engine));
block->addRule(new fl::MamdaniRule("if OurBearingAngle is POS and TheirBearingAngle" &
" is VERYNEG then ChangeInHeading is VERYLEFT", engine));
block->addRule(new fl::MamdaniRule("if OurBearingAngle is POS and TheirBearingAngle" &
" is NEG then ChangeInHeading is VERYLEFT", engine));
block->addRule(new fl::MamdaniRule("if OurBearingAngle is POS and TheirBearingAngle" &
" is LOWNEG then ChangeInHeading is VERYLEFT", engine));
block->addRule(new fl::MamdaniRule("if OurBearingAngle is POS and TheirBearingAngle" &
" is LOWPOS then ChangeInHeading is SLIGHTLEFT", engine));
block->addRule(new fl::MamdaniRule("if OurBearingAngle is POS and TheirBearingAngle" &
" is POS then ChangeInHeading is NOCHANGE", engine));
block->addRule(new fl::MamdaniRule("if OurBearingAngle is POS and TheirBearingAngle" &
" is VERYPOS then ChangeInHeading is NOCHANGE", engine));
block->addRule(new fl::MamdaniRule("if OurBearingAngle is VERYPOS and TheirBearingAngle" &
" is VERYNEG then ChangeInHeading is LEFT", engine));
block->addRule(new fl::MamdaniRule("if OurBearingAngle is VERYPOS and TheirBearingAngle" &
" is NEG then ChangeInHeading is LEFT", engine));
block->addRule(new fl::MamdaniRule("if OurBearingAngle is VERYPOS and TheirBearingAngle" &
" is LOWNEG then ChangeInHeading is LEFT", engine));
block->addRule(new fl::MamdaniRule("if OurBearingAngle is VERYPOS and TheirBearingAngle" &
" is LOWPOS then ChangeInHeading is SLIGHTLEFT", engine));
block->addRule(new fl::MamdaniRule("if OurBearingAngle is VERYPOS and TheirBearingAngle" &
" is POS then ChangeInHeading is NOCHANGE", engine));
block->addRule(new fl::MamdaniRule("if OurBearingAngle is VERYPOS and TheirBearingAngle" &
" is VERYPOS then ChangeInHeading is NOCHANGE", engine));

engine.addRuleBlock(block);
ourBearingAngle->setInput(ourAngle);
theirBearingAngle->setInput(theirAngle);
engine.process();
double output = changeHeading->output().defuzzify();
return output;
}

double AU_UAV_ROS::FuzzyLogicController::smallDistance2(double ourAngle, double theirAngle){
fl::FuzzyOperator& op = fl::FuzzyOperator::DefaultFuzzyOperator();
fl::FuzzyEngine engine("Heading=Change", op);

fl::InputLVar* ourBearingAngle = new fl::InputLVar("OurBearingAngle");
ourBearingAngle->addTerm(new fl::ShoulderTerm("VERYNEG", -90.0, -40.0, true));
ourBearingAngle->addTerm(new fl::TriangularTerm("NEG", -50, -10.0));
ourBearingAngle->addTerm(new fl::TriangularTerm("LOWNEG", -20.0, 1.0));
ourBearingAngle->addTerm(new fl::TriangularTerm("LOWPOS", -1.0, 20.0));
ourBearingAngle->addTerm(new fl::TriangularTerm("POS", 10, 50.0));
ourBearingAngle->addTerm(new fl::ShoulderTerm("VERYPOS", 40, 90.0, false));
engine.addInputLVar(ourBearingAngle);

fl::InputLVar* theirBearingAngle = new fl::InputLVar("TheirBearingAngle");
theirBearingAngle->addTerm(new fl::ShoulderTerm("VERYNEG", -90.0, -40.0, true));
theirBearingAngle->addTerm(new fl::TriangularTerm("NEG", -50, -10.0));
theirBearingAngle->addTerm(new fl::TriangularTerm("LOWNEG", -20.0, 1.0));
theirBearingAngle->addTerm(new fl::TriangularTerm("LOWPOS", -1.0, 20.0));
theirBearingAngle->addTerm(new fl::TriangularTerm("POS", 10, 50.0));
theirBearingAngle->addTerm(new fl::ShoulderTerm("VERYPOS", 40, 90.0, false));
engine.addInputLVar(theirBearingAngle);

fl::OutputLVar* changeHeading = new fl::OutputLVar("ChangeInHeading");
changeHeading->addTerm(new fl::TriangularTerm("VERYLEFT", -32.5, 12.5));
changeHeading->addTerm(new fl::TriangularTerm("LEFT", -21.25, -1.25));
changeHeading->addTerm(new fl::TriangularTerm("SLIGHTLEFT", -10.5, 0.0));
changeHeading->addTerm(new fl::TriangularTerm("NOCHANGE", -10.0, 10.0));
changeHeading->addTerm(new fl::TriangularTerm("SLIGHTRIGHT", 0.0, 10.5));
changeHeading->addTerm(new fl::TriangularTerm("RIGHT", 1.25, 21.25));
changeHeading->addTerm(new fl::TriangularTerm("VERYRIGHT", 12.5, 32.5));
engine.addOutputLVar(changeHeading);

fl::RuleBlock* block = new fl::RuleBlock();
//PlaneDist is CLOSE
block->addRule(new fl::MamdaniRule("if OurBearingAngle is VERYNEG and TheirBearingAngle" &
" is VERYNEG then ChangeInHeading is NOCHANGE", engine));
block->addRule(new fl::MamdaniRule("if OurBearingAngle is VERYNEG and TheirBearingAngle" &
" is NEG then ChangeInHeading is NOCHANGE", engine));
block->addRule(new fl::MamdaniRule("if OurBearingAngle is VERYNEG and TheirBearingAngle" &
" is LOWNEG then ChangeInHeading is SLIGHTRIGHT", engine));
block->addRule(new fl::MamdaniRule("if OurBearingAngle is VERYNEG and TheirBearingAngle" &
" is LOWPOS then ChangeInHeading is RIGHT", engine));
block->addRule(new fl::MamdaniRule("if OurBearingAngle is VERYNEG and TheirBearingAngle" &
" is POS then ChangeInHeading is RIGHT", engine));
block->addRule(new fl::MamdaniRule("if OurBearingAngle is VERYNEG and TheirBearingAngle" &
" is VERYPOS then ChangeInHeading is RIGHT", engine));

block->addRule(new fl::MamdaniRule("if OurBearingAngle is NEG and TheirBearingAngle" &
" is VERYNEG then ChangeInHeading is NOCHANGE", engine));
block->addRule(new fl::MamdaniRule("if OurBearingAngle is NEG and TheirBearingAngle" &
" is NEG then ChangeInHeading is NOCHANGE", engine));
block->addRule(new fl::MamdaniRule("if OurBearingAngle is NEG and TheirBearingAngle" &
" is LOWNEG then ChangeInHeading is SLIGHTRIGHT", engine));
block->addRule(new fl::MamdaniRule("if OurBearingAngle is NEG and TheirBearingAngle" &
" is LOWPOS then ChangeInHeading is VERYRIGHT", engine));
block->addRule(new fl::MamdaniRule("if OurBearingAngle is NEG and TheirBearingAngle" &
" is POS then ChangeInHeading is VERYRIGHT", engine));
block->addRule(new fl::MamdaniRule("if OurBearingAngle is NEG and TheirBearingAngle" &
" is VERYPOS then ChangeInHeading is VERYRIGHT", engine));

