UAV Team – Cycle 3 Development Deliverables

**Team Members:** David Jones, Jacob Conaway, Kayla Casteel, Michael Widick

**System Metaphor:** Our ground station will allow a user to implement and evaluate collision avoidance with an easy interface to control a fleet of real and virtual UAVs.

**Cycle Intent:** Refine GUI features and design. Continue testing the GUI. Fully integrate the ROS and GUI sides. Also continue testing the ROS back-end.

1. Introduction - David Jones

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

To reach this autonomy, UAVs need collision avoidance algorithms to prevent collisions. Three years ago, a program started at Auburn University to build an Aerial Terrestrial Test bed for Research in Aerospace, Computing, and maThematics (ATTRACT). The ultimate goal of the ATTRACT project is to autonomously, safely, and efficiently fly up to twelve UAVs within a limited airspace. As of now, roughly fifty students have worked and contributed to design and implement many parts of this project. James Holt, a graduate student advised by Dr. Biaz, designed and implemented a general software architecture for the ground station. Figure 1 illustrates this architecture for the ground station and will help describe the contributions he made to ATTRACT. The architecture is based on the Robot Operating System (ROS). The ROS was originally designed to provide a customizable framework for robot design and control that allows developers to divide tasks into separate processes known as nodes. The ATTRACT project uses a mixture of firmware and hardware to achieve the project’s goals. First, the project uses the Multiplex Easystar airframe. This airframe is lightweight EPO foam that allows for low-energy flight. Within this airframe is the ArduPilot Mega (APM), our autopilot chip. The APM allows for our UAV to fly autonomously. It does this by running the ArduPlane autopilot firmware, developed by DIYDrones.com. One integrated, these components allow the ATTRACT project to have autonomous UAVs, which allows for the use of collision avoidance among multiple UAVs.

2. User Stories – Kayla Casteel (GUI), David Jones (ROS), Michael Widick (Packet Loss)

Name: One UAV on a real-time map

Summary: Show a UAV in flight on a map and update the map in real-time, using the coordinates from the plane to move its associated icon.

Description: Since the UAVs are usually flying fairly high and into the sun, they can be a bit hard to track by eyesight. Line-of-sight could become a problem in the future as well if the UAVs start flying longer missions. To make monitoring the plane’s movements easier, we want to be able to see it on a map that updates in real-time. We’ll use Google Maps and markers to accurately show where the plane is, which way it’s facing, and other information about the plane.

Planned Hours: 40

Planned Hours this Cycle: 0

Actual Hours: 0

Total Hours To Date: 37.5

Coder: Jacob Conaway

Tester: Kayla Casteel

Reviewer: UAV Team

Status: Complete

Name: Multiple UAVs on a map in real-time

Summary: After we get viewing one plane on the map down, we would like to be able to see multiple UAVs on the map updating in real time.

Description: One of our project goals is to have multiple planes – simulated and real – be able to fly in the same airspace. The user interface side of the project should reflect this, so we want to see each plane, simulated and real, on the map.

Planned Hours: 30

Planned Hours this Cycle: 0

Actual Hours: 0

Total Hours To Date: 8

Coder: Jacob Conaway

Tester: Kayla Casteel

Reviewer: UAV Team

Status: Complete

Name: Single plane autonomous flight

Summary: Have the system manage a flight path and alter a real plane’s flight to accomplish a given set of waypoints.

Description: One of the main goals of the UAV program is to achieve autonomous flight with a single plane that is given an ordered set of waypoints. The plane and ground station will need to communicate with each other frequently in order to keep track of where the plane is and if a waypoint has been reached. The plane should be ‘smart’ enough to know when it’s arrived at a waypoint and how to move on to the next one, and does this by retrieving data from its onboard GPS.

Planned Hours: 40

Planned Hours this Cycle: 0

Actual Hours: 0

Total Hours To Date: 37.5

Coder: David Jones

Tester: UAV Team

Reviewer: David Jones

Status: Completed

Name: Multiple plane autonomous flight, simulated planes

Summary: Have the system able to manage real and simulated planes flying in the same airspace

Description: Upon completion of single plane autonomous flight, we would like to expand the system to handle one real plane and multiple simulated planes sharing the same airspace. Collision-detection algorithms will need to be implemented, since the simulated planes are considered real to the ground station, and the real plane will need to avoid colliding with the simulated planes.

Planned Hours: 30

Planned Hours this Cycle: 10

Actual Hours: 9

Total Hours To Date: 29

Coder: David Jones

Tester: David Jones

Reviewer: David Jones

Status: Completed

Name: Multiple plane autonomous flight, real planes

Summary: Have the system manage multiple real planes in the same airspace.

Description: Once the collision-avoidance algorithm is tested with the simulated planes, it will be implemented and tested with several real planes flying autonomously in the same area.

Planned Hours: 44

Planned Hours this Cycle: 26

Actual Hours: 7

Total Hours To Date: 7

Coder: David Jones

Tester: David Jones

Reviewer: David Jones

Status: Discarded

Name: Multiple plane autonomous flight, real and simulated planes

Summary: Have the system manage several planes, real and simulated, flying at the same time in the same airspace.

Description: After we’ve tested real planes flying at once, we’d like to push the limits of the system by flying up to 64 planes at the same time. Since we don’t have 64 real planes, the rest will be simulated planes.

Planned Hours: 44

Planned Hours this Cycle: 0

Actual Hours: -

Total Hours To Date: 0

Coder: David Jones

Tester: David Jones

Reviewer: David Jones

Status: Discarded

Name: Set waypoints using a map

Summary: Allow the user to place waypoint markers by clicking on the map at the point where they want the waypoint to be, and then add the waypoints to the flight path.

Description: Right now, the map reads from a pre-made list of waypoints. To change the waypoints, the user has to go in and edit the file they’re stored in. We’d like to simplify the process by using a map to add waypoints to a roster.

Planned Hours: 5

Planned Hours this Cycle: 5

Actual Hours: 7

Total Hours To Date: 7

Coder: Jacob Conaway

Tester: Kayla Casteel

Reviewer: Kayla Casteel

Status: Complete

Name: Remove waypoints on the map

Summary: Remove waypoints the user has placed from the map and flight path.

Description: A user may want to remove a waypoint they’ve previously set. The user interface should allow them to do that just as easily as they can place one.

Planned Hours: 5

Planned Hours this Cycle: 5

Actual Hours: 5

Total Hours To Date: 5

Coder: Jacob Conaway

Tester: Kayla Casteel

Reviewer: Kayla Casteel

Status: Complete

Name: See waypoint information on map

Summary: Allow the user to see information about waypoints on the flight path.

Description: The user interface should show the user information about a given waypoint, such as its coordinates.

Planned Hours: 5

Planned Hours this Cycle: 0

Actual Hours: -

Total Hours To Date: 0

Coder: -

Tester: -

Reviewer: -

Status: Discarded

Name: Create new missions in a user interface

Summary: Allow the user to create new flight plans easily by using the user interface.

Description: Right now, missions are either text files with a plane ID or just a plain path file. We’d like to allow the user to create their own missions without editing a text file. We can do this by allowing them to go into a “Create mission” mode, where they can click on the map and the waypoints are then recorded in the order they were marked.

Planned Hours: 15

Planned Hours this Cycle: 0

Actual Hours: 7

Total Hours To Date: 7

Coder: Jacob Conaway

Tester: Kayla Casteel

Reviewer: Kayla Casteel

Status: Complete

Name: Remove existing missions from a mission list with a user interface

Summary: Allow a user to remove a previously created mission from the selection of missions.

Description: Just as we want a user to be able to create their own missions easily, we’d like to allow them to remove or delete a previously made mission from the system through a GUI.

Planned Hours: 15

Planned Hours this Cycle: 0

Actual Hours: -

Total Hours To Date: 0

Coder: TBA – GUI Team

Tester: TBA – GUI Team

Reviewer: TBA

Status: Discarded

Name: View missions in a user interface

Summary: Allow the user to view previously created missions

Description: A user should be able to load a mission and see the waypoints on the map, along with information about those waypoints.

Planned Hours: 10

Planned Hours this Cycle: 0

Actual Hours: -

Total Hours To Date: 0

Coder: TBA – GUI Team

Tester: TBA – GUI Team

Reviewer: TBA

Status: Discarded

Name: Assign a mission to a UAV

Summary: A user should be able to add an existing mission to a plane

Description: After a usable mission is made and a UAV registered with the system, a user should be able to select a given plane and a mission to be executed to it.

Planned Hours: 15

Planned Hours this Cycle: 0

Actual Hours: -

Total Hours To Date: 0

Coder: TBA – GUI Team

Tester: TBA – GUI Team

Reviewer: TBA

Status: Discarded

Name: Remove a mission from a UAV

Summary: Remove a mission from the UAV’s queue

Description: Since we can add missions to a UAV, we need to be able to remove them as well so we can retire planes or switch missions. This should be done through fairly the same process as adding a mission.

Planned Hours: 15

Planned Hours this Cycle: 0

Actual Hours: -

Total Hours To Date: 0

Coder: TBA – GUI Team

Tester: TBA – GUI Team

Reviewer: TBA

Status: Discarded

Name: Select a registered UAV in a graphical interface

Summary: The user should be able to single out a specific UAV in order to find out more information about it.

Description: We can either use icons or menus to allow a user to select a UAV by ID, then display information about its speed, bearing, coordinates, et cetera in a formatted manner.

Planned Hours: 10

Planned Hours this Cycle: 0

Actual Hours: 7

Total Hours To Date: 7

Coder: Jacob Conaway

Tester: Kayla Casteel

Reviewer: Kayla Casteel

Status: Complete

Name: Display a selected UAV’s speed

Summary: Once selected, the given UAV’s speed should be displayed in a readable manner.

Description: The plane already reports its speed to the ground station, the GUI just needs to show it consistently with what the plane reports. Since the plane reports in meters per second, we’ll probably stick with that.

Planned Hours: 7

Planned Hours this Cycle: 0

Actual Hours: 5

Total Hours To Date: 5

Coder: Jacob Conaway

Tester: Kayla Casteel

Reviewer: Kayla Casteel

Status: Complete

Name: Display a selected UAV’s bearings

Summary: Show the selected UAV’s bearings in the GUI

Description: The plane also reports its bearings to the ground station in terms of degrees from north. We’ll need to put them in a format a user can easily understand.

Planned Hours: 5

Planned Hours this Cycle: 5

Actual Hours: 1

Total Hours To Date: 1

Coder: Jacob Conaway

Tester: Kayla Casteel

Reviewer: Kayla Casteel

Status: Complete

Name: View a UAV’s target destination during flight

Summary: Tell the user which waypoint the UAV is currently heading towards and show it on the map.

Description: We want to know at any given time, for the selected UAV, which waypoint is it currently headed to. We can do this by either singling it out on the map, or showing the waypoint’s coordinates or ID.

Planned Hours: 7

Planned Hours this Cycle: 0

Actual Hours: 0

Total Hours To Date: 3.5

Coder: Jacob Conaway

Tester: Kayla Casteel

Reviewer: UAV Team

Status: Completed

Name: Show UAV distance to target

Summary: Show the selected UAV’s distance to its current target.

Description: Since we know the target for a given UAV, we’d also like to know how far away the UAV is from reaching the waypoint. This will be displayed in meters, since that’s what the UAV reports to the ground station.

Planned Hours: 5

Planned Hours this Cycle: 5

Actual Hours: 1

Total Hours To Date: 1

Coder: Jacob Conaway

Tester: Kayla Casteel

Reviewer: Kayla Casteel

Status: Complete

Name: Compile QT project as a ROS project

Summary: We want to compile a QT project in the same way a ROS project is compiled.

Description: In order to get the GUI to launch the ROS system, we need to be able to use rosbuild in order to build QT code. This will keep us from having to compile the GUI and the UAV ground station separately.

Planned Hours: 5

Planned Hours this Cycle: 0

Actual Hours: 0

Total Hours To Date: 5

Coder: Jacob Conaway

Tester: Kayla Casteel

Reviewer: Kayla Casteel

Status: Completed

Name: Launch GUI from ROS

Summary: We want the GUI to launch at the same time the ROS part of the system is launched.

Description: In order to run the UAV right now, the ROS half of the system must be launched and a course file given, then the GUI needs to be launched separately after ROS is running. We want the GUI to launch on its own when ROS is launched, to make it easier on the end user.

Planned Hours: 7

Planned Hours this Cycle: 7

Actual Hours: 5

Total Hours to Date: 5

Coder: Jacob Conaway

Tester: Kayla Casteel

Reviewer: Kayla Casteel

Status: Completed

Name: Select a mission with the GUI

Summary: We would like to be able to select a mission to run graphically, instead of with the command line at launch.

Description: Since the UI should launch when ROS does, we want to let the user select the course file with a user interface as well. ROS will then load the user’s selection just as it would if the user had entered it in the command line.

