# Main Presentation

Hello everyone and welcome to the 1402 external demo for the collision awareness plugin team.

Since there are a few new faces at this demo, we wanted to take a few minutes to provide a bit of background about our group and the current project. Our group is called the Autonomous Flight Systems Laboratory within the department of Aeronautics and Astronautics at the University of Washington. Our mission is to conduct research that advances technologies relevant to unmanned systems. We mainly focus on autonomous algorithm development but we also focus on other issues such as human in the loop architectures a flight testing and integration.

Our department has had a long and successful history of collaboration with Insitu and I thought I’d give a brief history of our recent activities…

The current project is focused on collision awareness for air vehicles. Now with this audience, I don’t need to explain the motivation or technical challenges for collision avoidance, but I usually show this picture to illustrate the fact that for manned aviation, people have a good idea of how perform collision avoidance. (CLICK) Our goal is to instead focus on an ICOMC2 operator and provide situational awareness to help them assess potential conflicts. The important distinction is that we are not focused on sense and avoid but rather, our goal is (CLICK) to create a collision warning system for the ICOMC2 system so operators are aware of potential conflicts and can direct the system to deconflict flight paths.

As stated previously, this project is funded by the Washington Joint Center for Aerospace Technology. As such, there are several of their goals that we would like to fulfil. JCATI’s main directive is to foster the development of technologies which will benefit the Washington State aerospace industry and we are directly addressing this with the collision avoidance plugin. Another goal that we are focused on is to provide students with hands-on industry experience and educational opportunities. We’ll introduce our team in a bit and I think you’ll see we are meeting this requirement as well. We would also like to ensure that we are providing technology for Insitu. The scenario that we are considering is where the operator is controlling of a vehicle via ICOMC2. In addition to this, there may be other participating vehicles where the operator can obtain LOI 4 or higher. There may also be non-participating aircraft in the area such as general aviation aircraft. We would also like to support collision awareness with cursor on target entities and annotations which can represent no-fly zones or restricted areas. Within this framework, we would like to provide back end algorithms and analysis tools to allow a client to analyze this scenario and provide estimates of collisions. These will most likely be delivered in the form of stand-alone assemblies that can be included in larger projects as Insitu sees fit. The next step is to roll these algorithms into a working ICOMC2 2.0 collision awareness plug-in.

I’m actually going to present several user stories together at the same time here. Most of the functionality described here revolves around the entity manager module of the plug-in. This is the object in charge of maintaining a collection of all the entities that the plug-in is aware of and then handing out relevant information to the other modules to perform calculations.

# Entity Manager

To set the stage for these modules, I’d like to show how we envision the plug-in will be used by operators. Like we mentioned previously, there will likely be multiple entities in the environment. (CLICK) the user would like to select certain entities to be managed by the plug-in. (CLICK) Once these are committed to the plug-in, the entity manager will begin tracking them. The user can then add plug-in specific parameters to any of the managed entities. (CLICK) For example, they can modify the collision radius of a certain vehicle. (CLICK) Within this frame work, the user can ask the plugin to determine where potential conflicts with other managed entities exist and the plug-in should display this information to the operator in a useful fashion.

The entity manager can then hand off this information to other software modules which are in charge of performing different types of calculations. The module I’d like to present here is the call the group conflict calculator. Last sprint, we demonstrated the algorithm for computing the probability that two vehicles will collide given stochastic system variables. Darcy asked the question about multi-vehicle collision detection and this is exactly what this module does. (CLICK) The entity manager will hand this module spatial and statistical information about the vehicles and the operator will select a perspective entity. This is the so called “point of view” entity. (CLICK) The group conflict calculator is then in charge of providing information to the operator that will help them become aware of conflicts from the point of view of the perspective entity. (CLICK) The goals of this module are to once again, provide a mathematical, closed form, and conservative guarantee of analysis, allow the user to quickly identify which entities are potentially in conflict, and generate a prioritized list of impending conflicts so they can triage the situation.

To show this in action, we’ve generated a scenario with 5 entities, a ScanEagle, an integrator, a 747, a Cessna 172, and a cursor on target which represents a restricted airspace. (CLICK) the operator is in charge of the ScanEagle and would like to know, (CLICK) will other entities breach my airspace and (CLICK) will my aircraft breach anyone else’s airspace. (CLICK) We could also simulate the scenario where the operator obtains LOI 4 of another vehicle and would therefore like to switch the perspective to another entity.

To illustrate this, our systems currently output data to google earth for visualization so I’ve got that scenario we described shown here. As you can see, here are the 5 entities we mentions and you can see that all calculations are performed in full 3D, not just in the plane. Now, if the operator is controlling ScanEagle01, they would set this to the perspective entity and we can see what the situation looks like from the ScanEagle’s perspective. What we are showing is this red cylinder is the ScanEagle’s airspace and the other ellipsoids represent probable locations of the other entities. These other ellipsoids are color coded depending on the probability that they will breach the ScanEagle’s airspace. As you can see, the Cessna has a high probability of intrusion whereas the Hawaiian and Integrator are effectively 0 chance of breach. For fun, we can see how the scenario changes from the 747’s point of view. As you can see here, the 747 has to worry about traffic from 3 entities, the ScanEagle, the Integrator, and the Cessna in that order.

Let’s go back to the ScanEagle’s perspective. Like we mentioned earlier, if the operator is controlling the ScanEagle, they would like a prioritized list of probabilities that other entities will breach my airspace. The system outputs these types of data. As you can see, the CursorOnTarget has an effectively 100% chance of breaching the ScanEagle’s airspace, but this is actually not perceived as a threat because you aren’t concerned about restricted airspaces hitting you. Instead, the operator needs to be very worried about the Cessna. (CLICK) In addition to worrying about who might hit me, the operator also has to worry about how they might hit. This information is also available and we can see that they actually need to worry about three entities, the Cessna, the 747, and they also have a medium chance of breaching the restricted airspace of CursorOnTarget01.