block->addRule(new fl::MamdaniRule("if OurBearingAngle is LOWNEG and TheirBearingAngle" &
" is VERYNEG then ChangeInHeading is SLIGHTRIGHT", engine));
block->addRule(new fl::MamdaniRule("if OurBearingAngle is LOWNEG and TheirBearingAngle" &
" is NEG then ChangeInHeading is SLIGHTRIGHT", engine));
block->addRule(new fl::MamdaniRule("if OurBearingAngle is LOWNEG and TheirBearingAngle" &
" is LOWNEG then ChangeInHeading is VERYRIGHT", engine));
block->addRule(new fl::MamdaniRule("if OurBearingAngle is LOWNEG and TheirBearingAngle" &
" is LOWPOS then ChangeInHeading is VERYRIGHT", engine));
block->addRule(new fl::MamdaniRule("if OurBearingAngle is LOWNEG and TheirBearingAngle" &
" is POS then ChangeInHeading is VERYLEFT", engine));
block->addRule(new fl::MamdaniRule("if OurBearingAngle is LOWNEG and TheirBearingAngle" &
" is VERYPOS then ChangeInHeading is VERYLEFT", engine));
block->addRule(new fl::MamdaniRule("if OurBearingAngle is LOWPOS and TheirBearingAngle is VERYPOS then ChangeInHeading is VERYLEFT", engine));
block->addRule(new fl::MamdaniRule("if OurBearingAngle is LOWPOS and TheirBearingAngle is VERYNEG then ChangeInHeading is VERYRIGHT", engine));
block->addRule(new fl::MamdaniRule("if OurBearingAngle is LOWPOS and TheirBearingAngle is NEG then ChangeInHeading is VERYRIGHT", engine));
block->addRule(new fl::MamdaniRule("if OurBearingAngle is LOWPOS and TheirBearingAngle is LOWNEG then ChangeInHeading is VERYRIGHT", engine));
block->addRule(new fl::MamdaniRule("if OurBearingAngle is LOWPOS and TheirBearingAngle is POS then ChangeInHeading is VERYLEFT", engine));
block->addRule(new fl::MamdaniRule("if OurBearingAngle is LOWPOS and TheirBearingAngle is VERYPOS then ChangeInHeading is SLIGHTLEFT", engine));
block->addRule(new fl::MamdaniRule("if OurBearingAngle is LOWPOS and TheirBearingAngle is VERYNEG then ChangeInHeading is SLIGHTLEFT", engine));
block->addRule(new fl::MamdaniRule("if OurBearingAngle is POS and TheirBearingAngle is VERYPOS then ChangeInHeading is NO CHANGE", engine));
block->addRule(new fl::MamdaniRule("if OurBearingAngle is POS and TheirBearingAngle is VERYNEG then ChangeInHeading is LEFT", engine));
block->addRule(new fl::MamdaniRule("if OurBearingAngle is POS and TheirBearingAngle is NEG then ChangeInHeading is LEFT", engine));
block->addRule(new fl::MamdaniRule("if OurBearingAngle is POS and TheirBearingAngle is LOWNEG then ChangeInHeading is LEFT", engine));
block->addRule(new fl::MamdaniRule("if OurBearingAngle is POS and TheirBearingAngle is LOWPOS then ChangeInHeading is SLIGHTLEFT", engine));
block->addRule(new fl::MamdaniRule("if OurBearingAngle is POS and TheirBearingAngle is POS then ChangeInHeading is NO CHANGE", engine));
block->addRule(new fl::MamdaniRule("if OurBearingAngle is POS and TheirBearingAngle is VERYPOS then ChangeInHeading is NO CHANGE", engine));
block->addRule(new fl::MamdaniRule("if OurBearingAngle is POS and TheirBearingAngle is VERYNEG then ChangeInHeading is LEFT", engine));
block->addRule(new fl::MamdaniRule("if OurBearingAngle is POS and TheirBearingAngle is NEG then ChangeInHeading is LEFT", engine));
block->addRule(new fl::MamdaniRule("if OurBearingAngle is POS and TheirBearingAngle is LOWNEG then ChangeInHeading is LEFT", engine));
block->addRule(new fl::MamdaniRule("if OurBearingAngle is POS and TheirBearingAngle is LOWPOS then ChangeInHeading is SLIGHTLEFT", engine));
block->addRule(new fl::MamdaniRule("if OurBearingAngle is POS and TheirBearingAngle is POS then ChangeInHeading is NO CHANGE", engine));
block->addRule(new fl::MamdaniRule("if OurBearingAngle is POS and TheirBearingAngle is VERYPOS then ChangeInHeading is NO CHANGE", engine));

engine.addRuleBlock(block);
ourBearingAngle->setInput(ourAngle);
theirBearingAngle->setInput(theirAngle);
engine.process();
double output = changeHeading->output().defuzzify();
return output;
}

double AU_UAV_ROS::FuzzyLogicController::copyCat(double distance, double ourAngle){
fl::FuzzyOperator& op = fl::FuzzyOperator::DefaultFuzzyOperator();
fl::FuzzyEngine engine("Heading-Change", op);

fl::InputLVar* distanceBetweenPlanes = new fl::InputLVar("PlaneDist");
distanceBetweenPlanes->addTerm(new fl::ShoulderTerm("DANGER", 12.0, 26.0, true));
distanceBetweenPlanes->addTerm(new fl::TriangularTerm("ALERT", 18.0, 48.0));
distanceBetweenPlanes->addTerm(new fl::TriangularTerm("MEDIUM", 42.0, 78.0));
distanceBetweenPlanes->addTerm(new fl::ShoulderTerm("SAFE", 70.0, 78.0, false));
engine.addInputLVar(distanceBetweenPlanes);

fl::InputLVar* ourBearingAngle = new fl::InputLVar("OurBearingAngle");
fl::TriangularTerm *posleave = new fl::TriangularTerm("POSLEAVE", -185.0, -45.0);
posleave->setB(-90.0);
ourBearingAngle->addTerm(posleave);
ourBearingAngle->addTerm(new fl::TriangularTerm("POSFIT", -90.0, 0.0));
ourBearingAngle->setB(-0.01);
ourBearingAngle->addTerm(poshead);
fl::TriangularTerm *neghead = new fl::TriangularTerm("NEGHEAD", 0.0, 45.0);
neghead->setB(0.01);
ourBearingAngle->addTerm(neghead);
ourBearingAngle->addTerm(new fl::TriangularTerm("NEGFIT", 0.0, 90.0));
ourBearingAngle->addTerm(negleave);
ourBearingAngle->addTerm(negleave);
engine.addInputLVar(ourBearingAngle);

fl::OutputLVar* changeHeading = new fl::OutputLVar("ChangeInHeading");
changeHeading->addTerm(new fl::TriangularTerm("POSLARGE", -32.5, 12.5));
changeHeading->addTerm(new fl::TriangularTerm("POSMEDIUM", -21.25, -1.25));
changeHeading->addTerm(new fl::TriangularTerm("POSSMALL", -10.5, 0.0));
changeHeading->addTerm(new fl::TriangularTerm("NOCHANGE", -10.0, 10.0));
changeHeading->addTerm(new fl::TriangularTerm("NEGSMALL", 0.0, 10.5));
changeHeading->addTerm(new fl::TriangularTerm("NEGMEDIUM", 1.25, 21.25));
changeHeading->addTerm(new fl::TriangularTerm("NEGLARGE", 12.5, 32.5));
engine.addOutputLVar(changeHeading);

