RAMPART

Remote Aerial Mission Planning and Radio Tracker

Software Design Document

February 3, 2020

Team Members

Eric Gault

Samuel Gilb

Keller Mikkelson

Nathaniel Zeleny

Team Sponsor: Dr. Michael Shafer

Team Mentor: Mahsa Keshavarz

Version: 2.0

Table of Contents

1 Introduction

Wildlife radio telemetry is a very important tool for tracking the movement of animals attached with a transmitter. The reason why scientists want to track these animals is to detect significant behavioural and positional variations of any animal attached with a transmitter. These variations can help track environmental changes on a wide scale depending on the amount of animals. There are two main methods to tracking animals in this way, GPS tracking as well as very high frequency (VHF) tracking. Unsurprisingly, the use of VHF radio tags is becoming the standard due to their low cost and the exponentially growing pool of VHF-tagged specimens. This ensures that the VHF technology, and in turn, our work on this project, will be used at the benefit of biologists and mechanical engineers by being able to gather transmitted data faster than the conventional method of having to use a handheld receiver. While an overall positive tracking method, there are some problems with VHF. One of the problems with this approach to data gathering is the certain corporeal risks required to go and search for these VHF signals. Currently, the main way to gather the data from these VHF tags requires someone to go out into the potentially dangerous and secluded areas where animals have been tagged and use either handheld scanners or scanners attached to cars if the area permits vehicles.

Our sponsor, Dr. Michael Shafer, and his team have spent the last three years working on integrating a VHF receiver to an unmanned aerial vehicle, also known as a drone. They are currently being funded by a grant from the National Science Foundation to help develop a drone with the ability to track wildlife. Their goal is to decrease the cost of having to locate these animals by using the drone, which can perform a more rapid, efficient, and complete scan of an area when compared to a human. The way our client currently gathers data is to first go out into the field in the general area in which the tagged animal he is looking for is located. He will then use his laptop and open up Mission Planner, an open source flight planning software where he plots the route the drone will be taking. He then opens a program he created himself that connects with the drone so that he can send the correct configuration file to the drone to match the data he is collecting as well as receive status updates from the drone. The drone will then fly its path, collect the data, and then return to the starting point where Dr. Shafer will collect it and then return to his office to process the new data. Here are some specific steps in this workflow that we would like to improve on:

- ➢ Having to use two separate applications requires constant swapping to ensure maximum benefit from both pieces of software. Out in the field our client only has his laptop, therefore he is unable to both track the messages coming from the drone as well as where the drone is mid-flight at the same time.
- ➢ The custom application developed by Dr. Shafer has a very slow launch time and can take five to ten minutes to initially load. This slows down his workflow every time he wishes to go out into the field, costing him hours of time.
- ➢ How the drone configuration is handled, this could result in an error causing the drone to crash while losing all gathered data and potentially damaging valuable hardware.

Our solution to our client's problems requires taking all of his preflight software that he developed and recreating it in an open source flight planning software. The end result is a flight planning software that contains all of the features from our clients preprocessing software. This allows him to only need to use one program that has twice the functionality as before. This will help solve our clients problem by:

- ➢ Only requiring one program without losing any functionality.
- ➢ Increasing startup time to seconds which saves minutes each time our client wants to start gathering data.
- ➢ Better managing configuration files to ensure data is uploaded correctly.

The key requirements for this software are a fully developed configuration manager as well as communication with the drone via FTP and UDP. The functional requirements that we can derive from this are:

- ➢ Configuration file generator:
    - ○ Be able to create configuration files as well as import previously created ones from within the GUI.
    - ○ Provide input fields for configuration.
    - ○ Process user input into configuration file.

- ➢ FTP connection with drone:

    - Be able to connect to the FTP server the drone hosts before a flight.
    - Change FTP address
    - Display FTP's file tree.
    - Upload configuration files.
    - Download telemetry data.

- ➢ Heartbeat Terminal:
    - Be able to receive heartbeat messages from the drone that can be displayed.
    - Change UDP address.
    - User-side start/stop data collection.