Planned Hours: 10

Planned Hours this Cycle: 10

Actual Hours: 5.5

Total Hours to Date: 8.5

Coder: Jacob Conaway

Tester: Kayla Casteel

Reviewer: Kayla Casteel

Status: Completed

Name: Different icon colors for simulated and real planes

Summary: Allow the user to tell which planes are real or simulated when they are displayed on the map by altering the colors of the icons.

Description: Currently, simulated and real planes have the same colored plane icons on the map. We’d like to differentiate between real and simulated planes by changing the fill color of the icon based on its group.

Planned Hours: 2

Planned Hours this Cycle: 2

Actual Hours: .5

Total Hours to Date: .5

Coder: Jacob Conaway

Tester: Kayla Casteel

Reviewer: Kayla Casteel

Status: Completed

Name: Save locations in the mission planner

Summary: We’d like to save a center point for later use in planning missions.

Description: To plan a mission currently, the user has to click and drag the map and center it over the chosen area. This is fine for nearby locations, but for planning missions in other cities it becomes tedious. It would be more convenient to save a location’s coordinates so they can be quickly used in later missions.

Planned Hours: 7

Planned Hours this Cycle: 7

Actual Hours: 7

Total Hours to Date: 7

Coder: Jacob Conaway

Tester: Kayla Casteel

Reviewer: Kayla Casteel

Status: Complete

Name: Load locations in the mission planner

Summary: Load a previously saved location into the mission planner.

Description: In order to make creating missions easier, we’d like to load previously saved location coordinates into the mission planner and have the map center on them.

Planned Hours: 7

Planned Hours this Cycle: 7

Actual Hours: 2

Total Hours to Date: 2

Coder: Jacob Conaway

Tester: Kayla Casteel

Reviewer: Kayla Casteel

Status: Completed

Name: Address bugs found during GUI testing

Summary: Fix bugs found in the GUI during testing.

Description: In order to ensure the user interface acts as expected, it needs to be tested and any bugs found need to be resolved.

Planned Hours: 36

Planned Hours this Cycle: 36

Actual Hours: 40

Total Hours to Date: 40

Coder: Jacob Conaway

Tester: Kayla Casteel

Reviewer: Jacob Conaway, Kayla Casteel

Status: Completed (in terms of Senior Design, but ongoing from project perspective)

Name: Record packet loss for one UAV

Summary: We need to record packet loss between one UAV and the ground station.

Description: We’ll record packet loss rates across all the frequencies the planes use in order to determine at what frequency are the most packets arriving at the destination. This will make the connection between the UAV and ground station more reliable and our real-time updating more accurate.

Planned Hours: 20

Planned Hours this Cycle: 10

Actual Hours: 10

Total Hours To Date: 20

Coder: David Jones, Michael Widick

Tester: Michael Widick

Reviewer: Michael Widick

Status: Completed

Name: Record packet loss for five UAVs

Summary: Record packet loss for at least five UAVs. Four frequencies should be tested: 1Hz, 10Hz, 25Hz, and 50Hz.

Description: After testing the frequencies for one UAV, we want to test for packet loss between at least five UAVs and the ground station. The loss rates will likely be higher, which is important to know before we start flying multiple real UAVs at the same time.

Planned Hours: 15

Planned Hours this Cycle: 15

Actual Hours: 13.5

Total Hours To Date: 13.5

Coder: David Jones, Michael Widick

Tester: Michael Widick

Reviewer: Michael Widick

Status: Completed

Name: Record Packet Loss for five UAVs, with more specific frequencies

Summary: Record packet loss for at least five UAVs.

Description: After testing the previous five UAVs, we want to test some more specific frequencies. This frequencies will be determined in User Story 21.

Planned Hours: 20

Planned Hours this Cycle: 20

Actual Hours: 10

Total Hours To Date: 10

Coder: David Jones, Michael Widick

Tester: Michael Widick

Reviewer: Michael Widick

Status: Completed

Name: Generate a random mission

Summary: Generate a random mission with the GUI's mission planner

Description: We would like to have the GUI generate random waypoints for us, given a specific number of planes, number of waypoints per plane, and a boundary to work in.

Planned Hours: 30

Planned Hours this Cycle: 30

Actual Hours: 7

Total Hours To Date: 7

Coder: Jacob Conaway

Tester: Kayla Casteel

Reviewer: Kayla Casteel

Status: Complete

Name: Edit saved locations on the map

Summary: Edit a previously saved location and have the saved changes persist.

Description: Right now, once a location is selected and named, there is no way to edit that location without deleting it and making a new one with the same name. We would like to give the user the ability to edit a previously saved location to make this process easier. Any changes made to the location and saved should persist after the system is restarted.

Planned Hours: 5

Planned Hours this Cycle: 5

Actual Hours: 5

Total Hours To Date: 5

Coder: Jacob Conaway

Tester: Kayla Casteel

Reviewer: Kayla Casteel

Status: Complete

Name: Add required logo to weather data

Summary: Add the legally required WeatherUnderground image to weather data

Description: We display weather data from WeatherUnderground in the GUI for the user to see. The site requires in their terms of service that their logo is displayed along with their data.

Planned Hours: 3

Planned Hours this Cycle: 3

Actual Hours: 2

Total Hours To Date: 2

Coder: Jacob Conaway

Tester: Kayla Casteel

Reviewer: Kayla Casteel

Status: Complete

3. Design Documentation – David Jones (ROS), Jacob Conaway (GUI)

3.1 ROS Design Documentation – David Jones

The ATTRACT ground station has six nodes depicted as ovals on Figure 1: 1) X-Bee IO, 2) Plane Telemetry Data Simulator, 3) Coordinator, 4) Collision Avoidance Center, 5) Visualization Center, and 6) Control Menu. The X-Bee IO node is the interface between the real UAVs and the ground station: it implements the communication system to receive telemetry updates from the UAVs and to send them commands from the Coordinator node. The plane telemetry data simulator simulates virtual UAVs that mimic the real UAVs to fulfill assigned fictive missions. The X-Bee and plane telemetry data simulator nodes publish their associated telemetry updates on the Position Updates Topic. Specifically, the global positions, speeds, bearings, and other flight data are published on that Position Updates Topic. Other nodes such as the coordinator, the collision avoidance center, and the visualization center read these telemetry updates to accomplish their respective functions. The visualization center displays the real and virtual UAVs in flight using Google Maps. The collision avoidance center continues to monitor the telemetry updates to determine whether a conflict is imminent and, if needed, sends new evading mandatory waypoints to prevent a collision. These waypoints are sent to the coordinator node that will assign the new waypoint to the respective UAVs. The mission of the coordinator is to assign each UAV its next waypoint: it sends them on the *commands* topic. This topic is read by the X-Bee IO and plane telemetry data simulator nodes that will alert the conflicting UAVs of the new urgent evading waypoints. Finally, the control menu node serves as the user interface.

In ROS, each node is a stand-alone C++ program. This gives the user great versatility for deploying very specific missions. For instance, if we decided we only wanted to use real UAVs, we could omit deploying the plane telemetry data simulator. Therefore, each node performs a specific purpose, which is entirely encapsulated within that node. Within the ATTRACT ROS ground-station, the coordinator node is the “brains” of the operation. The coordinator receives frequent telemetry messages from each UAV. Every time a message is received, the coordinator decides if that particular UAV has reached its target waypoint. If so, the coordinator will delete that waypoint from the waypoint queue and send the next waypoint to the UAV. This command to the UAV is passed through the *commands* topic. The XbeeIO node receives commands from the *commands* topic. After converting this ROS command to a MAVLink message, the command is sent through the serial port to the UAV via Xbee. Likewise, the XbeeIO node receives frequent telemetry messages from each UAV. Once converting this message from a MAVLink message to a ROS message, the message is posted on the *Position Updates* topic. Essentially, the Plane Telemetry Data Simulator acts just like the XbeeIO node, except it doesn’t interface with real UAVs; the Plane Telemetry Data Simulator simulates the UAVs. The Collision Avoidance Center node also receives messages from the *Position Updates* topic. Every time a message is received, the Collision Avoidance Center runs a collision avoidance algorithm to decide if a collision is imminent. If a collision is close, the Collision Avoidance Center will send an avoidance waypoint, using the *commands* topic, which the UAV will immediately fly toward. Furthermore, the Visualization Center node is an intermediary between QT, the environment used for our Graphical User Interface (GUI). It will translate the *Position Updates* topic information into QT signal. Likewise, when sending a command from the GUI, the Visualization Center will translate this information into a ROS message that can be posted onto the *commands* topic.

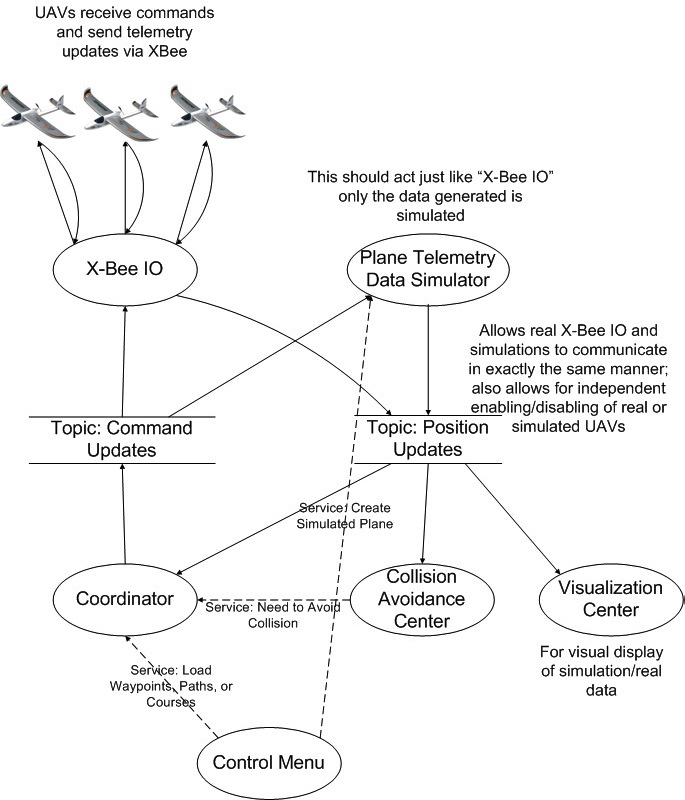


Figure 1: Ground Station Software Architecture.

A minor design decision involved the system IDs of the real and simulated planes. A decision had to be made concerning the naming conventions for the real planes. The existing system relied on matching system IDs to array indices. Therefore, the system IDs for the simulated planes, which had been implemented in the existing system, had the range of zero to 31. Rather than spending time altering the system, we chose to simply extend this convention and create a range of 32 to 64 for the system IDs that correspond to the physical planes.

3.2 GUI Design Documentation

**Cycle 1**

At first glance, it may not seem as though there was much focus on engineering design within the GUI. This is largely due to poor naming. Several files need to have more descriptive names, but the group did not want to risk destroying the product for the sake of simply renaming files. Now that a somewhat stable version has been achieved, time can be taken in future cycles to revise class and project names. An attempt was made to be as object oriented as possible, but a large amount of the code that interacts with the map is written in JavaScript, which does not use classes. Another hurdle in providing a clean design is the limitations of the GUI project. The GUI itself is written using QT, which is based off of C++. The map that is shown in the GUI is actually a website. All interaction with this website occurs through JavaScript. The base map and creation of variables resides in the website’s HTML file (in a JavaScript section) that is loaded when the program launches. This means that anytime the map needs to be altered, a QString (QTs string implementation) has to be created that contains the JavaScript code. This string with the code is then passed to the website, where it is executed by Google’s servers. This makes it very difficult to apply object oriented design principles. That being said, the GUI application as a whole was created with engineering design in mind.

We reviewed several different packages when deciding on how to best integrate a GUI with ROS. These packages allow for C++ code to be compiled with connections to both ROS and QT. Eros\_qt was one choice, but it was made for embedded ROS and would have required several changes to work correctly. PySide was another, but it would add yet another programming language to the mix and may further complicate cross-framework transitions later in the project. RQT was a strong contender, but it was very tightly integrated with ROS and only allowed for small QT plugins, which would have made adding the webview where the map resides more difficult. We finally settled on qt\_ros. It does a great job at keeping the two different frameworks separate. With a few modifications, the QT application that had been created for the architectural spike could communicate with the existing ROS application. The abstraction is so good that the ROS and QT codebase can be in two separate directories. We felt that this would allow for scalability and prevent the GUI from being limited by any restrictions imposed by ROS. It also would make it very easy for each system to be updated or significantly changed without affecting the other. The qt\_ros package also included a feature that builds a bare bones skeleton structure so that it is easy to figure out what is going on.