fl::RuleBlock* block = new fl::RuleBlock();
//PlaneDist is CLOSE
block->addRule(new fl::MamdaniRule("if PlaneDist is SAFE and OurBearingAngle" &
  " is POSLEAVE then ChangeInHeading is POSSMALL", engine));
block->addRule(new fl::MamdaniRule("if PlaneDist is SAFE and OurBearingAngle" &
  " is POSFIT then ChangeInHeading is NOCHANGE", engine));
block->addRule(new fl::MamdaniRule("if PlaneDist is SAFE and OurBearingAngle" &
  " is POSHEAD then ChangeInHeading is NEGSMALL", engine));
block->addRule(new fl::MamdaniRule("if PlaneDist is SAFE and OurBearingAngle" &
  " is NEGFIT then ChangeInHeading is NOCHANGE", engine));
block->addRule(new fl::MamdaniRule("if PlaneDist is SAFE and OurBearingAngle" &
  " is NEGLEAVE then ChangeInHeading is NEGSMALL", engine));
block->addRule(new fl::MamdaniRule("if PlaneDist is MEDIUM and OurBearingAngle" &
  " is POSLEAVE then ChangeInHeading is POSSMALL", engine));
block->addRule(new fl::MamdaniRule("if PlaneDist is MEDIUM and OurBearingAngle" &
  " is POSFIT then ChangeInHeading is NOCHANGE", engine));
block->addRule(new fl::MamdaniRule("if PlaneDist is MEDIUM and OurBearingAngle" &
  " is POSHEAD then ChangeInHeading is NEGSMALL", engine));
block->addRule(new fl::MamdaniRule("if PlaneDist is MEDIUM and OurBearingAngle" &
  " is NEGFIT then ChangeInHeading is NOCHANGE", engine));
block->addRule(new fl::MamdaniRule("if PlaneDist is MEDIUM and OurBearingAngle" &
  " is NEGLEAVE then ChangeInHeading is NEGSMALL", engine));
"is NEGLAVE then ChangeInHeading is NEGSMA"L", engine));

block->addRule(new fl::MamdaniRule("if PlaneDist is ALERT and OurBearingAngle" &
"is POSLEAVE then ChangeInHeading is NOCHANGE", engine));
block->addRule(new fl::MamdaniRule("if PlaneDist is ALERT and OurBearingAngle" &
"is POSFIT then ChangeInHeading is NEGSMA"L", engine));
block->addRule(new fl::MamdaniRule("if PlaneDist is ALERT and OurBearingAngle" &
"is POSHEAD then ChangeInHeading is NEGLARGE", engine));
block->addRule(new fl::MamdaniRule("if PlaneDist is ALERT and OurBearingAngle" &
"is NEGLEAVE then ChangeInHeading is POSMALL", engine));
block->addRule(new fl::MamdaniRule("if PlaneDist is ALERT and OurBearingAngle" &
"is NEGFIT then ChangeInHeading is POSSMALL", engine));
block->addRule(new fl::MamdaniRule("if PlaneDist is ALERT and OurBearingAngle" &
"is NEGHEAD then ChangeInHeading is NOCHANGE", engine));
block->addRule(new fl::MamdaniRule("if PlaneDist is DANGER and OurBearingAngle" &
"is POSLEAVE then ChangeInHeading is NEGSMA"L", engine));
block->addRule(new fl::MamdaniRule("if PlaneDist is DANGER and OurBearingAngle" &
"is POSFIT then ChangeInHeading is NEGLARGE", engine));
block->addRule(new fl::MamdaniRule("if PlaneDist is DANGER and OurBearingAngle" &
"is POSHEAD then ChangeInHeading is NEGLARGE", engine));
block->addRule(new fl::MamdaniRule("if PlaneDist is DANGER and OurBearingAngle" &
"is NEGLEAVE then ChangeInHeading is POSMALL", engine));
block->addRule(new fl::MamdaniRule("if PlaneDist is DANGER and OurBearingAngle" &
"is NEGFIT then ChangeInHeading is POSMALL", engine));
block->addRule(new fl::MamdaniRule("if PlaneDist is DANGER and OurBearingAngle" &
"is NEGHEAD then ChangeInHeading is NOCHANGE", engine));

engine.addRuleBlock(block);
ourBearingAngle->setInput(ourAngle);
engine.process();
double output = changeHeading->output().defuzzify();
return output;

}

double AU_UAV_ROS::FuzzyLogicController::lowerTurnsHighDists(double ourAngle, double theirAngle){
  fl::FuzzyOperator& op = fl::FuzzyOperator::DefaultFuzzyOperator();
  fl::FuzzyEngine engine("Heading-Change", op);

  fl::InputLVar* ourBearingAngle = new fl::InputLVar("OurBearingAngle");
  ourBearingAngle->addTerm(new fl::ShoulderTerm("VERYNEG", -90.0, -40.0, true));
  ourBearingAngle->addTerm(new fl::TriangularTerm("NEG", -50, -10.0));
  ourBearingAngle->addTerm(new fl::TriangularTerm("LOWNEG", -20.0, 1.0));
  ourBearingAngle->addTerm(new fl::TriangularTerm("LOWPOS", -1.0, 20.0));
  ourBearingAngle->addTerm(new fl::TriangularTerm("POS", 10, 50.0));
  ourBearingAngle->addTerm(new fl::ShoulderTerm("VERYPOS", 40, 90.0, false));
  engine.addInputLVar(ourBearingAngle);

  fl::InputLVar* theirBearingAngle = new fl::InputLVar("TheirBearingAngle");
  theirBearingAngle->addTerm(new fl::ShoulderTerm("VERYNEG", -90.0, -40.0, true));
  theirBearingAngle->addTerm(new fl::TriangularTerm("NEG", -50, -10.0));
  theirBearingAngle->addTerm(new fl::TriangularTerm("LOWNEG", -20.0, 1.0));
  theirBearingAngle->addTerm(new fl::TriangularTerm("LOWPOS", -1.0, 20.0));
  theirBearingAngle->addTerm(new fl::TriangularTerm("POS", 10, 50.0));
  theirBearingAngle->addTerm(new fl::ShoulderTerm("VERYPOS", 40, 90.0, false));
  engine.addInputLVar(theirBearingAngle);
fl::OutputLVar* changeHeading = new fl::OutputLVar("ChangeInHeading");
changeHeading->addTerm(new fl::TriangularTerm("VERYLEFT", -15.0, 10.0));
changeHeading->addTerm(new fl::TriangularTerm("LEFT", -15.0, -1.0));
changeHeading->addTerm(new fl::TriangularTerm("SLIGHTLEFT", -5.0, 0.0));
changeHeading->addTerm(new fl::TriangularTerm("NOCHANGE", -3.0, 3.0));
changeHeading->addTerm(new fl::TriangularTerm("SLIGHTRIGHT", 0.0, 5.0));
changeHeading->addTerm(new fl::TriangularTerm("RIGHT", 1.0, 15.0));
changeHeading->addTerm(new fl::TriangularTerm("VERYRIGHT", 10.0, 15.0));
engine.addOutputLVar(changeHeading);

fl::RuleBlock* block = new fl::RuleBlock();
// PlaneDist is CLOSE
block->addRule(new fl::MamdaniRule("if OurBearingAngle is VERYNEG and TheirBearingAngle &
  is VERYNEG then ChangeInHeading is NOCHANGE", engine));
block->addRule(new fl::MamdaniRule("if OurBearingAngle is VERYNEG and TheirBearingAngle &
  is NEG then ChangeInHeading is NOCHANGE", engine));
block->addRule(new fl::MamdaniRule("if OurBearingAngle is VERYNEG and TheirBearingAngle &
  is LOWNEG then ChangeInHeading is SLIGHTRIGHT", engine));
block->addRule(new fl::MamdaniRule("if OurBearingAngle is VERYNEG and TheirBearingAngle &
  is LOWPOS then ChangeInHeading is RIGHT", engine));
block->addRule(new fl::MamdaniRule("if OurBearingAngle is VERYNEG and TheirBearingAngle &
  is POS then ChangeInHeading is RIGHT", engine));
block->addRule(new fl::MamdaniRule("if OurBearingAngle is VERYNEG and TheirBearingAngle &
  is VERYPOS then ChangeInHeading is RIGHT", engine));