Non-functional requirements for our solution will be:

- ➢ A startup time of 90 seconds or less.
- ➢ Documentation of code and use modified software needs to be as simplified as possible.
    - This is to ensure our client and his coworkers can better understand the changes we made and can further modify them to better suit their process if needed.

Environmental requirements for our solution are:

- ➢ Must fit with current drone standards.
    - We must continue with the same decisions in regards to the configuration file to ensure it will work with our clients drone.
- ➢ New GUI must be similar in look to the old one for consistency.

2 Implementation Overview

Now that we have talked about the problem our client is having, as well as what our proposed solution is, we will now be going into detail on how we plan on implementing the requested changes.

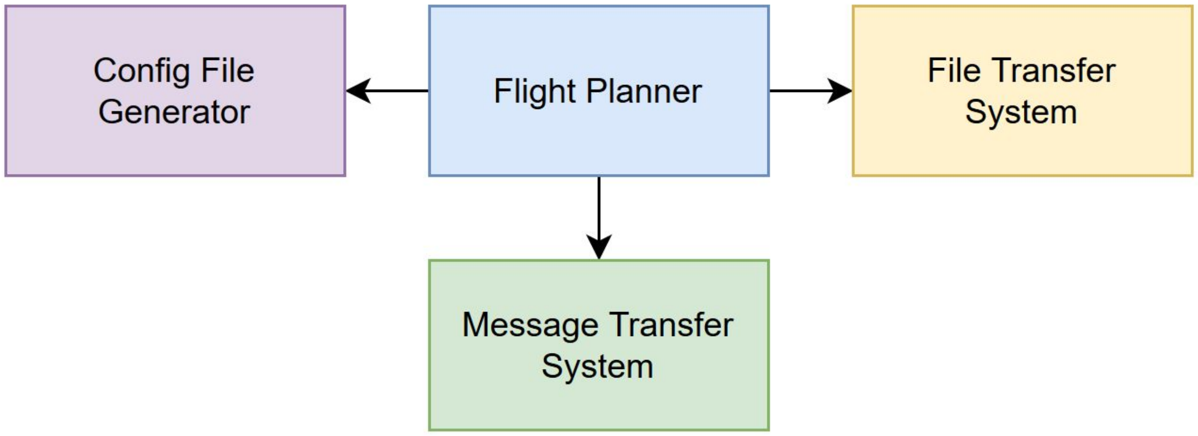

Diagram 1.1 Solution Vision

The solution to our client's program is to develop our own modification of an already developed flight planning program. We will accomplish this by redeveloping software that our client has already created in MATLAB and rewrite it so that it will work with in conjunction with this new software. This will give us one piece of software that does everything that our client needs. Our client's preprocessing software contains config file generation and FTP (File Transfer Protocol) upload/download to be able to send and receive files and messages from the drone. We will also be adding the functionality of listening for the drone's heartbeat messages sent over UDP (User Datagram Protocol). We will be taking a very modular approach to make the software easy to change unique parts to allow for better testing as per our client's needs.

2.1 Key Technologies

Because we will be adding onto an already developed piece of software, many decisions about what to use for development has already been chosen by the developers of the software we will be adding to. There were two main options when deciding which software to use as a base, QGroundControl or Mission Planner. We ultimately decided to work with QGroundControl. If you would like to know more about our reasons behind this decision, please read our Technological Feasibility document located on our capstone project website.

Using QGroundControl locks us into a couple development environments to ensure we are creating software that will work with the preexisting architecture. The main environment that this choice locks us into is that we now are required to use Qt, specifically version 5.12.5 as stated by the developer in their documentation. Qt is an open source widget toolkit that is used for creating graphical user interfaces as well as cross platform programs. It contains many frameworks within it to aid in the development process. Most of our code will be written in the C++ language as Qt mainly supports the usage of this language and QGroundControl is written in C++ through Qt as well. QtCreator is the main way of utilizing Qt's features, as it is a cross platform integrated development environment (IDE) to assist with using Qt. It contains a large amount of development and design tools to aid in the use of Qt and will be our main development tool for this project.