One major section of the application is the QNode class. It acts as the adapter between ROS and QT. ROS and QT handle callbacks differently. ROS uses publishers and subscribers. QT uses Signals and Slots. QNode is the intermediary between these differing approaches. For the UAV project in particular, this class is used to subscribe to the Telemetry Update topic on the ROS side and then subsequently publish this information as a Signal, which can be received by a Slot on the QT side. The communication is currently one way, but as the project progresses it should be fairly easy to have bidirectional communication between the two frameworks. The flow would simply need to be reversed. A QT Signal would be received by a Slot in the QNode and then resubmitted as publisher on the ROS side. This class is the heart of the adaptor philosophy and allows us to keep each section of code separate to prevent any cross-framework issues.

In the QT application there is a very simple main class that starts the program and launches the main window. The aptly named MainWindow file creates a main window that serves as a simple container for user interface “widgets”. This main window class also connects the signals from the QNode class to the slots in Form class. The Form class is the heart of the visible GUI. It is the only widget in the program and contains all of the check boxes, a list pane (to be used later), and a webview where the Google Map resides. As mentioned previously, the MainWindow serves as a mere container for widgets and the only widget used in the program is the Form widget. Therefore, from the perspective of the user, the main window and the form are the same thing. Both the MainWindow and the Form are comprised of two main files: a standard .cpp file and a QT specific .ui file. The .ui file for the Form class is set up with a grid layout. Each of the items in the window aligns itself to the grid. This makes spacing very easy and uniform. The .cpp Form file is where the majority of the code lies. We had a tough time deciding if a separate class needed to be added to house this code. In the end, we felt that because of the JavaScript limitations discussed in the first paragraph and the fact that all of the code did in fact deal with the user interface it would be best to leave it in the Form class. As far as functionality is concerned, the Form class is responsible for receiving the adapted telemetry update messages from the QNode class and then updating the marker position, flight path, and waypoint circle appropriately. In addition, it sets the center of the map to best view the planes and, if enabled, zooms the map to an appropriate level. It also is responsible for all of the user options that can be set which consists of enabling and disabling different features. The majority of code in this class is a hybrid of client side C++, and JavaScript that is sent to Google’s servers to be executed. The class is designed so that the methods are on the C++ side and statements are simply executed on the JavaScript side. We may consider adding functions to the JavaScript section of the HTML file, and then calling these functions in the Form class, but this will be determined in later cycles.

The final piece of the GUI application is the HTML file that is loaded by the webview. This HTML file displays a loading image, preloads the plane icons used by the map, and sets up the Google Map. The preloading of icons is accomplished by displaying the images on a hidden section of the page. The Google Map is set up using a section of JavaScript that sends a request to Google for the map and then creates the variables that will be needed later. Currently, the JavaScript code uses global variables. We usually try to avoid this, but given the unique characteristics of JavaScript (no concept of classes) and the reassurance that we will only be running a single script in the webview, we felt that it would be acceptable.

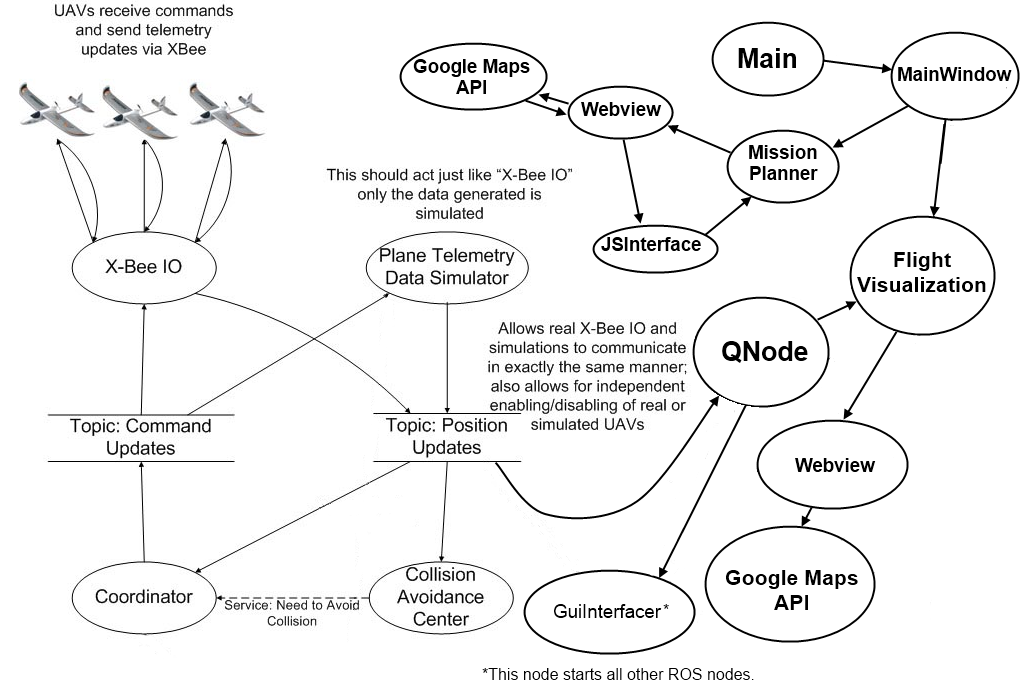


Figure 2: Graphical User Interface Architecture

**Cycle 2**

After reviewing our source code, we have noticed several opportunities to improve the readability, performance, or overall design. We plan on spending time in the following cycles to refactor and revise our code to improve it. This includes renaming all ambiguous or misleading titles, adding comments where necessary, and general refactoring. Several of our user stories have been marked as completed. When we begin to refactor, we will simply add the hours appropriately regardless of the user story's status.

Cycle 2 focused mainly on new features. With that being said, several functions were renamed for clarity and the Form class was renamed FlightVisualization to better reflect its role in the system. The Javascript section of the HTML file has been reworked into functions, so now the FlightVisualization class simply calls the corresponding Javascript function instead of sending code line by line to be dynamically executed. A new tab was added to the MainWindow called MissionPlanner. This new tab and corresponding class allows a user to create a random and manual course file that can then be loaded into the ROS system. It also allows the user to edit existing course files. The class is similar to the FlightVisualization class, in that it contains a webview with a Google Map. In this instance, however, there is two-way communication. Currently this is handled by passing the MissionPlanner instance to the Javascript code. This exposes all of the public methods of MissionPlanner to the Javascript code. The GUI team will evaluate this approach and may introduce an intermediary class known as JSInterface to handle this two-way communication. This decision will be made in cycle 3. The MissionPlanner tab allows a user to select various waypoints on the map for each plane ID and then write the generated file to whatever directory the user chooses. These files can then be loaded into the ROS system. The user has the option to reset the map and to set the center of the map. There is also a button to get the current wind conditions for the area.

In addition to the new mission planner tab, several changes were made to the existing GUI. The user can now select planes individually from a drop down list to see relevant information. This information has been correctly formatted and is now displayed in a sidebar on the right of the screen. Flight paths can be reset to help reduce the visual clutter on the map.

Several performance changes have been made after the most recent test flight. This test flight occurred in Tuskegee and required the use of a cellular tethered Internet connection. It exposed some of the weaknesses in the GUI system. Previously, the plane icons were image files hosted on a webserver that had to be loaded before the application started. The ~3MB worth of data caused a very small load time on the Auburn network, but this load was unacceptable on the slower cellular network and almost crippled the application. To address this, the GUI team decided to instead use vector graphics for the plane icons. Vector graphics are essentially sets of coordinates that can be rendered into shapes much like a connect-the-dots drawing. This rendering can be done dynamically and eliminates the need for any outside hosting of images. This decision has the added benefit of reducing the loading time to a nearly instantaneous level. Another issue discovered during the flight, was how crowded the map can become with 33 planes. This was resolved by adding a popup window that allows the user to select which planes they wish to show or hide on the map.

**Cycle 3**

The final cycle was largely devoted to a general cleanup of code and an attempt to fully comment all of the source code. This has largely been completed, although there are still a few sections that need to be commented. The team will continue to comment these sections and will provide an updated CD for the binder when it is complete. There were also a few features that were added. One of the most significant is the further integration of the GUI subsystem and the ROS subsystem, which is discussed in the following section. Another noteworthy design decision was to add an intermediate class between the JavaScript and MissionPlanner code. Previously, each public method and slot of the MissionPlanner class was visible to the JavaScript code in the embedded browser. The GUI team felt that this posed a security risk and decided to add an intermediate class called JSInterface. This new class allows the team to expose the methods that they want to be visible to the browser and the class, in turn, calls the corresponding functions in the MissionPlanner class. Another minor design decision dealt with the plane icons on the Flight Visualization tab. At a previous test flight it became apparent that it was difficult to quickly, at a glance, distinguish between real and simulated planes. Therefore, the decision was made to change the fill color of each. Simulated planes now have a white fill color, while real planes have a black fill. The planner map markers were changed from using a deprecated Google Charts API service to a group of third party icons. The API service worked great, but the team was worried that Google would terminate the service without notice. In addition to the new features, time was dedicated to renaming various pieces of code. Unfortunately, some of the names (such as Google\_Maps\_CMAKE, a holdover from cycle 1) are deeply trenched and could not be easily renamed. The decision was made to hold off on changing those names, given the time remaining and the risk of introducing bugs during the process.

3.3 ROS and GUI Integration Design Documentation

During the third cycle, the team further integrated the ROS and GUI system. We decided that the GUI should initiate the ROS subsystem rather than both starting simultaneously. This new approach allows the GUI and map images to fully load before the course is loaded. This may not seem that important, but in the field with tethered Internet access it can sometimes take a long time to load the map. In previous test flights, there has been a blind spot period where the plane is under the system’s control, but is not yet visible on the map. It also allows for the course file to be selected graphically rather than through the terminal, which is a great improvement in the user interface. To achieve this goal, we added a node called GUIInterface between ROS and the GUI. This new node waits on a file name to be passed from QNode and then starts the other ROS nodes when the filename is received. The team was faced with a decision on whether or not to simply add this functionality to QNode or to create a new node. We decided to create a new node, because it reinforced the idea of the GUI and ROS subsystems communicating with a clear, simple protocol that would allow each to be easily updated. QNode does not need to know about the other ROS nodes. It simply needs to pass the filename to the ROS system and then wait for the telemetry updates. This way, the ROS system can still be easily changed without a large effect on the GUI and vice-versa. The altered ROS system would simply have to accept a file name and publish telemetry data. This may seem convoluted and more complicated than necessary, but the team felt it was for the best. In essence, QNode is the interface between ROS and QT. It adapts ROS information into something that QT can understand. Similarly, GUIInterface is the adapter between the ROS standard protocols and the specific AU\_UAV\_ROS implementation. It takes care of the project-specific details of handing off a course file to the rest of the nodes. These levels of abstraction were very important to the team. They cement the goal of interoperability between the device framework and the graphical interface, allowing each to be updated independently while providing a seamless integration for the end user. This separation also allows for each system to run independently. For instance, a user can open the GUI to create a course file without having to run the ROS back-end (although they will receive a friendly warning that the ROS framework is not running) and the ROS system can still be executed on its own by using a different launch file. Figure 2 has been updated to reflect this new architecture.

4. Setup and Toolchain – David Jones (ROS), Jacob Conaway (GUI)

4a. ROS setup and toolchain:

1. First, install the Ubuntu environment on the current OS.
   1. Please note that only Ubuntu versions 11.10 – 12.0.4 have been tested with the UAV ROS package
   2. For Windows or Mac, install the NOOTRIX virtual box from <http://nootrix.com/downloads/>
      1. Step by step instructions are provided here: http://nootrix.com/2012/09/virtualizing-ros/
   3. For Linux machines, follow the instructions on www.ubuntu.com/download/desktop
2. Second, set up the ROS environment
   1. For Windows or Mac, once the NOOTRIX virtual box is installed, the ROS environment comes preconfigured.
   2. For Linux machines running Ubuntu 12.04, follow the step by step instructions on http://nootrix.com/2012/05/ros-installation/
3. Lastly, obtain the AU\_UAV\_ROS package from the provided CD and place it in the ros workspace.

4b. ArduPlane set up and toolchain:

1. First, download and install Arduino onto the computer
   1. For Macs, download this client: http://code.google.com/p/ardupilot-mega/downloads/detail?name=ArduPilot-Arduino-1.0.3-Mac.zip&can=2&q=
   2. For Windows, download this client: http://code.google.com/p/ardupilot-mega/downloads/detail?name=ArduPilot-Arduino-1.0.3-windows.zip&can=2&q=
   3. For Linux, there is not an Arduino IDE. The source code must be manually compiled and uploaded.
2. Second, download the ArduPlane firmware from the provided CD.
3. After connecting an APM board via USB, the firmware can be edited and uploaded to an UAV.