block->addRule(new fl::MamdaniRule("if OurBearingAngle is NEG and TheirBearingAngle &
  is VERYNEG then ChangeInHeading is NOCHANGE", engine));
block->addRule(new fl::MamdaniRule("if OurBearingAngle is NEG and TheirBearingAngle &
  is NEG then ChangeInHeading is NOCHANGE", engine));
block->addRule(new fl::MamdaniRule("if OurBearingAngle is NEG and TheirBearingAngle &
  is LOWNEG then ChangeInHeading is SLIGHTRIGHT", engine));
block->addRule(new fl::MamdaniRule("if OurBearingAngle is NEG and TheirBearingAngle &
  is LOWPOS then ChangeInHeading is VERYRIGHT", engine));
block->addRule(new fl::MamdaniRule("if OurBearingAngle is NEG and TheirBearingAngle &
  is POS then ChangeInHeading is VERYRIGHT", engine));
block->addRule(new fl::MamdaniRule("if OurBearingAngle is NEG and TheirBearingAngle &
  is VERYPOS then ChangeInHeading is VERYRIGHT", engine));

block->addRule(new fl::MamdaniRule("if OurBearingAngle is LOWNEG and TheirBearingAngle &
  is VERYNEG then ChangeInHeading is SLIGHTRIGHT", engine));
block->addRule(new fl::MamdaniRule("if OurBearingAngle is LOWNEG and TheirBearingAngle &
  is NEG then ChangeInHeading is SLIGHTRIGHT", engine));
block->addRule(new fl::MamdaniRule("if OurBearingAngle is LOWNEG and TheirBearingAngle &
  is LOWNEG then ChangeInHeading is VERYRIGHT", engine));
block->addRule(new fl::MamdaniRule("if OurBearingAngle is LOWNEG and TheirBearingAngle &
  is LOWPOS then ChangeInHeading is VERYLEFT", engine));
block->addRule(new fl::MamdaniRule("if OurBearingAngle is LOWNEG and TheirBearingAngle &
  is POS then ChangeInHeading is VERYLEFT", engine));
block->addRule(new fl::MamdaniRule("if OurBearingAngle is LOWPOS and TheirBearingAngle &
  is VERYNEG then ChangeInHeading is VERYRIGHT", engine));
block->addRule(new fl::MamdaniRule("if OurBearingAngle is LOWPOS and TheirBearingAngle &
  is NEG then ChangeInHeading is VERYRIGHT", engine));
block->addRule(new fl::MamdaniRule("if OurBearingAngle is LOWPOS and TheirBearingAngle &
  is LOWNEG then ChangeInHeading is VERYRIGHT", engine));
block->addRule(new fl::MamdaniRule("if OurBearingAngle is LOWPOS and TheirBearingAngle" &
"is LOWPOS then ChangeInHeading is VERYLEFT", engine));
block->addRule(new fl::MamdaniRule("if OurBearingAngle is LOWPOS and TheirBearingAngle" &
"is POS then ChangeInHeading is SLIGHTLEFT", engine));
block->addRule(new fl::MamdaniRule("if OurBearingAngle is LOWPOS and TheirBearingAngle" &
"is VERYPOS then ChangeInHeading is SLIGHTLEFT", engine));
block->addRule(new fl::MamdaniRule("if OurBearingAngle is POS and TheirBearingAngle" &
"is VERYNEG then ChangeInHeading is VERYLEFT", engine));
block->addRule(new fl::MamdaniRule("if OurBearingAngle is POS and TheirBearingAngle" &
"is NEG then ChangeInHeading is LEFT", engine));
block->addRule(new fl::MamdaniRule("if OurBearingAngle is POS and TheirBearingAngle" &
"is LOWNEG then ChangeInHeading is LEFT", engine));
block->addRule(new fl::MamdaniRule("if OurBearingAngle is POS and TheirBearingAngle" &
"is LOWPOS then ChangeInHeading is SLIGHTLEFT", engine));
block->addRule(new fl::MamdaniRule("if OurBearingAngle is POS and TheirBearingAngle" &
"is POS then ChangeInHeading is NOCHANGE", engine));
block->addRule(new fl::MamdaniRule("if OurBearingAngle is POS and TheirBearingAngle" &
"is VERYPOS then ChangeInHeading is NOCHANGE", engine));
block->addRule(new fl::MamdaniRule("if OurBearingAngle is VERYPOS and TheirBearingAngle" &
"is VERYNEG then ChangeInHeading is LEFT", engine));
block->addRule(new fl::MamdaniRule("if OurBearingAngle is VERYPOS and TheirBearingAngle" &
"is NEG then ChangeInHeading is LEFT", engine));
block->addRule(new fl::MamdaniRule("if OurBearingAngle is VERYPOS and TheirBearingAngle" &
"is LOWNEG then ChangeInHeading is LEFT", engine));
block->addRule(new fl::MamdaniRule("if OurBearingAngle is VERYPOS and TheirBearingAngle" &
"is LOWPOS then ChangeInHeading is SLIGHTLEFT", engine));
block->addRule(new fl::MamdaniRule("if OurBearingAngle is VERYPOS and TheirBearingAngle" &
"is POS then ChangeInHeading is NOCHANGE", engine));
block->addRule(new fl::MamdaniRule("if OurBearingAngle is VERYPOS and TheirBearingAngle" &
"is VERYPOS then ChangeInHeading is NOCHANGE", engine));

engine.addRuleBlock(block);
ourBearingAngle->setInput(ourAngle);
theirBearingAngle->setInput(theirAngle);
engine.process();
double output = changeHeading->output().defuzzify();
return output;
}

double AU_UAV_ROS::FuzzyLogicController::semiLowerTurnsHighDists(double ourAngle, double theirAngle){
fl::FuzzyOperator& op = fl::FuzzyOperator::DefaultFuzzyOperator();
fl::FuzzyEngine engine("Heading-Change", op);

fl::InputLVar* ourBearingAngle = new fl::InputLVar("OurBearingAngle");
ourBearingAngle->addTerm(new fl::ShoulderTerm("VERYNEG", -90.0, -40.0, true));
ourBearingAngle->addTerm(new fl::TriangularTerm("NEG", -50, -10.0));
ourBearingAngle->addTerm(new fl::TriangularTerm("LOWNEG", -20.0, 1.0));
ourBearingAngle->addTerm(new fl::TriangularTerm("LOWPOS", -1.0, 20.0));
ourBearingAngle->addTerm(new fl::TriangularTerm("POS", 10, 50.0));
ourBearingAngle->addTerm(new fl::ShoulderTerm("VERYPOS", 40, 90.0, false));
engine.addInputLVar(ourBearingAngle);

fl::InputLVar* theirBearingAngle = new fl::InputLVar("TheirBearingAngle");
theirBearingAngle->addTerm(new fl::ShoulderTerm("VERYNEG", -90.0, -40.0, true));
theirBearingAngle->addTerm(new fl::TriangularTerm("NEG", -50, -10.0));
theirBearingAngle->addTerm(new fl::TriangularTerm("LOWNEG", -20.0, 1.0));
theirBearingAngle->addTerm(new fl::TriangularTerm("LOWPOS", -1.0, 20.0));
theirBearingAngle->addTerm(new fl::TriangularTerm("POS", 10, 50.0));
theirBearingAngle->addTerm(new fl::ShoulderTerm("VERYPOS", 40, 90.0, false));
engine.addInputLVar(theirBearingAngle);