As our client's computer runs the Microsoft Windows operating system (Windows for short) we will mainly be developing in the Windows environment and we will exclusively test large changes in the system to ensure our product fits our clients computer.

3 Architectural Overview

Now that we have talked about some of the key technologies that are going to be in our software as well as a brief overview of the implementation process we will now talk about the architecture that is involved in the project.

When looking at the overall model of the architecture it can seem complex and intimidating, but Diagram 1.1 helps divide the model into more manageable sections. Our system consists of four main components, some of which can be broken down into our GUI interface (blue), our FTP module (yellow) which consists of an FTP system as well as a file browser, our configuration file generator (purple) containing value checkers as well as formatting and finally the UDP module (green) which has a terminal as well as a UDP listener. These sections make up the solution our team has made to improve upon Dr. Shafer's process.

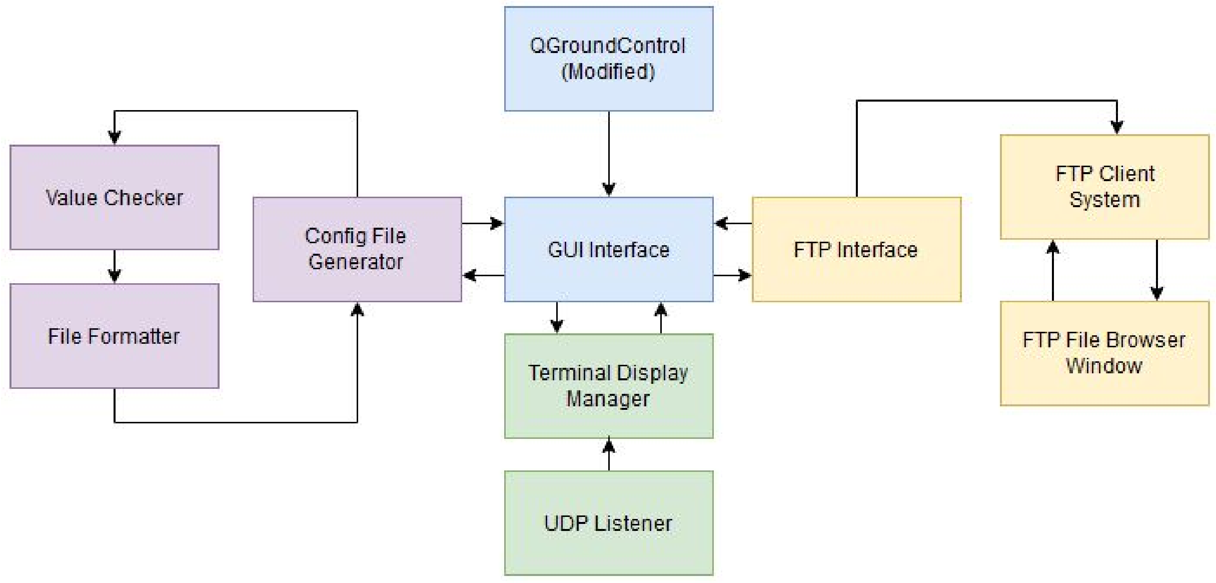

Diagram 2.1 System Architecture Overview

To further clarify our solution we provide the following overviews for the components (as shown in figure 1.1):

3.1 Flight Planner

- ➢ QGroundControl(Modified)
  - ○ This is the keystone component where all of our other components are built into and communicate with. It also provides all of the functionality that QGroundControl already has.
- ➢ GUI Interface
  - ○ This component interfaces with our GUI to receive, display, and send data from the Config file generation system, the UDP heartbeat listener, and the FTP system. The key responsibilities include calling the necessary systems when needed and receiving data and sending it to our GUI. This module manages an interface which is needed for all other modules to function correctly and for our program to display the right information.

3.2 FTP Module

The FTP module contains all of the necessary functions to ensure reliable file transfer between the drone and the computer. There are three main parts to this module which are the interface, the FTP system and finally the browsing window.