4c. GUI setup and toolchain:

1. To get started with the GUI subset of the project, you will need to start with a Ubuntu Linux machine that has the ROS system installed (See Above) and the AU UAV package loaded.
2. Next, navigate to <http://qt-project.org/downloads> and click on “Qt libraries 4.8.4 for Linux/X11 (225 MB)”. This will start a fairly large download that might take some time. If prompted on where to save it, just pick a folder with a path that you can easily remember. (Note: Please ensure that the correct version is downloaded. Portions of the file saving code are version-specific.)
3. Once the file has been downloaded, it must be extracted. This can be done by double clicking on the file and using the archive manager, or by using the terminal. If using the terminal, first change the directory to the path where you saved the file and then run “gunzip qt-everywhere-opensource-src-4.8.4.tar.gz” followed by “tar xvf qt-everywhere-opensource-src-4.8.4.tar”.
4. Now open terminal (if it is not already open) and change the directory to the extracted folder labeled “qt-everywhere-opensource-src-4.8.4” and then run “sudo ./configure”. When prompted type ‘o’ for open source edition and “yes” to accept the license. This may take a little while to run (~30 mins).
5. After that is finished, run “sudo make”. This step will take several hours (up to 10 if using a virtual machine).
6. Once make is finished, run “sudo make install”. This will also take several hours, but less than required for “make”
7. Next, we need to install QT Creator. This allows for the user interface files to be edited and provides a very good IDE for the QT system. Once again we need to visit <http://qt-project.org/downloads>. From there, we will click on “QT Creator 2.7.0 for Linux/X11 32-bit (60 MB)”.
8. Once the file has downloaded, open terminal and navigate to the directory where it is saved. After this, run “sudo chmod +x qt-creator-linux-x86-opensource-2.7.0.bin”
9. Next, run “./qt-creator-linux-x86-opensource-2.7.0.bin”. This will open an installer. Follow the on screen steps to complete the installation (Next, Next, Agree, Install, Finish). On the last screen, be sure to uncheck the box that labeled start QT Creator. We will need to run it from the terminal instead. If you forget, simply close the instance of QT Creator that appears.
10. We now need to install qt-ros, an intermediate between QT and ROS. This can be done through the terminal with “sudo apt-get install ros-fuerte-qt-ros”. Type “y” at the appropriate prompt. Note: you may have to run “sudo apt-get update” first if you receive an error.
11. Obtain the “Google\_Maps\_CMAKE” folder from the CD.
12. Move it to your ROS workspace folder.
13. Open the “Google\_Maps\_CMAKE” folder and delete the “build” folder.
14. Navigate to your ROS workspace and then to the “Google\_Maps\_CMAKE” folder.
15. Next type “mkdir build” and navigate to the new build folder.
16. Now type “cmake ..”.
17. Once that has completed, type “rosmake Google\_Maps\_CMAKE”.
18. Now you are ready to edit or run the files. The easiest way to do this is with QT Creator. To launch QT Creator, open a terminal and type “cd” to go back to the base directory. Then type “cd qtcreator-2.6.2/bin/” followed by “./qtcreator”. This will launch the QT Creator IDE. It is necessary to open it through the terminal so that all of the necessary headers are included. The project is now ready to executed via the terminal, if desired. Also, roslaunch should work from this step forward.
19. Once QT Creator opens, go to File->Open File or Project. Navigate to the Google\_Maps\_CMAKE folder in your ROS workspace and select “CMakeLists.txt”.
20. If this your first time loading the project you will see the CMake Wizard. Simply press “Run CMake” and then “Finish”. Note: This does not mean that you can skip the cmake step. If you do, you will end up changing the build directory.
21. Once the project is loaded you can edit, run, or debug the files. For the GUI to launch, ROS must be already running. If you are not ready to launch the AU\_UAV\_ROS node you can simply type “roscore” at the command line and the ROS subsystem will start up.

5. Test Documentation – David Jones (ROS), Kayla Casteel (GUI), Michael Widick (Packet Loss)

5a. Acceptance Tests

* User Story 1 – Map Testing - Launch the map with the UAV simulator, simulating one plane. Reference the icon’s coordinates with the ones the simulator sends. Watch the map for real-time updates. – AU\_UAV\_ROS, GUI program
* User Story 2 – Map Testing - Launch the map with the UAV simulator, simulating multiple planes. Reference the icons’ coordinates with the ones the simulator sends. Watch the map for real-time updates. – AU\_UAV\_ROS, GUI program
* User Story 3 – Flight Tests - During a flight with one real UAV, the ROS system should command the UAV to fly an entire mission while receiving telemetry updates periodically. – AU\_UAV\_ROS, ArduPlane
* User Story 4 – Flight Tests – During a flight with one real UAV and multiple simulated UAVs, the ROS system should alter the real UAV’s path to avoid a collision. – AU\_UAV\_ROS, ArduPlane
* User Story 5 – Flight Tests – During a flight with multiple real UAVs, the ROS system should command the UAVs to fly their entire mission while receiving telemetry updates periodically. – AU\_UAV\_ROS, ArduPlane
* User Story 6 – Flight Tests – During a flight with multiple real UAVs and multiple simulated UAVs, the ROS system should alter the real UAVs’ path to avoid a collision. – AU\_UAV\_ROS, ArduPlane
* User Story 7 – GUI Tests – Load the map and click on it. Record if waypoint is added, and if added, in the correct place with correct coordinates. – AU\_UAV\_ROS, GUI program
* User Story 8 – GUI Tests – Load the map and a waypoint. Choose to remove the waypoint from the map. Observe if the waypoint was removed. – AU\_UAV\_ROS, GUI program
* User Story 9 – GUI Tests – Load the map and a waypoint. Select the waypoint. Observe if the correct information about the waypoint is displayed. – AU\_UAV\_ROS, GUI program
* User Story 10 – GUI Tests – Load the GUI. Choose to create a new mission, add waypoints to it, then save. Check if the mission was saved, and if so, it if contains the correct list of coordinates that match the added waypoints. – AU\_UAV\_ROS, GUI program
* User Story 11 – GUI Tests – Load the GUI and the mission list. Choose to remove a mission from the list. Check if the mission was removed from the list and all the UAVs unassigned from it. – AU\_UAV\_ROS, GUI program
* User Story 12 – GUI Tests – Load the GUI and a mission. Check if the waypoints associated with the mission appear on the map in their correct places. – AU\_UAV\_ROS, GUI program
* User Story 13 – GUI Tests – Load the GUI, at least one mission, and at least one UAV. Assign the mission to the UAV using the GUI. Check if the UAV queued the waypoints associated with the mission. – AU\_UAV\_ROS, GUI program
* User Story 14 – GUI Tests – Load the GUI and at least one plane with a mission already assigned to it. Remove the mission from the UAV using the GUI. Check if the UAV has an empty waypoint queue. – AU\_UAV\_ROS, GUI program
* User Story 15 – GUI Tests – Load the GUI and at least one plane. Select the plane using the GUI. Observe if any information about it is displayed and its path is shown on the map. – AU\_UAV\_ROS, GUI program
* User Story 16 – GUI Tests – Load the GUI and at least one plane. Select the plane. Launch the plane. Check if its speed is shown and if so, if the speed is correct and in meters per second. – AU\_UAV\_ROS, GUI program
* User Story 17 – GUI Tests - Load the GUI and at least one plane. Select the plane. Launch the plane. Check if its bearing is shown and if so, if the bearing is correct and presented in a readable manner. See if the plane’s bearing is reflected in the plane icon on the map. – AU\_UAV\_ROS, GUI program
* User Story 18 – GUI Tests - Load the GUI and at least one plane. Select the plane. Launch the plane. Check if its target waypoint is shown on the map in the correct place. – AU\_UAV\_ROS, GUI program
* User Story 19 – GUI Tests - Load the GUI and at least one plane. Select the plane. Launch the plane. Check if its distance to the next waypoint is shown, and if so, if the distance is in meters and correct. – AU\_UAV\_ROS, GUI program
* User Story 20 – GUI test – Create a QT project. Try to compile the project with rosbuild. See if the project will still run. – GUI program, rosbuild
* User Story 21 – Packet Loss Testing – The Packet Loss Rate should be 0% for one UAV. This information should be recorded in an excel spreadsheet. – mavlink\_ros, ArduPlane
* User Story 22 – Packet Loss Testing – The Packet Loss Rate should be recorded for multiple UAVs. This information should be stored in an excel spreadsheet to produce a significant graph. – mavlink\_ros, ArduPlane

**GUI System**

These tests are all written from the point of view of the latest version. The tests are written using a simulated plane, but should also apply to real UAVs. For each of these tests, the ROS program and GUI program must be running.

User Story 1 - Basic Test 1

* Start the ROS program with a single simulated plane.
* Start the GUI application.
* The GUI should set the center of the map to the single plane and should readjust the center every five seconds.
* A flight path should be dynamically created as the plane moves.
* The current waypoint should be highlighted on the map and be updated to the next waypoint when the plane hits the current waypoint.

User Story 2 - Basic Test 2

* Start the ROS program with a multiple simulated planes.
* Start the GUI application.
* The GUI should set the center of the map to the center of all of the plane locations and should readjust the center every five seconds.
* A flight path should be dynamically created for each plane as the plane moves. Each path should have a unique color that is associated with each plane.
* The current waypoint for each plane should be highlighted on the map and be updated to the next waypoint when the plane hits the current waypoint. Each waypoint should have a unique color that is associated with each plane.

User Stories 1, 2, 18 - Functionality Test 1

* Start with Basic Test 1 or Basic Test 2.
* Uncheck the “Show Flight Path” box.
* All paths currently on the map should disappear.
* Check the “Show Flight Path” box.
* All paths that were on the map should reappear.

User Stories 1, 2, 18 - Functionality Test 2

* Start with Basic Test 1 or Basic Test 2.
* Uncheck the “Show Waypoints” box.
* All waypoints currently on the map should disappear.
* Check the “Show Waypoints” box.
* All waypoints that were on the map should reappear.

User Stories 1, 2, 18 - Functionality Test 3

* Start with Basic Test 1 or Basic Test 2.
* Uncheck the “Show Waypoints” box.
* All waypoints currently on the map should disappear.
* Check the “Show Waypoints” box.
* All waypoints that were on the map should reappear.

User Stories 1, 2, 18 - Functionality Test 4

* Start with Basic Test 1 or Basic Test 2.
* Uncheck the “Auto Center Map” box.
* The “Auto Zoom Map” checkbox will be disabled.
* The map will no longer auto center (or auto zoom if it was checked).
* Check the “Auto Center Map” box.
* The map will once again automatically center itself every five seconds.
* The “Auto Zoom Map” checkbox will be re-enabled.

User Stories 1, 2, 18 - Functionality Test 5

* Start with Basic Test 1 or Basic Test 2
* Check the “Auto Zoom Map” box
* The map will start to automatically zoom to the appropriate level each time it auto centers itself.
* Uncheck the “Auto Center Map” box
* The map will no longer automatically zoom.

5b. Unit Tests

**ROS System**

XBeeIO Tests:

|  |  |  |
| --- | --- | --- |
| # | Input | Expected Output |
| 1 | open\_port(“/dev/ttyUSB0”) | Output > -1 |
| 2 | open\_port(“blah”) | -1 |
| 3 | setup\_port(0, 57600, 100, 200, true, true) | True |
| 4 | setup\_port(-1, 8, 0, 100, true, true) | False |
| 5 | close\_port(0) | True |

standardDefs Tests:

|  |  |  |
| --- | --- | --- |
| # | Input | Expected Output |
| 1 | isBlankLine(“ “) | true |
| 2 | isBlankLine(“blah”) | false |
| 3 | isBlankLine(“\t”) | true |
| 4 | isBlankLine(“\n”) | true |
| 5 | isValidYesNo(‘Y’) | true |
| 6 | isValidYesNo(‘N’) | true |
| 7 | isValidYesNo(‘y’) | true |
| 8 | isValidYesNo(‘n’) | true |
| 9 | isValidYesNo(‘Q’) | false |
| 10 | distanceBetween(wp, wp) | 0 |
| 11 | Wp1 = (35.603464, -85.490434) Wp2 = (35.601340, -85.490477)  distanceBetween(Wp1, Wp2) | 236.2 |

GUI Tests

Launching the System

1: Starting the GUI

Type “roslaunch AU\_UAV\_ROS guiDriven.launch” into the terminal, without

quotation marks.

The GUI window with the map should open. The Flight Visualization tab

should be active.

2: Loading a course

Start the GUI as in test 1.

In the top left, go to File > Load Mission.

A window should open the with courses folder open.

Select a course file and Open it.

The mission should launch and the map should center on the loaded planes.

3: Running a non-course file

Create a text file called NonCourseFile.txt.

Place NonCourseFile.txt in the

ros\_workspace/AU\_UAV\_stack/AU\_UAV\_ROS/courses folder.

Start the GUI as done in test 1 and go to the file selection menu like in test 2.

In the file selection screen, in the bottom right, click on the dropdown box

that has “Course File” selected.

It shouldn't let you select any other file type, and NonCourseFile.txt shouldn't

appear in the file list.

4: Running a broken course file

Create a text file called BrokenCourse.course. Type a random string of letters

in so that the file is gibberish.