fl::OutputLVar* changeHeading = new fl::OutputLVar("ChangeInHeading");
changeHeading->addTerm(new fl::TriangularTerm("VERYLEFT", -15.0, 10.0));
changeHeading->addTerm(new fl::TriangularTerm("LEFT", -15.0, -1.0));
changeHeading->addTerm(new fl::TriangularTerm("SLIGHTLEFT", -5.0, 0.0));
changeHeading->addTerm(new fl::TriangularTerm("NOCHANGE", -3.0, 3.0));
changeHeading->addTerm(new fl::TriangularTerm("SLIGHTRIGHT", 0.0, 5.0));
changeHeading->addTerm(new fl::TriangularTerm("RIGHT", 1.0, 15.0));
changeHeading->addTerm(new fl::TriangularTerm("VERYRIGHT", 10.0, 15.0));
engine.addOutputLVar(changeHeading);

fl::RuleBlock* block = new fl::RuleBlock();

//PlaneDist is CLOSE
block->addRule(new fl::MamdaniRule("if OurBearingAngle is VERYNEG and TheirBearingAngle is VERYNEG and ChangeInHeading is NOCHANGE", engine));
block->addRule(new fl::MamdaniRule("if OurBearingAngle is VERYNEG and TheirBearingAngle is NEG and ChangeInHeading is NOCHANGE", engine));
block->addRule(new fl::MamdaniRule("if OurBearingAngle is VERYNEG and TheirBearingAngle is LOWNEG and ChangeInHeading is SLIGHTRIGHT", engine));
block->addRule(new fl::MamdaniRule("if OurBearingAngle is VERYNEG and TheirBearingAngle is LOWPOS and ChangeInHeading is RIGHT", engine));
block->addRule(new fl::MamdaniRule("if OurBearingAngle is VERYNEG and TheirBearingAngle is POS and ChangeInHeading is RIGHT", engine));
block->addRule(new fl::MamdaniRule("if OurBearingAngle is VERYNEG and TheirBearingAngle is VERYPOS and ChangeInHeading is RIGHT", engine));
block->addRule(new fl::MamdaniRule("if OurBearingAngle is NEG and TheirBearingAngle is VERYNEG and ChangeInHeading is NOCHANGE", engine));
block->addRule(new fl::MamdaniRule("if OurBearingAngle is NEG and TheirBearingAngle is NEG and ChangeInHeading is NOCHANGE", engine));
block->addRule(new fl::MamdaniRule("if OurBearingAngle is NEG and TheirBearingAngle is LOWNEG and ChangeInHeading is SLIGHTRIGHT", engine));
block->addRule(new fl::MamdaniRule("if OurBearingAngle is NEG and TheirBearingAngle is LOWPOS and ChangeInHeading is VERYRIGHT", engine));
block->addRule(new fl::MamdaniRule("if OurBearingAngle is NEG and TheirBearingAngle is POS and ChangeInHeading is VERYRIGHT", engine));
block->addRule(new fl::MamdaniRule("if OurBearingAngle is NEG and TheirBearingAngle is VERYPOS and ChangeInHeading is VERYRIGHT", engine));
block->addRule(new fl::MamdaniRule("if OurBearingAngle is LOWNEG and TheirBearingAngle is VERYNEG and ChangeInHeading is SLIGHTLEFT", engine));
block->addRule(new fl::MamdaniRule("if OurBearingAngle is LOWNEG and TheirBearingAngle is NEG and ChangeInHeading is SLIGHTLEFT", engine));
block->addRule(new fl::MamdaniRule("if OurBearingAngle is LOWNEG and TheirBearingAngle is LOWPOS and ChangeInHeading is LEFT", engine));
block->addRule(new fl::MamdaniRule("if OurBearingAngle is LOWNEG and TheirBearingAngle is POS and ChangeInHeading is LEFT", engine));
block->addRule(new fl::MamdaniRule("if OurBearingAngle is LOWNEG and TheirBearingAngle is VERYPOS and ChangeInHeading is LEFT", engine));
8.2 Collision Avoidance Node

This code is designed as a node to be used in the simulator designed by Holt et al. Uses basic math to decide whether to enter a collision avoidance maneuver and then uses the fuzzy logic system to determine the change in heading.
collisionAvoidance

This file receives telemetry updates from all UAVs and sends waypoints to these UAVs based on Fuzzy Logic Collision Avoidance Algorithm

Authors: J. Matthew Holt, Michelle Hromatka, Jeffrey West

/*

//standard C++ headers
#include <sstream>
#include <stdlib.h>
#include <time.h>
#include <map>
#include <iostream>
#include <fstream>
#include <sstream>

//ROS, FL, Dubins headers
#include "ros/ros.h"
#include "AU_UAV_ROS/TelemetryUpdate.h"
#include "AU_UAV_ROS/GoToWaypoint.h"
#include "AU_UAV_ROS/RequestWaypointInfo.h"
#include "AU_UAV_ROS/standardFuncs.h"
#include "AU_UAV_ROS/FuzzyLogicController.h"
#include "AU_UAV_ROS/dubins.h"
#include "AU_UAV_ROS/FuzzyLogicOne.h"
#include "AU_UAV_ROS/OursAndTheirs2.h"
#include "AU_UAV_ROS/OTWOWBHtighter.h"

#define rho 15/(22.5*(M_PI/180.0))
#define TIMESTEP 1
#define UAV_AIRSPEED 11.176

//collisionAvoidance does not have a header file, define several methods / variables here
AU_UAV_ROS::OursAndTheirs2* fuzzyLogic;
AU_UAV_ROS::OTWOWBHtighter* OTWOWBHtighter;

//ROS service clients for calling a service from the coordinator
ros::ServiceClient goToWaypointClient;
ros::ServiceClient requestInfoClient;

//initialize some cool variables
std::map<int,AU_UAV_ROS::PlanePose> planeMap;
AU_UAV_ROS::FuzzyLogicController fuzzy1;
std::ofstream myfile;
double distToWP;
double currentHeading = 0.0;
double fuzzyHeading = 0.0;
AU_UAV_ROS::fuzzyParams fuzzyParams;

//Initialize deez pointers
AU_UAV_ROS::PlanePose* currentUAV;
AU_UAV_ROS::PlanePose* oldUAV;
AU_UAV_ROS::PlanePose* closestUAV;

//this function is run everytime new telemetry information from any plane is recieved
void telemetryCallback(const AU_UAV_ROS::TelemetryUpdate::ConstPtr& msg)
{
    AU_UAV_ROS::waypoint planeLatLongAlt;
    planeLatLongAlt.longitude = msg->currentLongitude;
    planeLatLongAlt.latitude = msg->currentLatitude;
    planeLatLongAlt.altitude = msg->currentAltitude;
    AU_UAV_ROS::position currentPose = getXYZ(planeLatLongAlt);

    //If planeMap is empty, add current telemetry information
    if(planeMap.count(msg->planeID)==0)
    {
        currentUAV = new AU_UAV_ROS::PlanePose;
        currentUAV->setID(msg->planeID);
        currentUAV->setX(currentPose.x_coordinate);
        currentUAV->setY(currentPose.y_coordinate);
        currentUAV->setZ(currentPose.altitude);
        currentUAV->setVelocity(11.176);
        currentUAV->setHeading(0.0);
        planeMap[msg->planeID] = *currentUAV;
        delete currentUAV;
    }