- FTP Interface
    - This component provides an interface between the FTP system and the GUI interface. Methods common to both the FTP client and FTP file.

- FTP Client System
    - This component creates, maintains and closes the FTP connection to the drone based on user input received from the FTP interface.

- FTP File Browser Window
    - This component provides a user friendly interface for downloading and uploading files to the drone.

3.3 Configuration File Generator

This module contains functions relating to the creation of configuration files to be sent to the drone via the FTP module.

- Config File Generator
    - This component serves as an interface for the value checker and the file formatting required to generate a valid config file. It also contains the necessary fields to input information to create the configuration file.

- Value Checker
    - This component validates the entered fields making sure the input is within acceptable bounds as defined by our client. It then passes the values if all are verified.

- File Formatter
    - This component takes the verified inputs and formats them into an appropriate config file conforming to our clients pre existing format.

3.4 UDP Module

This module contains all the necessary functions to ensure a stable UDP connection to help receive and display messages coming to and from the drone.

- Terminal Display Manager

    - This component receives data from the UDP listener, processes it into useful messages and sends it to the GUI interface to be displayed.

- ➢ UDP Listener
    - This component listens for the UDP heartbeat messages the drone broadcasts while it is powered on.

In order for our components to perform the behaviors listed above they will need meaningful input generated by the user or the drone. The main communication method we will be using to send this input though the program will be function calls passing the requisite data to the necessary function in the call. All information will flow to and from the GUI interface providing a centralized container which will directly interface with the GUI. In each branch or our program we then send data through a singular point and move the data based on needs and function calls.

When we were designing our program we were heavily influenced by the object oriented programming style. In this style we focused mostly upon the concepts of Modularity and Abstraction. Modularity in this context is the flexibility of our program. We aimed to minimize coupling to allow for an easier to read program. Abstraction in this context focusing on the collecting of common functions for each of our systems into the topmost level possible. An example of the modularity and abstraction in our program is the Config File Generation component which holds common functions for both the Value Checker and the File Formatting. This also is flexible in the sense that the unit could be added to a similar framework and function the same.

4 Module and Interface Descriptions

Our program contains 4 major modules: The Base Module, The FTP Module, The UDP module, and the Config Module. In the section below we will briefly describe what each of the modules do, followed by a UML class diagram and finished off with a description of all the public functions.

4.1 Base Module

The base QGroundControl software provides a large portion of what users need to interact with the drone, but there are some key functions of Dr. Shafer's process that are left out. With this extra tab added to the toolbar, we can house the special modules in a dedicated place.