Place BrokenCourse.course in the

ros\_workspace/AU\_UAV\_stack/AU\_UAV\_ROS/courses folder.

Start the GUI as done in test 1 and go to the file selection menu like in test 2.

In the file selection screen, select BrokenCourse.course and Open it.

A message box should pop up, explaining that the course file was invalid.

Click “OK”

The GUI will then close itself.

You can now stop ROS in the terminal with Ctrl+C (Mac+C for macs).

Flight Visualization Tab

1: Selecting a plane with no missions loaded

With the GUI loaded, click on the “Select a Plane” dropdrown box under Plane

Information.

There should be no planes to choose from.

2: Reset hidden flight paths

Ensure “Show Flight Paths” check box is checked.

Uncheck “Show Flight Paths”

The “Reset Flight Path(s)” button should be greyed out and you should be

unable to click on it.

3: Manually centering map disables auto zoom

Ensure “Auto Center Map” is checked.

Uncheck “Auto Center Map”

“Auto Zoom Map” should become unchecked (or remain unchecked) and grey

itself out. You should not be able to click on it.

Check “Auto Center Map” again.

“Auto Zoom Map” should be unchecked but unlocked for interaction.

For these, load a course with File > Load Mission.

4: Removing flight paths

Ensure that the “Show Flight Paths” box is checked.

Click the checkbox labeled “Show Flight Paths”

All the trails following the planes on the map should disappear.

5: Showing flight paths

Ensure that the “Show Flight Paths” box is unchecked.

Click the checkbox labeled “Show Flight Paths”

The planes should show the trails matching their icon's outline color.

6: Resetting flight paths

Click the button labeled “Reset Flight Path(s)” in the top right.

The trails following the planes should be cleared, but the planes should begin leaving new trails matching their icon's outline color.

7: Hide waypoints

Ensure that the “Show Waypoints” box is checked.

Click the checkbox labeled “Show Waypoints”

All the colored circles indicating waypoints should disappear.

8: Show waypoints

Ensure that the “Show Waypoints” box is unchecked.

Click the checkbox labeled “Show Waypoints”

All the colored circles indicating waypoints should appear.

9: Stop centering the map

Ensure that the “Auto Center Map” box is checked.

Click the checkbox labeled “Auto Center Map”

The map should stop centering itself relative to the planes, allowing the user

to freely drag the map around without it jumping back to center.

10: Center the map

Ensure that the “Auto Center Map” box is unchecked.

Click the checkbox labeled “Auto Center Map”

The map should begin centering itself relative to the planes. If the user drags

the map around, the map should re-center itself the next time the plane coordinates are updated.

11: Disabling auto zoom

Ensure that the “Auto Zoom” box is checked.

Click the checkbox labeled “Auto Zoom Map”

The map should stop adjusting its zoom to the planes on screen. Note that a

large number of planes need to be running in order for this to have a noticeable effect.

12: Enabling auto zoom

Ensure that the “Auto Zoom” box is unchecked.

Click the checkbox labeled “Auto Zoom Map”

The map should begin adjusting its zoom to the planes on screen. Note that a

large number of planes need to be running in order for this to have a noticeable effect.

13: Selecting a plane (no planes selected)

Locate the label titled “Plane Information” on the right side of the GUI.

Click the drop-down box right below it. Assuming no planes have been

selected, it should say “Select A Plane”.

Click on any plane that appears in the drop-down menu.

The plane's information should show up in the latitude, longitude, altitude,

ground speed, target bearing, waypoint index, and distance to destination fields located below the drop-down box.

14: Switching selected planes

Locate the label titled “Plane Information” on the right side of the GUI.

Click the drop-down box right below it. Assuming a plane has been selected,

it should show the currently selected plane's number in the box.

Click on any plane (other than the one already selected) that appears in the

drop-down menu.

The newly-selected plane's information should show up in the latitude,

longitude, altitude, ground speed, target bearing, waypoint index, and distance to destination fields located below the drop-down box.

15: Hiding all planes

Locate the label titled “Plane Information” on the right side of the GUI.

Click the button labeled “Show Planes” right below it.

A small window should appear with a checkbox labeled “All Planes”. Below it

should be checkboxes with a colored circle labeled “Display Plane” and the plane number. If this is the first time in this menu, all the checkboxes should be checked.

Click the checkbox labeled “All Planes”

All planes on the map should be hidden, as should their trails.

16: Show all planes

Locate the label titled “Plane Information” on the right side of the GUI.

Click the button labeled “Show Planes” right below it.

A small window should appear with a checkbox labeled “All Planes”. Below it

should be checkboxes with a colored circle labeled “Display Plane” and the plane number.

Click the checkbox labeled “All Planes”. It should be unchecked for this test.

All planes on the map should appear, and their trails should not have been

reset.

17: Hiding one plane (All Planes checked)

Locate the label titled “Plane Information” on the right side of the GUI.

Click the button labeled “Show Planes” right below it.

A small window should appear with a checkbox labeled “All Planes”. It should

be checked. Below it should be checkboxes with a colored circle labeled “Display Plane” and the plane number. If this is the first time in this menu, all the checkboxes should be checked.

Click one of the checkboxes corresponding to a plane.

The plane matching the color of the circle next to the checkbox should be

hidden, as should its trail.

18: Hiding multiple planes (All Planes checked)

After completing test 17, continue to deselect planes as before.

The deselected planes should disappear.

None of them should reappear when another plane is hidden.

19: Hiding all planes manually (All Planes checked)

After completing test 18, continue to deselect planes until no planes are

showing.

The “All Planes” checkbox should remain checked.

20: Deselecting “All Planes” when all planes have been hidden

After completing test 19, uncheck the “All Planes” checkbox.

All the planes should remain hidden.

21: Showing a plane (All Planes unchecked)

After completing test 20, click the checkbox next to any hidden plane.

The plane and its corresponding trail should reappear on the map.

“All Planes” should remain unchecked.

22: Showing multiple planes (All Planes unchecked)

After completing test 21, continue to select planes as before.

The “All Planes” checkbox should remain unchecked.

23: Showing all planes manually (All Planes unchecked)

After completing test 22, continue to select planes until all planes are

showing.

The “All Planes” checkbox should remain unchecked.

24: Selecting “All Planes” when all planes are showing

After completing test 23, check the “All Planes” checkbox.

All the planes should remain visible.

25: Closing the Plane Visibility window after hiding planes

Hide a few planes by unchecking their checkboxes.

Close the window by clicking the circular 'x' button in the top left.

The hidden planes should remain hidden.

26: Closing the program with the Plane Visibility window open

Ensure that the GUI is in the Flight Visualization tab.

Locate the label titled “Plane Information” on the right side of the GUI.

Click the button labeled “Show Planes” right below it.

Close the main program by clicking on the window with the map, then

clicking the circular 'x' button in the top right.

The Plane Visibility window should close as well.

27: Switching between tabs

Ensure that the GUI is in the Flight Visualization tab.

Swtich to the Mission Planner tab by clicking the tab next to the one labeled

“Flight Visualization” at the top of the screen.

The mission planning interface should show.

Switch back to the Flight Visualization tab.

The planes and their trails should not have reset.

Mission Planner Tab

For these tests, you should switch to the Mission Planner Tab.

1: Placing a marker

Click anywhere on the map.

A marker should appear with the number 1.

Above the map, the text field should update with the plane ID and the latitude,

longitude, and altitude.

2: Placing multiple markers

After completing test 1, click anywhere else on the map.

A marker should appear with the number 2, and a line should be drawn

between the two markers.

Above the map, the text field should update with the plane ID and the latitude,

longitude, and altitude.

3: Deleting a marker

After completing test 1 or 2, double-click on a marker.

It should disappear from the map.

Its information should be removed from the text field.

If it's not the only marker on the map, lines connecting it to other markers

should be removed.

4: Moving around the map

Click and, without releasing the button, drag the mouse.

The map should move.

Release the mouse.

No marker should be placed.

5: Attempting to edit the text field

Attempt to select the text field and type.

The text field should not accept any input from the keyboard.

6: Editing the text field

Click the button labeled “Edit File”

Attempt to select the text field and type.

The text field should accept input from the keyboard.

The “Edit File” button should read “Done Editing”

7: Writing an invalid course file

While in edit mode, type in a random string of characters.

Click “Done editing”

A warning message box should pop up and tell you it couldn't parse the

course file.

A comment saying that the file has been manually edited should appear in the

text box.

The text box should no longer accept input from the keyboard.

“Done Editing” should now read “Edit File”

8: Manually editing a valid course file

Click “Edit File”

Delete whatever you entered in test 7.

Set some markers on the map. Don't enter anything in the text field.

Click “Done Edting”

A comment saying that the file has been manually edited should be added in

the text box.

The markers should persist on the map.

7: Hiding a path

Set a few waypoints as done in tests 1 and 2.

Look on the right side of the GUI for the “Path Visible” label.

Uncheck one of the checkboxes next to a plane ID.

That plane's path should disappear from the map.

If the path hidden is that of the path that is currently being edited, the

markers will still be visible.

8: Unhiding a path

After completing test 7, check the box that was previously unchecked.

The plane's path should reappear.

9: Adding multiple paths

Place a few waypoints as done in tests 1 and 2.

Under the label “Manual Mission” on the right, click the button labeled “Done

With Plane”.

Under the label “Manual Mission”, the Plane ID value should have

incremented.

The markers for the previous plane should disappear.

The path for the previous plane should still show on the map.

Place two more markers.

The line connecting them should be a different color.

In the text field at the top of the window, another plane ID should have been

added.

10: Editing a previous path

After completing test 9, select another plane by clicking on its plane ID label

in the “Path Visible” list on the right. Do not click the checkbox; the

plane's label should be highlighted, and the checkbox still marked.

Click the “Show Markers for Plane” button below the list.

The map should show the selected plane's markers on its path.

It should hide the other plane markers, but not their paths.

“Plane ID” should be updated to the ID of the plane selected.

11: Setting an altitude

Under “Manual Mission”, enter a number between 0 and 999 in the box

beside “Altitude”.

Set a marker.

The corresponding waypoints for the plane in the text field should reflect the

altitude that was just set.

Note that the altitude is the last number in the row, not the number under the

Altitude heading in the text field.

12: Setting a negative altitude

Try to set a negative altitude in the same way the altitude was set previously.

The box should not accept any keyboard input that isn't a number.

Try to use the arrows to the side of the field to select a number less than 0.

When the plane ID is equal to zero, the down arrow shouldn't accept clicks.

13: Setting a blank altitude

Highlight the contents of the Altitude box and erase them with backspace or

delete.

The 'm' suffix should remain.

Click off of the box, into the map, the text field, or any other component of the

GUI that isn't the background. Note that clicking into the map will place a marker, unless you click and drag.

The altitude should reset itself to the last number that was entered.

14: Setting an altitude greater than 999

Highlight the contents of the Altitude box and erase them with backspace or

delete.

The 'm' suffix should remain.

Try to enter any 4 numbers.

The Altitude box should only accept the first 3.

Try to use the arrows to the side of the field to select a number greater than

63.

When the plane ID is equal to 63, the up arrow shouldn't accept clicks.

15: Adding another plane with a manually selected plane ID

Under “Manual Mission”, set the Plane ID field to a number less than 64.

Click on the map to add a marker.

The plane with the entered ID should appear last in the Path Visible list.

The plane's waypoint information should be written into the text window

and the IDs being written should be in order from least to greatest.

16: Entering a negative plane ID

Select the Plane ID text field.

Try to type in a number less than 0.

It shouldn't take any input that isn't a number.

Try to use the arrows to the side of the field to select a number less than 0.

When the plane ID is equal to zero, the down arrow shouldn't accept clicks.

17: Entering a plane ID greater than 63

Select the Plane ID text field.

Try to type in a greater than 63.

It will accept the 6, but not another number greater than 3.

Try to use the arrows to the side of the field to select a number greater than

63.

When the plane ID is equal to 63, the up arrow shouldn't accept clicks.

18: Getting weather information

Click on the “Wind Conditions” button.

A popup should appear with a verbal wind rating (ex. “Calm”), wind direction,

wind speed, and gust, along with the Weather Underground logo on the right.

Close the window with the x in the top right.

19: Cancelling a map reset

Place a few markers on the map.

Click “Reset Map”

A message box should pop up. Click “Cancel”

The map and markers shouldn't change.

20: Resetting the map without saving

Place a few markers on the map.

Click “Reset Map”

A message box should pop up. Click “No”

The map should now be blank and the output file preview should only say

“This file will be automatically generated.”

21: Resetting the map after it's just been reset

Immediately after test 20, click “Reset Map”

Nothing should happen.

22: Closing the map reset message box

Place a few markers on the map.

Click “Reset Map”

A message box should pop up. Click the red x in the top left to close the box.

The map and markers shouldn't change.

23: Resetting the map and saving

Place a few markers on the map.

Click “Reset Map”

A message box should pop up. Click “Yes”.

A file selection window should open to the courses folder.