    //After initial planeMap creation, don't update planeMap until planes have been sent a new Waypoint
    else if (planeMap.count(msg->planeID)!=0 && msg->currentWaypointIndex == -1)
    {
        /* Don't do anything here!
        This function is super important, because the UAVs should
        not do anything until waypoints are received */
    }
    else

    /*----------------------------------------------------------------------------------*/
    //After initial map update, we can now update things like heading during second time through
    currentUAV = new AU_UAV_ROS::PlanePose;
    currentUAV->setID(msg->planeID);
    currentUAV->setX(currentPose.x_coordinate);
    currentUAV->setY(currentPose.y_coordinate);
    currentUAV->setZ(currentPose.altitude);
    oldUAV = &planeMap.find(msg->planeID)->second;
    double currentHeading = getNewHeading(oldUAV->getPosition(), currentUAV->getPosition());
    currentUAV->setHeading(currentHeading);

    /*----------------------------------------------------------------------------------*/
    //Find the Closest UAV and the distance between the two
    int closestPlane = getClosestPlane(msg->planeID , planeMap);
    closestUAV = &planeMap.find(closestPlane)->second;
    double distBtwnPlanes = getDist(currentUAV->getPosition(), closestUAV->getPosition());

    /*----------------------------------------------------------------------------------*/
    AU_UAV_ROS::waypoint nextWaypoint;
    //ENTER Collision Avoidance Mode:
    if ((closestPlane != -1) && (distBtwnPlanes < 100)) {
        //grab necessary parameters for the Fuzzy Logic:
        fuzzyParams = getFuzzyParams(currentUAV, closestUAV);
        //Process Fuzzy Logic to return a change in heading (fuzzyHeading)
        fuzzyHeading = fuzzyLogic->process(distBtwnPlanes, fuzzyParams.ourBearingAngle, fuzzyP
double nextHeadingForReal = fuzzyHeading + currentUAV->getHeading();
//use this heading and find a waypoint (lat, long, alt) to send
nextWaypoint = getCAWaypoint(nextHeadingForReal, currentPose);

//Did not enter Collision Avoidance Mode:
else {
    //Find the next goal waypoint
    AU_UAV_ROS::waypoint nextGoal;
    AU_UAV_ROS::RequestWaypointInfo srv;
    srv.request.planeID = msg->planeID;
    srv.request.isAvoidanceWaypoint = false;
    srv.request.posInQueue = 0;
    if (requestInfoClient.call(srv)){
        //ROS_INFO("clean");
    } else{
        ROS_INFO("error");
    }
    nextGoal.latitude = srv.response.latitude;
    nextGoal.longitude = srv.response.longitude;
    nextGoal.altitude = srv.response.altitude;
    //Go to the next goal waypoint
    nextWaypoint = nextGoal;
}

/*----------------------------------------------------------------------------------*/
//go to nextWaypoint everytime
//service request to go to the waypoint determined by fuzzy logicness OR normal waypoint
AU_UAV_ROS::GoToWaypoint gotosrv;
gotosrv.request.planeID = msg->planeID;
gotosrv.request.latitude = nextWaypoint.latitude;
gotosrv.request.longitude = nextWaypoint.longitude;
gotosrv.request.isAvoidanceManeuver = true;
gotosrv.request.isNewQueue = true;
if (goToWaypointClient.call(gotosrv)){
    //ROS_INFO("clean");
} else{
    ROS_ERROR("error");
}
//update map with current Plane Pose, Heading (currentUAV) everytime
planeMap[msg->planeID] = *currentUAV;
delete currentUAV;
}

int main(int argc, char **argv)
{
    //standard ROS startup
    ros::init(argc, argv, "collisionAvoidance");
    ros::NodeHandle n;
    //subscribe to telemetry outputs and create client for the avoid collision service
    ros::Subscriber sub = n.subscribe("telemetry", 1000, telemetryCallback);
    goToWaypointClient = n.serviceClient<AU_UAV_ROS::GoToWaypoint>("go_to_waypoint");
    requestInfoClient = n.serviceClient<AU_UAV_ROS::RequestWaypointInfo>("request_waypoint_info");
}
//create Fuzzy Logic Controllers for use in telemetryCallback
fuzzyLogic = new AU_UAV_ROS::OursAndTheirs2();

//random seed for if statement in telemetryCallback, remove when collision avoidance work begins
srand(time(NULL));

//needed for ROS to wait for callbacks
ros::spin();

return 0;
}

8.3 Standard Functions

These are the helper functions used throughout the collision avoidance node, functions such as computing the distance between planes, heading or bearing angle between planes.

/*
Implementation of standardFuncs.h. For information on how to use these functions, visit standardFuncs.h. Comments in this file are related to implementation, not usage.
*/

#include <cmath>
#include <algorithm>
#include <stdlib.h>
#include <ros/ros.h>
#include "AU_UAV_ROS/standardFuncs.h"
#include "AU_UAV_ROS/standardDefs.h"
#include "AU_UAV_ROS/PlanePose.h"
#include "AU_UAV_ROS/FuzzyLogicController.h"
#include "fuzzylite/FuzzyLite.h"
#include <map>
#define WEST_MOST_LONGITUDE -85.490356
#define NORTH_MOST_LATITUDE 32.606573
#define EARTH_RADIUS 6371000
#define METER_TO_LATITUDE (1.0/110897.21)
#define METER_TO_LONGITUDE (1.0/93865.7257)
#define LATITUDE_TO_METERS (111200.0)
#define METERS_TO_LATITUDE (1.0/111200.0)
#define DEGREES_TO_RADIANS (M_PI/180.0)
#define RADIANS_TO_DEGREES (180.0/M_PI)
#define MPS_SPEED 11.176
#define MPH_SPEED 25

AU_UAV_ROS::PlanePose* thisUAV;
AU_UAV_ROS::PlanePose* thatUAV;

AU_UAV_ROS::fuzzyParams getFuzzyParams(AU_UAV_ROS::PlanePose* currentUAV, AU_UAV_ROS::PlanePose* currentUAV, ...
{ AU_UAV_ROS::fuzzyParams fuzzyParams;
    fuzzyParams.ourBearingAngle = getBearingAngle(*currentUAV, *closestUAV);
    fuzzyParams.theirBearingAngle = getBearingAngle(*closestUAV, *currentUAV);

    return fuzzyParams;
}

//This function is passed a heading value returned by our fuzzy logic engine
//The function returns a waypoint to pass to the simulator
AU_UAV_ROS::waypoint getCAWaypoint(double fuzzyHeading, AU_UAV_ROS::position currentPose)
{
    AU_UAV_ROS::position NWPmeters;
    NWPmeters.x_coordinate = currentPose.x_coordinate + 30.0*sin(fuzzyHeading*DEGREES_TO_RADIANS);
    NWPmeters.y_coordinate = currentPose.y_coordinate + 30.0*cos(fuzzyHeading*DEGREES_TO_RADIANS);
    NWPmeters.altitude = currentPose.altitude;

    AU_UAV_ROS::waypoint nextWaypoint = convertPositionToWaypoint(NWPmeters);
    return nextWaypoint;
}

AU_UAV_ROS::waypoint convertPositionToWaypoint(AU_UAV_ROS::position position)
{
    double deltaX = position.x_coordinate;
    double deltaY = position.y_coordinate;
    AU_UAV_ROS::waypoint waypoint;

