UAV Software Simulation Writeup

*Ohanes Dadian*

Abstract

This year, the Northrop Grumman Collaborative Project (NGCP) at Cal Poly Pomona put forth a research effort to develop a technology stack for performing software-in-the-loop simulation of our new collision avoidance algorithm. The intent of this system is to leverage the open source FlightGear platform to verify and validate the algorithm before including it into our flight controller.

Overall Architecture

To start the simulation process, the Software-in-the-Loop (SIL) program is started. Upon startup, it waits for communication from FlightGear. FlightGear is then started with the Telemaster flight dynamics model. Upon receiving the current position from FlightGear, SIL creates a message out of our aircraft’s information and passes it over to Collision Avoidance (CA). CA then generates an avoid waypoint and sends it back to SIL for inclusion into FlightGear’s flight plan.

**Figure 1: SIL Architecture**

Software-in-the-Loop

Our Software-in-the-Loop systems was developed using the C programming language. It utlized WinSock, the windows socket layer, for performing communication with FlightGear. Multitasking is achieved using the POSIX Threading library (Pthreads).

This product acts as the common link between aircraft simulation and collision avoidance. It takes care of preparing flight parameters in a format that the two aforementioned components understanding. Providing seamless dialog between both products.

FlightGear

On the UAV simulation side, we are leveraging FlightGear to simulate our aircraft. We are using a Telemaster model, developed by Nigam Patel and Luis Andrade.

**Figure 2: Telemaster model**

FightGear transmits the current position (longitude, latitude, altitude) to SIL over UDP using the property tree. The output is dictated by output\_protocol.xml.



**Figure 3: Output Protocol**

FlightGear adds the avoid waypoints to the flight plan using code written in the Nasal scripting language. A thread is run that continually polls a file buffer for new waypoints. Upon receiving a new waypoint, it is added to the route manager. The route manager manages the flight path for the simulated aircraft’s autopilot.



**Figure 4: Nasal Code**

Collision Avoidance

The first step in the collision avoidance process is detection. Our aircraft obtains information about its position, velocity, and heading from GPS and other onboard sensors. At the same time, our aircraft receives messages from other aircraft, which are broadcasting their position, velocity, and heading. The next step is to prune the received data to eliminate the aircraft which will not influence the flight path of our aircraft. First, the priority of each aircraft determines which aircraft will avoid the others. Lower priority aircraft must avoid higher priorities, therefore any aircraft of lower priority than our aircraft will be neglected by the collision avoidance system. Second, a circle with a radius of 600 meters is placed around our aircraft. Any aircraft that is located outside the circular area will be considered to be far enough away as to be neglected by the collision avoidance system.

**Sense**

The sense algorithm assumes straight paths from waypoint to waypoint. Based on information obtained from Detect (position, heading, velocity) it predicts a future path for both our airplane and the airplanes within our detect area. The path prediction has time steps of half a second for a total of ten seconds into the future. For airplanes to be avoided, the first time they are detected, the sense algorithm creates a circle that must be avoided, with a radius of 20 m around that airplane. This circle represents the area where the airplane could reasonably be located considering GPS error factors. As the time step prediction increases, the circle of probability locations increases proportional to the aircraft’s velocity. If our airplane path prediction is projected to intersect the other airplane’s circle of probabilities, the sense algorithm will call the avoid function; if not, it will continue on its predicted path towards the next pre-set waypoint.

**Avoidance**

Once our aircraft has sensed a possible collision, the avoidance algorithm is run to create a new course for our aircraft. Beginning with a change in heading of 10 degrees to the right, the sense function predicts if the change of course avoids the possible impending collision. If not, a change in heading of 10 degrees to the left is analyzed to determine if the path avoids the collision. The heading continues to change in increments of 10 degrees, to the right and to the left, until a safe path is determined. A new waypoint is set for the new course, and uploaded to the flight computer. If no new collisions are detected, our aircraft returns to its original course for succeeding waypoints.

Testing

To test collision avoidance, the following environment was used.

* FlightGear instance running Telemaster model (port 5000).
* FlightGear instance running Grumman F-14B model (port 5001).
* Software-in-the-Loop and Collision Avoidance sandboxed in threads.

Both aircraft were put into air at 10,000 ft going at 200 kt.

Software-in-the-Loop received current position of both aircrafts.



**Figure 5: SIL getting both aircraft positions**

This is the first case that I am testing, where a collision was not eminent. As the second aircraft is moving further and further away from the first.



**Figure 6: Use Case 1**