Type in a filename at the top and click “Save.”

Navigate to the courses folder. The file should be in there with name given to

it and the data as shown in the text preview box.

24: Resetting the map, saving, closing at the save menu

Place a few markers on the map.

Click “Reset Map”

A message box should pop up. Click “Yes”.

A file selection window should open to the courses folder.

Close the file selection window by clicking the 'x' in the top left.

The map should now be blank and the output file preview should only say

“This file will be automatically generated.”

No new file should be added.

25: Setting a latitude

Underneath the map, select the text field to the right of “Latitude”.

Type in a number or decimal greater than -90 and less than 90.

The text field should accept the input.

26: Setting a latitude greater than 90

Underneath the map, select the text field to the right of “Latitude”.

Type in a number or decimal greater than 90.

The text field should accept the 9, but no other number greater than 0.

27: Setting a latitude less than -90

Underneath the map, select the text field to the right of “Latitude”.

Type in a number or decimal less than 90.

The text field should accept the -9, but no other number greater than 0.

28: Entering invalid input into latitude

Underneath the map, select the text field to the right of “Latitude”.

Try to type non-numerical characters in the text field.

The text field shouldn't accept the input.

29: Setting a longitude

Underneath the map, select the text field to the right of “Longitude”.

Type in a number or decimal greater than -180 and less than 180.

The text field should accept the input.

30: Setting a longitude greater than 180

Underneath the map, select the text field to the right of “Longitude”.

Type in a number or decimal greater than 180.

The text field should accept the 18, but no other number greater than 0.

31: Setting a longitude less than -180

Underneath the map, select the text field to the right of “Longitude”.

Type in a number or decimal less than -180.

The text field should accept the -18, but no other number greater than 0.

32: Entering invalid input into longitude

Underneath the map, select the text field to the right of “Longitude”.

Try to type non-numerical characters in the text field.

The text field shouldn't accept the input.

33: Setting the center of the map

Underneath the map, select the text field to the right of “Latitude”.

Enter the number 32.6.

Now select the text field next to “Longitude”.

Enter the number -85.49.

Click “Set Center”

The map should be centered near Jordan-Hare Stadium.

34: Saving a location, cancel the save

Underneath the map, select the text field to the right of “Latitude”.

Enter the number 32.6.

Now select the text field next to “Longitude”.

Enter the number -85.49.

Click “Save Location”

A popup should appear asking for a name.

Close the popup by clicking “Cancel” or the red 'x' on the top left.

The location should not be saved. If this is the first location you've tried to

save, the “Saved Locations” drop down box should be greyed out.

35: Saving a location

Underneath the map, select the text field to the right of “Latitude”.

Enter the number 32.6.

Now select the text field next to “Longitude”.

Enter the number -85.49.

Click “Save Location”

A popup should appear asking for a name.

Type in a name.

Click “Okay”

The location should be saved. The “Saved Locations” drop down box should

not be greyed out, and should be populated with any saved locations.

36: Saving a location, blank lat and long fields

Ensure both the latitude and longitude fields are empty.

Click “Save Location”

A popup should appear asking for a name.

Type in a name.

Click “Okay”

The current center of the map should be saved. The “Saved Locations” drop

down box should not be greyed out, and should be populated with any

saved locations.

37: Loading a saved location

After completing test 35, click the “Saved Locations” drop down box.

Select the location you saved in test 35.

The map should center on the latitude and longitude you entered. In this case,

it should be Jordan-Hare Stadium.

38: Edit a saved location

After completing test 35 and 36, click the “Saved Locations” drop down box.

Select “Edit Locations”.

A window should open with the names of the saved locations, their latitudes

and longitudes, and a delete checkbox beside each in a table format.

For now, just change one of the names and close the window with the red 'x'

in the top left.

Click on the “Saved Locations” dropdown box.

Whatever changes to the name you made should be reflected in the

dropdown box.

39: Edit a saved location's latitude and longitude

After completing test 35 and 36, click the “Saved Locations” drop down box.

Select “Edit Locations”.

A window should open with the names of the saved locations, their latitudes

and longitudes, and a delete checkbox beside each in a table format.

Change the latitude and/or longitude of one location and close out the

window with the red 'x' in the top left.

Click on the “Saved Locations” dropdown box and select the location you

edited.

The map should center on the new latitude and longitude.

40: Delete a saved location

After completing test 35 and 36, click the “Saved Locations” drop down box.

Select “Edit Locations”.

A window should open with the names of the saved locations, their latitudes

and longitudes, and a delete checkbox beside each in a table format.

Check the delete checkbox next to one of the locations and close out the

window with the red 'x' in the top left.

Click on the “Saved Locations” dropdown box.

The location you deleted should no longer appear.

41: Saving a location with a blank name

Click “Save Location”  
 Click “Save”

A warning box should appear, explaining that blank names aren't allowed.

42: Overriding an existing location with a new location of the same name

Click “Save Location”

Enter the letter 'A' in the name field.  
 Click “Save”

Move the map to a new center.

Click “Save Location”

Enter the letter 'A' in the name field.  
 Click “Save”

Move the map.

Click “Saved Locations”. 'A' should only appear once.

Select 'A'.

The map should center on the most recent version of A that was saved.

43: Saving a course

Click “Save File”

A file selection window should appear with the courses folder already

selected.

Type in a name for the file in the box at the top and click “Save”.

The file should appear as a .course file in the courses folder. It should contain

the exact same text as shown in the file review text pane.

44: Saving a course, cancel before save

Click “Save File”

A file selection window should appear with the courses folder already

selected.

Click “Cancel”

No file should be saved.

45: Loading a course, cancel before load

Click “Load File”

A file selection window should appear with the courses folder already

selected.

Select a course file and click “Cancel”

No file should be loaded.

46: Loading a course

Click “Load File”

A file selection window should appear with the courses folder already

selected.

Select a course file and click “Open”

The selected file should be loaded.

47: Making a path file

Place some markers on the map.

Under Options, select the radio button next to “Path File”.

A popup will appear asking if you want to save. Click “Yes”.

A file selection window should appear with the courses folder already

selected.

Enter a name for the file. It should save it as a course file.

The map and the file preview pane should reset and the radio button next to

“Path File” should be selected.

48: Switch to path file with clear map

Reset the map. Don't save any changes.

Under Options, select the radio button next to “Path File”.

The radio button next to “Path File” should be selected.

49: Switching back to a course file

After completing test 45, place some markers on the map.

Under Options, select the radio button next to “Course File”.

A popup will appear asking if you want to save. Click “Yes”.

A file selection window should appear with the courses folder already

selected.

Enter a name for the file. It should save it as a path file.

The map and the file preview pane should reset and the radio button next to

“Course File” should be selected.

50: Switch to course file with clear map

Reset the map. Don't save any changes.

Under Options, select the radio button next to “Course File”.

The radio button next to “Course File” should be selected.

51: Switch to random mission generation from manual path file

Reset the map. Don't save any changes. Select “Manual Mission” if it isn't

already selected.

Under Options, select the radio button next to “Path File”.

Put some markers on the map.

Click the radio button next to “Random Mission”

A popup will ask if you want to save the file. Click “Yes”

A file selection window will appear with the courses folder open and the path

file type selected.

Enter a file name and click “Save”

The file should be saved with the name you gave it and the contents exactly as

shown in the file preview.

The map should be clear.

The radio button next to “Random Mission” should now be selected.

The radio button and label saying “Path File” should be greyed out and should

not accept clicks.

The radio button next to “Course File” should now be selected.

Everything in the “Manual Mission” section should be greyed out.

Under “Random Mission”:

Waypoints should default to 50

Planes should default to 5

Field Size should default to 500 m2

Altitude Min should default to 400 m

Altitude Max should default to 400 m

52: Switch to random mission generation from manual course file

Reset the map. Don't save any changes. Select “Manual Mission” if it isn't

already selected.

Under Options, select the radio button next to “Course File” if it isn't already

selected.

Put some markers on the map.

Click the radio button next to “Random Mission”

A popup will ask if you want to save the file. Click “Yes”

A file selection window will appear with the courses folder open and the

course file type selected.

Enter a file name and click “Save”

The file should be saved with the name you gave it and the contents exactly as

shown in the file preview.

The map should be clear.

The radio button next to “Random Mission” should now be selected.

The radio button and label saying “Path File” should be greyed out and should

not accept clicks.

The radio button next to “Course File” should now be selected.

Everything in the “Manual Mission” section should be greyed out.

Under “Random Mission”:

Waypoints should default to 50

Planes should default to 5

Field Size should default to 500 m2

Altitude Min should default to 400 m

Altitude Max should default to 400 m

53: Switching from path file to random mission, blank map

Reset the map. Don't save any changes. Select “Manual Mission” if it isn't

already selected.

Under Options, select the radio button next to “Path File” if it isn't already

selected.

Now select the “Random Mission” option.

The radio button next to “Random Mission” should now be selected.

The radio button and label saying “Path File” should be greyed out and should

not accept clicks.

The radio button next to “Course File” should now be selected.

Everything in the “Manual Mission” section should be greyed out.

Under “Random Mission”:

Waypoints should default to 50

Planes should default to 5

Field Size should default to 500 m2

Altitude Min should default to 400 m

Altitude Max should default to 400 m

54: Switching from course file to random mission, blank map

Reset the map. Don't save any changes. Select “Manual Mission” if it isn't

already selected.

Under Options, select the radio button next to “Course File” if it isn't already

selected.

Now select the “Random Mission” option.

The radio button next to “Random Mission” should now be selected.

The radio button and label saying “Path File” should be greyed out and should

not accept clicks.

The radio button next to “Course File” should now be selected.

Everything in the “Manual Mission” section should be greyed out.

Under “Random Mission”:

Waypoints should default to 50

Planes should default to 5

Field Size should default to 500 m2

Altitude Min should default to 400 m

Altitude Max should default to 400 m

55: Generating a random mission

Select “Random Mission”. Don't save any changes. Leave all the fields to

default.

Click “Generate Mission” in the bottom right.

A box should be drawn on the map in black. None of the paths of the planes

should go outside this box.

In the file preview pane, the file should have comments stating that the

course was randomly generated and detailing the settings that were used. The settings should match those in their respective text boxes. The coordinates for the northwest corner of the box should also be displayed.

In the Path Visible list, planes 0 through 4 should be present.

56: Generating a random mission after one has already been generated, cancel

Immediately after test 53, click “Generate Mission”

A popup will ask if you wish to save the file. Click “Cancel”

The map should not change.

57: Generating a random mission after one has already been generated, don't save

Immediately after test 54, click “Generate Mission”

A popup will ask if you wish to save the file. Click “No”

A new random course should be generated.

58: Generating a random mission after one has already been generated, cancel before save

Immediately after test 53, click “Generate Mission”

A popup will ask if you wish to save the file. Click “Yes”

A file selection window will appear with the courses folder open and the

course path file type selected.

Click “Cancel” or close the window with the red 'x' in the top left

The map should not change.

59: Generating a random mission after one has already been generated, save

Immediately after test 53, click “Generate Mission”

A popup will ask if you wish to save the file. Click “Yes”

A file selection window will appear with the courses folder open and the

course file type selected.

Enter a file name and click “Save”

The file should be saved with the name you gave it and the contents exactly as

shown in the file preview.

A new mission should be generated and display on the map.

60: Random mission, change number of waypoints

Reset the map. Don't save any changes.

“Random Mission” should still be selected.

Enter a number in the Waypoints field. It should not allow anything other

than a number greater than 0 or less than or equal to 99. If you enter 0, clicking off the text field will set the number back to the last valid number.

Click “Generate Mission” in the bottom right.

In the file preview pane, the number of waypoints per plane should match the

number you entered.

61: Random mission, change number of planes

Reset the map. Don't save any changes.

“Random Mission” should still be selected.

Enter a number in the Planes field. It should not allow anything other than a

number greater than 0 or less than or equal to 64. If you enter 0, clicking off the text field will set the number back to the last valid number. If you enter a 6 as the first number, it should not allow the next number to be greater than 4.

Click “Generate Mission” in the bottom right.

In the file preview pane, the number of planes should match the number you entered.

62: Random mission, change field size

Reset the map. Don't save any changes.

“Random Mission” should still be selected.

Enter a number in the Field Size field. It should not allow anything other than

a number greater than 100 or less than or equal to 999. If you enter a number less than 100, clicking off the text field will set the number back to the last valid number. The text field should only accept a maximum of 3 digits.

Click “Generate Mission” in the bottom right.

In the file preview pane, the field size should match the number you entered.

The black bounding box on the map should grow or shrink to reflect the

change in area.

Depending on the number you entered, the number of waypoints, and the

number of planes, some planes (and their paths) may go outside the bounding box.

63: Random mission, change altitude min size

Reset the map. Don't save any changes.

“Random Mission” should still be selected.

Enter a number in the Altitude Min field. It should not allow anything other

than a number greater or equal to 0 or less than or equal to 999. The

text field should only accept a maximum of 3 digits.