    //Take x and y and convert back to lat long
    waypoint.longitude = WEST_MOST_LONGITUDE + (deltaX*METER_TO_LONGITUDE);
    waypoint.latitude = NORTH_MOST_LATITUDE + (deltaY*METER_TO_LATITUDE);
    waypoint.altitude = position.altitude;

    return waypoint;
}

//get closest plane
//eventually, it will be useful to make this function return closest plane with an imminent collision!
int getClosestPlane(int planeID, std::map<int,AU_UAV_ROS::PlanePose> planeMap)
{
    int closestPlane = -1;
    double smallestDist = std::numeric_limits<double>::infinity();

    for (int i = 0; i < planeMap.size(); i++) {
        double dist = getDist(planeMap.find(planeID)->second.getPosition(), planeMap.find(i)->second.getPosition());
        if (dist<smallestDist)
        {
            smallestDist = dist;
            closestPlane = i;
        }
    }
    return closestPlane;
}
return closestPlane;
}

//return distance (in meters) between two AU_UAV_ROS::position variables given in meters
double getDist(AU_UAV_ROS::position first, AU_UAV_ROS::position second)
{
    double dist = sqrt(pow((first.x_coordinate - second.x_coordinate),2)+pow((first.y_coordinate - second.y_coordinate),2)+pow((first.altitude - second.altitude),2));
    return dist;
}

//This function will return XYZ coordinates for any waypoint
//(usually used to find a plane's position)
//(the same coordinates being published to the RVIZ simulator)
//(This function DOES take the earth's curvature into consideration)
//The function uses the "getActualDistance" function with the top/left-most point and the plane's lat/lng/altitude
AU_UAV_ROS::position getXYZ(AU_UAV_ROS::waypoint planePose)
{
    AU_UAV_ROS::waypoint origin;
    origin.longitude=WEST_MOST_LONGITUDE;
    origin.latitude=NORTH_MOST_LATITUDE;
    origin.altitude=0.0;

    AU_UAV_ROS::waypoint northsouthpoint;
    northsouthpoint.latitude=planePose.latitude;
    northsouthpoint.longitude=WEST_MOST_LONGITUDE;

    AU_UAV_ROS::waypoint eastwestpoint;
    eastwestpoint.latitude=NORTH_MOST_LATITUDE;
    eastwestpoint.longitude=planePose.longitude;

    AU_UAV_ROS::position planeXYZ;
    planeXYZ.x_coordinate = (eastwestpoint.longitude - origin.longitude)/METER_TO_LONGITUDE;
    planeXYZ.y_coordinate = (northsouthpoint.latitude - origin.latitude)/METER_TO_LATITUDE;
    planeXYZ.altitude = planePose.altitude;
    return planeXYZ;
}

//This function will return the actual distance between two points in space (lat/long/alt) in meters
//This function DOES take the earth's curvature into consideration
double getActualDistance(AU_UAV_ROS::waypoint first, AU_UAV_ROS::waypoint second)
{
    double deltaLat = first.latitude - second.latitude;
    double deltaLong = first.longitude - second.longitude;
    double deltaY = deltaLat*110897.4592048873;
    double deltaX = deltaLong*93865.73571034615;
    double dist = sqrt(pow(deltaX,2) + pow(deltaY,2));
    return dist;
/* this is Matt's method which we decided not to use:
difference in latitudes in radians
double lat1 = first.latitude*DEGREES_TO_RADIANS;
double lat2 = second.latitude*DEGREES_TO_RADIANS;
double long1 = first.longitude*DEGREES_TO_RADIANS;
double long2 = second.longitude*DEGREES_TO_RADIANS;
double deltaLat = lat2 - lat1;
double deltaLong = long2 - long1;
//haversine crazy math, should probably be verified further beyond basic testing
//calculate distance from current position to destination
double a = pow(sin(deltaLat / 2.0), 2);
a = a + cos(lat1)*cos(lat2)*pow(sin(deltaLong / 2.0), 2);
a = 2.0 * asin(sqrt(a));
return EARTH_RADIUS * a;
*/

//This will take two waypoints and measure the heading between them (based on position)
//This is an estimation of plane heading based on the position heading from a point a time t and time t+1
//waypoints must be in meters
//A zero degree heading points directly North (and East is 90 degrees and West is -90 degrees to keep in [−180,180] range)
double getNewHeading(AU_UAV_ROS::position first, AU_UAV_ROS::position second)
{
    double deltaX = second.x_coordinate - first.x_coordinate;
double deltaY = second.y_coordinate - first.y_coordinate;
double heading = atan2(deltaX,deltaY);
heading = (heading*RADIANS_TO_DEGREES);
return heading;
}

double getBearingAngle(AU_UAV_ROS::PlanePose first, AU_UAV_ROS::PlanePose second)
{
    //grab the (x, y, alt) values from each PlanePose object
    AU_UAV_ROS::position planePose1 = first.getPosition();
    AU_UAV_ROS::position planePose2 = second.getPosition();
double theta = first.getHeading();
    //AU_UAV_ROS::position midpoint = getMidpoint(planePose1, planePose2);
    //get phi for plane 1 only, or the angle between North and the segment connecting the two planes
    double phi = getNewHeading(planePose1, planePose2);
double bearingAngle = phi - theta;
bearingAngle = manipulateAngle(bearingAngle);
return bearingAngle;
}

//return the midpoint between two points in three space
AU_UAV_ROS::position getMidpoint(AU_UAV_ROS::position planePose1, AU_UAV_ROS::position planePose2)
{
    AU_UAV_ROS::position midpoint;
    midpoint.x_coordinate = ((planePose1.x_coordinate - planePose2.x_coordinate)/2)+planePose2.x_coordinate;
}
\[
\text{midpoint.y-coordinate} = \frac{(\text{planePose1.y-coordinate} - \text{planePose2.y-coordinate})}{2} + \text{planePose2.y-coordinate}; \\
\text{midpoint.altitude} = \frac{(\text{planePose1.altitude} - \text{planePose2.altitude})}{2} + \text{planePose2.altitude};
\]

```
return midpoint;
```

// The rest of the functions in this file were written by the APF team in 2011 REU
// They may or may not be used in our algorithm

/* Modify the angle so that it remains on the interval [-180, 180] */

```
double manipulateAngle(double angle){
    while (angle > 180){
        /* decrease angle by one 360 degree cycle */
        angle -= 360;
    }

    while (angle < -180){
        /* increase angle by one 360 degree cycle */
        angle += 360;
    }

    if (angle == -0){
        ROS_INFO("just checking");
        /* increase angle by one 360 degree cycle */
        angle = 0;
    }
    return angle;
}
```