Click “Generate Mission” in the bottom right.

In the file preview pane, the waypoint altitudes should vary between the

minimum altitude you entered and the max altitude.

64: Setting Altitude Min to above Altitude Max

Enter a number greater than Altitude Max in the Altitude Min field.

Altitude Max should set itself to Altitude Min.

65: Random mission, change altitude max size

Reset the map. Don't save any changes.

“Random Mission” should still be selected.

Enter a number in the Altitude Max field. It should not allow anything other

than a number greater or equal to 0 or less than or equal to 999. The text field should only accept a maximum of 3 digits. If the number you enter is less than the minimum altitude, clicking off the text field should revert the maximum back to the last valid number.

Click “Generate Mission” in the bottom right.

In the file preview pane, the waypoint altitudes should vary between the

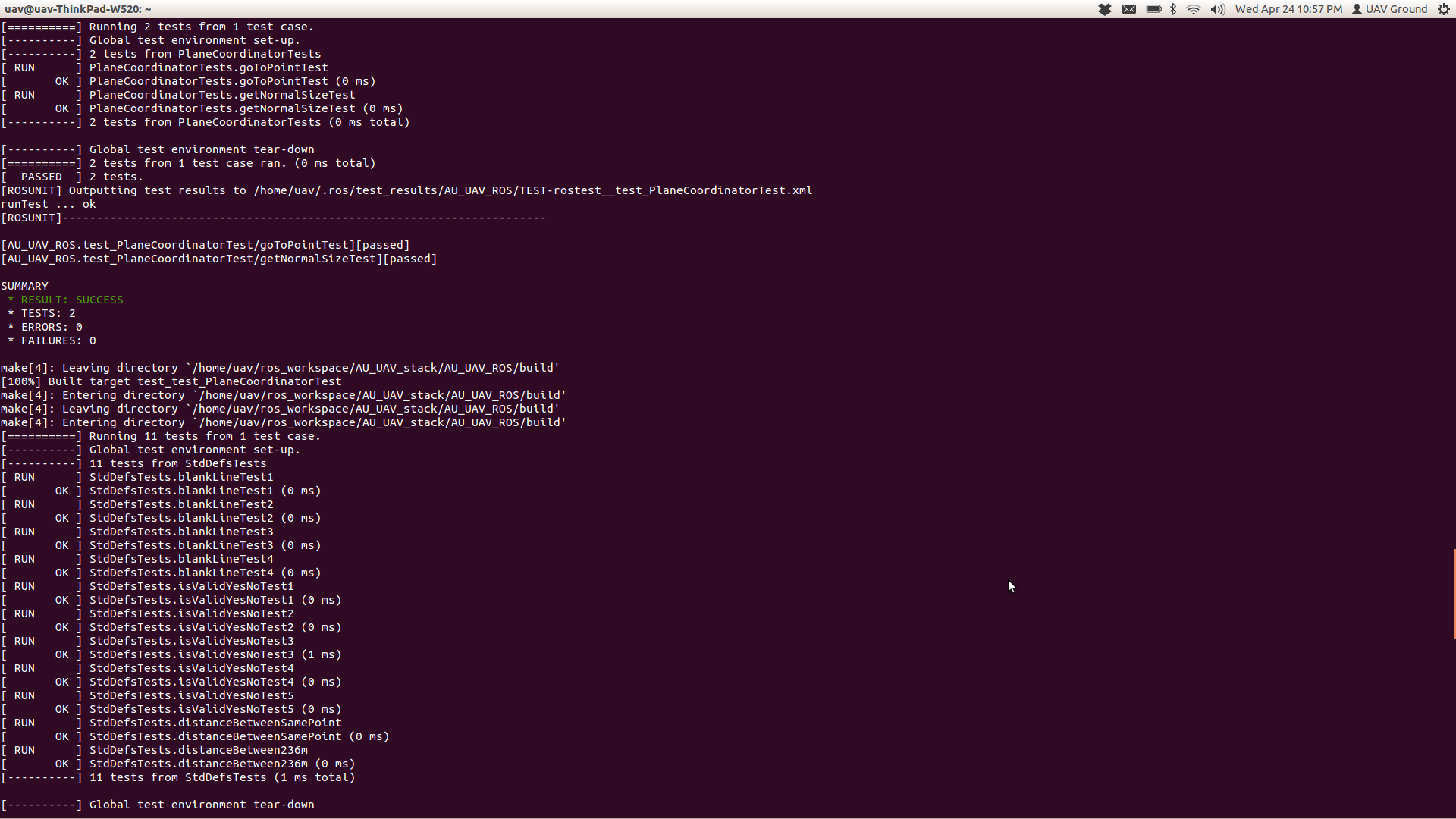
maximum altitude you entered and the minimum altitude.

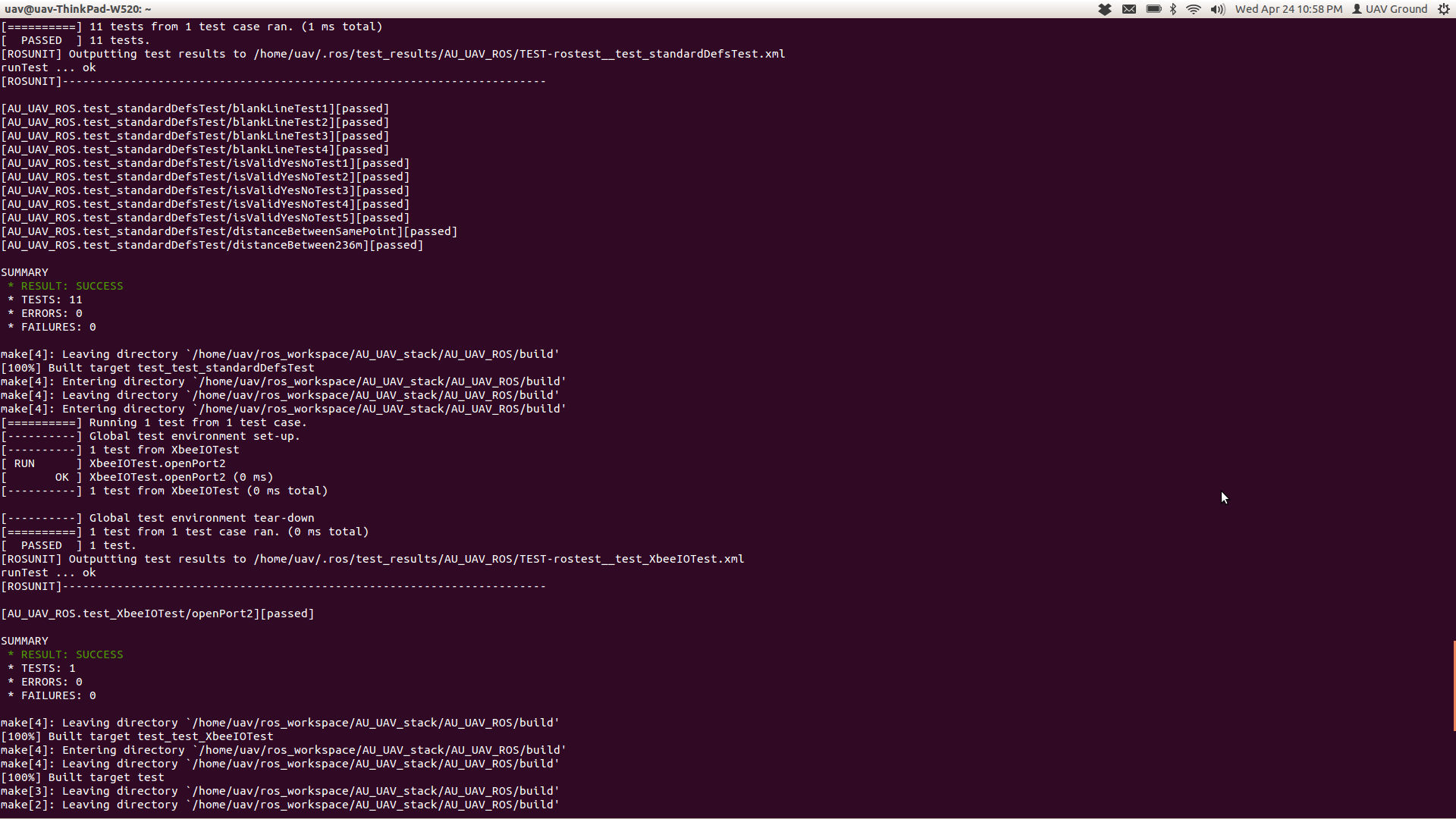
**Packet Loss Testing**

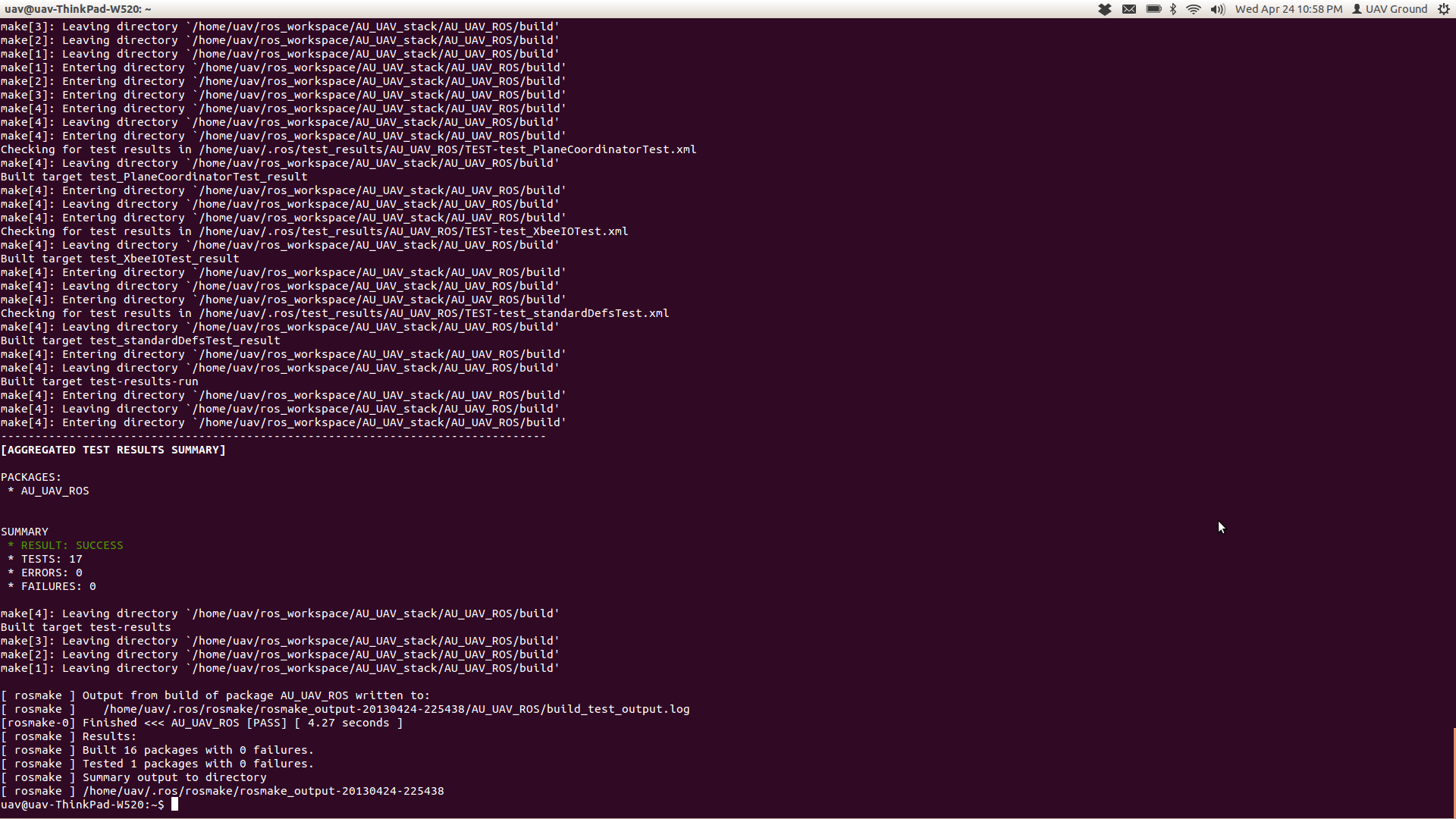
A packet is a formatted unit of data that is carried through a network. Whenever a packet needs to be transmitted, there will always be a probability that it could be dropped. It is imperative to determine a high enough frequency to allow to planes not to hit, but a low enough frequency to not lose too much information. The packet loss rate of our system needed to be tested to determine this frequency. To reach this goal, we had to determine the amount of time needed to run the test. We ran a packet loss test for five UAVs, and stopped when the packet loss rates converged. This gave us the time we needed. We continued to test by running four frequencies, 1hz, 10hz, 25hz, and 50hz, with a single plane. We added a plane to the network and ran the system again. We continued this pattern with all five of the planes. With any system, the higher the frequency, the more congested the network becomes. As the network becomes more congested, packet loss rate will inevitably increase. We determined that with only one plane, or a small number of planes, the frequency does not matter. The one plane test validated this, as the rate was always zero percent. We realized that if the frequency is low enough, that the number of planes is negligible. The tests validated this by showing that the packet loss rate was constant during low frequencies. We understand the frequency should change with the amount of planes in the network. With five planes, we decided to set the frequency between 50hz and 25hz. With any amount of data being passed, packet loss rate needs to be controlled.

5c. Test Logs

**ROS System**







**GUI**

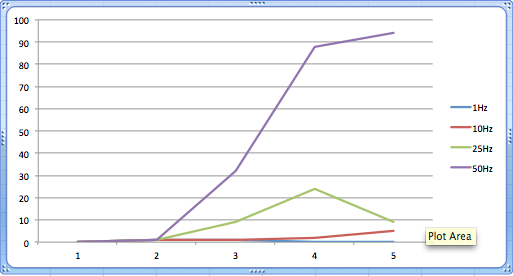
**Cycle 2**

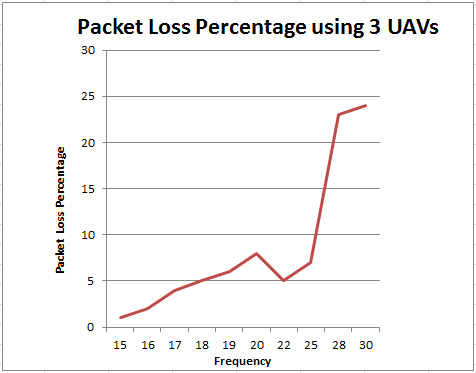
The GUI team is currently working on integrating an automatic testing framework to run their acceptance and user tests. This has proven to be a very difficult endeavor given the QT/ROS hybrid nature of the GUI. Testing frameworks designed for QT cannot properly link with the ROS headers and vice versa. We have made great progress and believe we finally have a system that will allow for some automated unit tests. Unfortunately, it was too late to actually write test cases with this new framework. Another hurdle is the nature of our program. Several of the program’s functions are used to update the Google Map. We have not yet found a system that allows us to tell if an object on the map has been manipulated. We will continue to investigate any options that may allow us to integrate with the map. Finally, the focus of the team this cycle has been to ensure that all of the required features have been implemented. Test cases have been written along the way, as the process requires, but we felt that it was more important to develop required features than to pursue a testing framework. As such, our test cases are now manually run after each major version update. This has led to the discovery of several UI errors, errors with the recording of altitude, and the introduction of errors after refactoring. Cycle 3 will allow us the time we need to begin focusing on automation of these tests.

**Cycle 3**

The decision was made to abandon the idea of an automated testing framework at the advice of Mr. Denis, our teaching assistant. The system is simply too complex and too unique to be able to quickly get started with an off the shelf solution. Instead, the GUI team focused on writing extensive manual test cases that can be executed by hand, and that can later be incorporated into a framework.

**Packet Loss Testing**





6. Risk Management – Michael Widick

For our system to visualize UAVs on Google Maps, we must have Internet access. Without Internet access, the system will not be able to load the map data from Google and we will be unable to visualize the planes while they are flying. This is no longer a risk because we have the availability to tether our mobile devices to use as a WiFi hotspot.

Throughout this project, we will rely on pre-existing software to work as intended. There are some minor risks involved with trusting that these systems work. Although this system has been around for some time, there is no way to insure it is bug free. This may become a risk if the autopilot system doesn’t work as intended. To mitigate this risk, we can review the code for bugs. Once bugs are found, we can assess the importance of that particular bug and fix the issue.

For the ground station to monitor UAVs, packets are being sent down to the ground station constantly. With any wireless communication, packet loss can become an issue. Packet loss can occur because of two reasons: the UAVs are sending packets too frequently or if the number of planes is too high, the communication line can become flooded. To mitigate these risks, we can alter the frequency of packets sent from each plane and the number of UAVs. Moreover, we intended to do some extensive testing to find a sufficient telemetry rate with a minimal packet loss rate.

6a. Lessons learned

**Cycles 1 & 2**

One of the greatest ways to learn something is to become involved in the project. When writing software for UAVs, the way that the system is tested is by flying the UAVs and seeing if it is sending and receiving messages. One thing we discovered was the ability for packet loss to occur. Through this, we became educated to the possible interference of packet loss. We began a trial and error process for a couple of weeks, but the packet loss rate did not change. After much thought, we realized the packet loss rate was wrong, due to the CRC check on the ground station side. We learned that we must think of every networking problem, before assuming the problem lies in a particular project. Also, the DIYDrones community is very helpful for troubleshooting, so we learned to post discussion topics as soon as possible to their forum for feedback.

Something that is always involved in flying a plane is the weather. Severe weather at international airports delay, if not cancel, plane flights. One problem that has occurred in our project is that the weather did not allow us to fly the planes. One learning experience we established was to check the weather before flying a UAV. Checking the weather before flying a UAV plane is imperative.

In the beginning of our project, we were allowed to fly on Auburn campus. As the semester proceeded, people began realizing the dangers of the location. We had to begin to test in Tuskegee. This required that we plan the time, and ask for permission. One lesson that we learned from this experience is to never have anything set in stone.

Throughout the semester, the DIYDrones community will release updated versions of the ArduPlane code. This will require us to update our ArduPlane code to have the latest version. To ensure that we do not lose any of our progress, we discovered that it is important to keep versions throughout the project. This will ensure that if we switch to a new ArduPlane and break the code, we will always have a working version to fall back upon.

A system that is wireless will undergo packet loss, and so packet loss testing must be done. The packet loss rate becomes steady as time increases. Also, as time gets smaller, the packet loss rate becomes unpredictable. Therefore, we needed some way of determining this time. We began running the system with the most planes possible, and their highest frequency possible. We learned that an ideal test time is around five minutes. Although, with more testing, we believe this time can be reduced. With the number of planes being small, the frequency becomes negligible. As the frequency increases, the packet loss rate of each plane will increase. The frequency of the sent packet and the number of senders in the system both need to be monitored.

To integrate a GUI, we had to integrate QT and ROS. This required the GUI developers to install a working version of ROS. ROS has many versions, and our ROS system is built for the fuerte version. Initially, the GUI developers tried using the groovy version, but quickly realized that fuerte and groovy were not compatible. Therefore, we learned to always ensure that everyone on the team is using the same version of software.

**Cycle 3**

While we were testing the planes on April 23, 2013, we discovered that data tethering with an iPhone works better with a MacBook than it does with the IBM computer we had been using. The MacBook can then forward the data connection to a virtual machine running Linux. The loading of the GUI was reduced by a substantial amount. With the IBM laptop it took the map around six minutes to load, while the Mac computer took less than ten seconds. The lesson learned is that the people involved with the project in the future should use a MacBook when conducting test flights.

Another lesson that we learned in cycle three was to make sure the computer that will be used for the test flight has a charged battery. We had a problem during our last test flight, because the computer that we normally test with only had sixteen minutes of battery power left. This is how we discovered that the MacBook was faster for loading than the IBM. Also, the UAV’s battery needs to be fully charged concluding every flight. The problem could have resulted in a cancelled test flight or a ruined battery.

With any software system, the only way for the writer to realize if the system is correct is by testing the software. We began realizing this late in the state of the project causing us to have to concentrate on producing test cases very rapidly. Testing should be implemented in the middle of the project. It would be best if testing began at the beginning of the project, if possible. Testing is a vital part of software process, and the consumer could not know that the system works correctly without these tests.

Estimation is the ability to be able to plan the time that is required to accomplish a task. With any project, estimation is important. We had a problem with one of our team members having to partake in overtime effort because we underestimated that amount of time needed to implement saving and updating field locations in the GUI. With correct estimation, this could have been avoided.

With any aircraft vehicle, the speed of the wind is a controlling factor. We tried to fly the UAVs in ten mph winds, and we had a hard time controlling the aircraft. We discovered that to be able to test the flights safely, the speed of the wind should be less than five mph. The lesson that we learned is to check the weather and possibly have a wind instrument with us to determine if the test is possible.

While doing tests with the GUI we realized that we needed an easy way to distinguish between real and simulated UAVs. This was a very important lesson learned, because this functionality improved our live flight tests. We made the decision to make the fill color of real planes black and the fill color of simulated planes off-white. Also, to decrease the loading time, we decided not to use images for the plane icons and instead dynamically draw them using vector graphics.

7. Source Code

Please refer to the CD for our source code.

8. Version Description – David Jones (ROS), Jacob Conaway (GUI)

1. ROS System
   1. 0.5 – “Maui” – The ROS system fully implements simulated UAVs, but there is not a communication with real UAVs.
   2. 0.7 – The ROS system can receive telemetry updates from real UAVs through the XbeeIO node.
   3. 0.9 – The ROS system can send commands to real UAVs through the XbeeIO node.
   4. 1.0 – “O’ahu” – The wireless Xbee communication works for a single real UAV.
   5. 1.5 – “Kauai” – The ROS system can handle one UAV with multiple simulated UAVs
      1. This version introduces collision avoidance into real UAVs
   6. 2.0 – “French Frigate Shoals” – The ROS system is now integrated with the GUI system. The GUI now initiates the ROS system.
2. GUI System
   1. 0.5 - “Haiti” - The map updates a single generic marker based off of fake coordinates.
   2. 0.7 - The coordinates are no longer fake, but are now received from the ROS topic.
   3. 0.8 - The generic marker was replaced with a plane icon. The plane icon does not correctly orient itself.
   4. 1.0 - “Dominican Republic” - The system is now working with multiple planes.
   5. 1.5 - "Tobago" The plane icon correctly orients itself based off of the plane’s bearing.
   6. 2.0 - “Nevis” - Each plane’s path is now visible on the map.
   7. 2.1 - The destination waypoint for each plane is now visible on the map.
   8. 2.2 - Each path and waypoint now has a color that is unique to the associated planes.
   9. 2.4 - The plane icon had previously flickered as the plane’s bearing changed. There are 72 plane icons and each time the bearing changed by five degrees a new image had to be loaded from a website. In order to solve this issue, all plane images are now loaded as the program starts. This loading takes roughly five seconds, in which time the GUI remained blank. A loading screen was added so that the user would not feel like the application was non-responsive.
   10. 2.5 - The GUI has been updated to give the user the ability to toggle the path and waypoint information.
   11. 2.6 - The map no longer auto centers on each telemetry update. Instead, it updates every 5 seconds.
   12. 2.7 - The GUI has been updated to allow the user to disable the auto center feature.
   13. 2.8 - A new auto fit feature has been added that automatically adjust the zoom level of the map to best fit all of the planes.
   14. 2.9 - The option to disable auto fit was added. The decision was made to automatically disable auto fit when auto center is disabled.
   15. 2.95 - The plane information labels have been updated with a more pleasing visual style.
   16. 3.0 - “Saint Kitts” - A mission planner tab has been added that allows for a simple path file to be created.
   17. 3.2 - Various user interface components were updated to allow for a more pleasing design. A reset flight paths button was added to help reduce visual clutter on the map.
   18. 3.5 - Course files can now be created using the mission planner.
   19. 3.7 - The ability to edit and delete individual waypoints was added to the mission planner.
   20. 3.9 - The different flight paths are now color coded in the mission planner.
   21. 4.0 - “Mustique” - A random mission can now be generated using the mission planner.
   22. 4.2 - A minor bug fix involving incorrect altitudes being displayed in the resulting file.
   23. 4.5 - To increase performance, plane icons have been changed to vector graphics that can be rendered on the fly rather than .png files that have to be loaded.
   24. 4.7 - The ability to hide/show all planes and their paths was added in an effort to cut the clutter of the display.
   25. 4.8 - A wind conditions button was added to the mission planner tab that displays the current wind info for the center of the map.
   26. 4.9 – The required WeatherUnderground logo was added to the weather data.
   27. 5.0 – “Bajo Nuevo Bank” – The GUI can now be launched using roslaunch so that the GUI and the ROS framework can be started simultaneously.
   28. 5.2 – The ability to chose a course file with the GUI was added. Previously, the course file name had to be entered in the terminal window.
   29. 5.4 – To better distinguish between real and simulated planes, the fill colors of each were changed. Simulated planes now have a white fill, while real planes have a black fill.
   30. 5.6 – The planner map markers were changed from using a deprecated Google Charts API service to a group of third party icons. The API service worked great, but the team was worried that Google would terminate the service without notice.
   31. 5.8 – An “About Us” dialog box was added with information about the team and a required icon for the planner map marker images.
   32. 6.0 – “Jost Van Dyke” – The ability to save and load field locations in the mission planner was added.
   33. 6.5 – The ability to edit and delete the saved field locations was added.