/* Returns the Cardinal angle between two points of latitude and longitude in degrees. The starting point is given by lat1 and long1 (the first two parameters), and the final point is given by lat2 and long2 (the final two parameters). The value returned is on the interval [-180, 180]. */

```
double findAngle(double lat1, double long1, double lat2, double long2){
    double lonDiff = 0.0, angle = 0.0;
    double x = 0.0, y = 0.0;

    /* Convert latitudes to radians */
    lat2 *= DEGREE_TO_RAD;
    lat1 = lat1 * DEGREE_TO_RAD;

    lonDiff = (long2 - long1) * DEGREE_TO_RAD; /* convert difference in longitude to radians */

    /* Haversine math: see http://www.movable-type.co.uk/scripts/latlong.html for more information */
    y = sin(lonDiff) * cos(lat2);
    x = cos(lat1) * sin(lat2) - sin(lat1) * cos(lat2) * cos(lonDiff);

    angle = atan2(y, x) * 180/PI;

    // Angle will be in degrees.
```
Given a waypoint (latitude, longitude, and altitude) as well as the bearing and angular distance to travel, calculateCoordinate will return the new location in the form of a waypoint.

```
AU_UAV_ROS::waypoint calculateCoordinate(AU_UAV_ROS::waypoint currentPosition, double bearing, double distance) {
    // Calculate final latitude and longitude; see movable-type.co.uk/scripts/latlong.html for more details.
    bearing *= DEGREES_TO_RADIANS; // convert angle of force to radians
    double lat1 = currentPosition.latitude * DEGREES_TO_RADIANS; // lat1 = current latitude in radians
    double dLat = distance * cos(bearing); // calculate change in latitude
    double lat2 = lat1 + dLat; // calculate final latitude
    double dPhi = log(tan(lat2/2 + PI/4) / tan(lat1/2 + PI/4)); // East-West line gives dPhi=0
    double dLon = distance * sin(bearing) / dPhi; // calculate change in longitude

    // check for some daft bugger going past the pole, normalise latitude if so
    if (abs(lat2) > PI/2) {
        lat2 = lat2 > 0 ? PI - lat2 : -(PI - lat2);
    }
    double lon2 = (currentPosition.longitude * DEGREES_TO_RADIANS + dLon) * RADIANS_TO_DEGREES; // wrap around if necessary to ensure final longitude is on the interval [-180, 180]
    lon2 = manipulateAngle(lon2);
    lat2 *= RADIANS_TO_DEGREES; // convert final latitude to degrees

    AU_UAV_ROS::waypoint coordinate;
    coordinate.latitude = lat2;
    coordinate.longitude = lon2;
    coordinate.altitude = currentPosition.altitude;
    return coordinate;
}
```

/* Convert Cardinal direction to an angle in the Cartesian plane */
```
double toCartesian(double UAVBearing) {
    UAVBearing = manipulateAngle(UAVBearing); // get angle on the interval [-180, 180]
    if (UAVBearing < 180 && UAVBearing >= 0) /* UAV bearing is in the first or fourth quadrant */
        return 90 - UAVBearing;
    else if (UAVBearing < 0 && UAVBearing >= -90) /* UAV bearing is in the second quadrant */
        return -1 * UAVBearing + 90;
    else if (UAVBearing < -90 && UAVBearing > -180) /* UAV bearing is in the third quadrant */
        return -1 * (UAVBearing + 180) - 90;
    else if (UAVBearing == 180 || UAVBearing == -180)
        return -90;
    else
        return -999; /* should never happen in current setup */
}
```

/* Convert angle in the Cartesian plane to a Cardinal direction */
```
double toCardinal(double angle) {
    angle = manipulateAngle(angle); // get angle on the interval [-180, 180]
    ROS_INFO("angle %f", angle);
    return angle;
}
```
if (angle <= 90 && angle >= -90) /* angle is in the first or fourth quadrant */
    return 90 - angle;
else if (angle >= 90 && angle <= 180) /* angle is in the second quadrant */
    return -1*angle + 90;
else if (angle <= -90 && angle >= -180) /* angle is in third quadrant */
    return -180 + -1*(90 + angle);
else
    return -999; /* should never happen in current setup */
}

//The rest of the functions were originally meant to be used in the first fuzzy logic controller, but

/*
Returns the distance between two points of latitude and longitude in meters. The first two parameters
are the latitude and longitude of the starting point, and the last two parameters are the latitude and
longitude of the ending point.
*/

double findDistance(double lat1, double long1, double lat2, double long2){
    double latDiff = 0.0, lonDiff = 0.0;
    double squareHalfChord = 0.0, angularDistance = 0.0;

    /* Get difference in radians */
    latDiff = (lat1 - lat2)*DEGREE_TO_RAD;
    lonDiff = (long2 - long1)*DEGREE_TO_RAD;

    /* Find the square of half of the chord length between the two points */
    /* sin(lat difference / 2)^2 + cos(lat1) * cos(lat2)*sin(lon difference / 2)^2 */
    squareHalfChord = pow(sin(latDiff / 2), 2) +
    pow(sin(lonDiff / 2), 2) *
    cos(lat1 * DEGREE_TO_RAD) *
    cos(lat2 * DEGREE_TO_RAD);

    /* Calculate the angular distance in radians */
    /* 2 * arctan(sqrt(squareHalfChord), sqrt(1 - squareHalfChord)) */
    angularDistance = 2 * atan2(sqrt(squareHalfChord),
    sqrt(1 - squareHalfChord));

    /* Return result in kilometers */
    return angularDistance * EARTH_RADIUS;
}

//This function will return the minimum distance to collision or the min(A,B)
//does NOT work in three space yet, whatever.

double getDistanceToCollision(AU_UAV_ROS::PlanePose first, AU_UAV_ROS::PlanePose second)
{
    AU_UAV_ROS::position collisionPoint = getTwoPlanesIntersect(first, second);

    AU_UAV_ROS::position planePose1 = first.getPosition();
    AU_UAV_ROS::position planePose2 = second.getPosition();

    double aValue = sqrt(pow((collisionPoint.x_coordinate - planePose1.x_coordinate),2)+pow((collisionPoint.y_coordinate - planePose1.y_coordinate),2));
    double bValue = sqrt(pow((collisionPoint.x_coordinate - planePose2.x_coordinate),2)+pow((collisionPoint.y_coordinate - planePose2.y_coordinate),2));

    if (aValue > bValue)
    {
        return bValue;
    }
else {
  return aValue;
}
}

// This function will take two plane positions and returns the intersection of the lines produced by each plane's respective heading. This point of intersection is the location of a possible intersection.
// It will expect the planes' positions to be in meters and heading to be in degrees.
AU_UAV_ROS::position getTwoPlanesIntersect(AU_UAV_ROS::PlanePose first, AU_UAV_ROS::PlanePose second) {
  double planeHeading1 = first.getHeading() * DEGREES_TO_RADIANS;
  double planeHeading2 = second.getHeading() * DEGREES_TO_RADIANS;
  // Convert planeHeading to "m" here where m is the slope of the line in the X-Y plane
  planeHeading1 = 1 / tan(planeHeading1);
  planeHeading2 = 1 / tan(planeHeading2);
  // Grab the (x, y, alt) values from each PlanePose object
  AU_UAV_ROS::position planePose1 = first.getPosition();
  AU_UAV_ROS::position planePose2 = second.getPosition();
  double x_coordinate = ((planeHeading1 * planePose1.x_coordinate) - (planeHeading2 * planePose2.x_coordinate) + planePose2.y_coordinate) / (planeHeading1 - planeHeading2);
  double y_coordinate = planePose1.y_coordinate + planeHeading1 * (x_coordinate - planePose1.x_coordinate);
  // CollisionPoint returned in meters
  AU_UAV_ROS::position collisionPoint;
  collisionPoint.x_coordinate = x_coordinate;
  collisionPoint.y_coordinate = y_coordinate;
  collisionPoint.altitude = 0.0;
  return collisionPoint;
}

// This function will take two plane positions and find the difference in the distance away from the two's shared collision point.
// For example, if plane1 is 10 meters away from the collision point where plane1 and plane2 would crash and plane2 is 15 meters away from the same collision point, this function returns -5 meters.
double getOverlapDistance(AU_UAV_ROS::PlanePose first, AU_UAV_ROS::PlanePose second) {
  AU_UAV_ROS::position collisionPoint = getTwoPlanesIntersect(first, second);
  // Grab the (x, y, alt) values from each PlanePose object
  AU_UAV_ROS::position planePose1 = first.getPosition();
  AU_UAV_ROS::position planePose2 = second.getPosition();
  double overlapDistance = sqrt(pow((collisionPoint.x_coordinate - planePose2.x_coordinate), 2) + pow((collisionPoint.y_coordinate - planePose2.y_coordinate), 2));
  return overlapDistance;
}
9 Bibliography