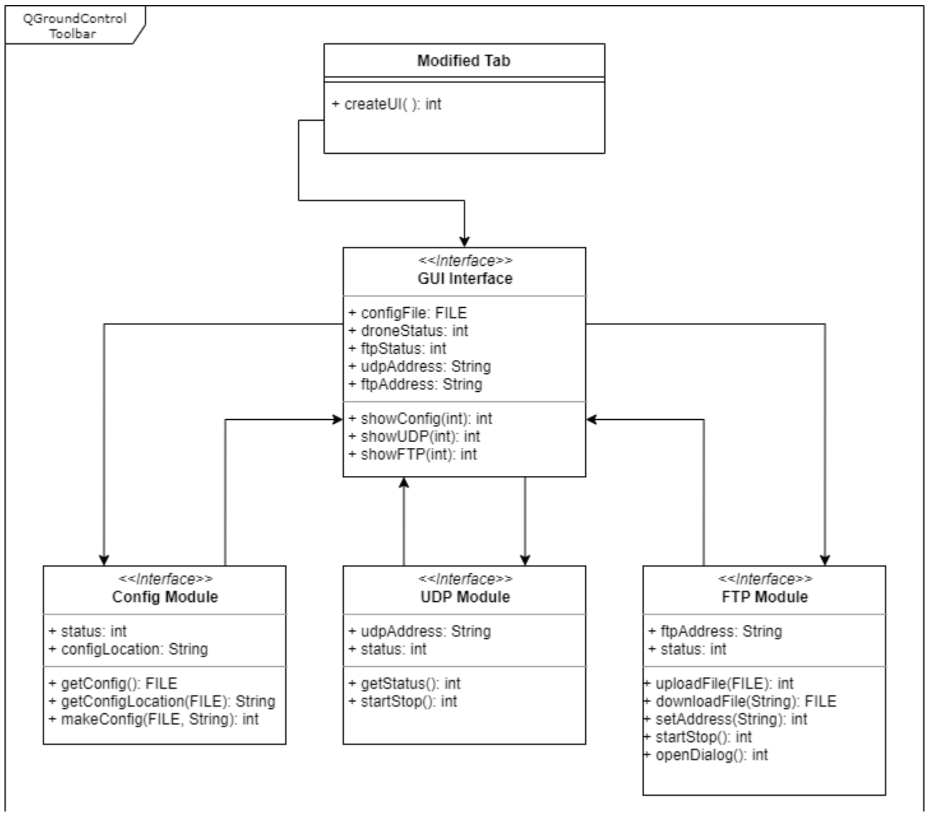

Diagram 2.2 - Base Module Diagram

Modified GUI

- ➢ createUI: This function generates a GUI object where all of our modules are called from. It returns a 0 or a 1 based on success 0 meaning successful and 1 meaning failed.

GUI Interface

- ➢ showConfig: This function signals the config module to start and initialize itself. It returns a 0, 1, or 2 based on success with instantiation error being 1, runtime error being 2, or no error being 0.
- ➢ showUDP:This function signals the UDP module to start and initialize itself. It returns a 0, 1, or 2 based on success with instantiation error being 1, runtime error being 2, or no error being 0.
- ➢ showFTP:This function signals the FTP module to start and initialize itself. It returns a 0, 1, or 2 based on success with instantiation error being 1, runtime error being 2, or no error being 0.

As shown in the diagram above, the base GUI originates from the other tabs of the QGroundControl software. This modified tab has initiator functions to start the interaction between the other modules as well as variables that store pertinent information of current statuses and locations. When looking at the functions there are only initiator functions; this is because all of the other interfaces do all of the “heavy lifting”. These initiator functions have an int return type that provides an indicator if there is either an instantiation error (1), runtime error (2), or no error (0). For showConfig(int), we send a signal to the Config Module to startup and initialize itself (with a command of 1) or stop (with a command of 0). This function displays the UI of the file configuration portion as well as giving the proper file locations of the configs so the FTP Module can send the correct ones. Next, the showUDP(int) function does a similar start command, but it also establishes a heartbeat connection to the drone’s companion computer. This heartbeat is used to determine if the user can actually communicate with the drone. Meaning if an error is produced here, then the other two modules will not function properly and the drone must be fixed or reset. Finally for the functions, showFTP(int); this function is a way for users to send correct config files to the drone as well as receive telemetry data from the drone. It requires a valid config file from the Config Module as well as a healthy heartbeat from the UDP Module in order to do it’s work. The variables listed are important for multiple modules so these act as a holder for them. The configFile variable is unique as it holds the actual file information of the drone’s data collection configuration. This is to be sure that the user can call upon it even if the config module is not working properly or if the drone needs to “reset” the settings due to an error or bug. The two status variables (droneStatus and ftpStatus) keep track of the two connections that the user needs to properly interact with the system. The int variable is used instead of a boolean just in case we find a need to add other error codes instead of having a working (0) or not working (1) status. These statuses will be shown by a red or green indicator next to the relative sub-module’s UI. The last two variables (Strings) pertain to the addresses needed for the user’s interactions. Each one is displayed in the corresponding sub-module UI and is updated if the user needs it. The other interfaces in the diagram were needed to provide context, but they are unique and complex enough to have their own subsection and delve into them more.

4.2 FTP Module

This module creates, manages and closes FTP connections with the drone. These connections are made handshaking using the TCP protocol. This module interacts with the Base Module to receive commands input from the User. This module implements the interface FTPInterface and the classes FTPClientSystem.

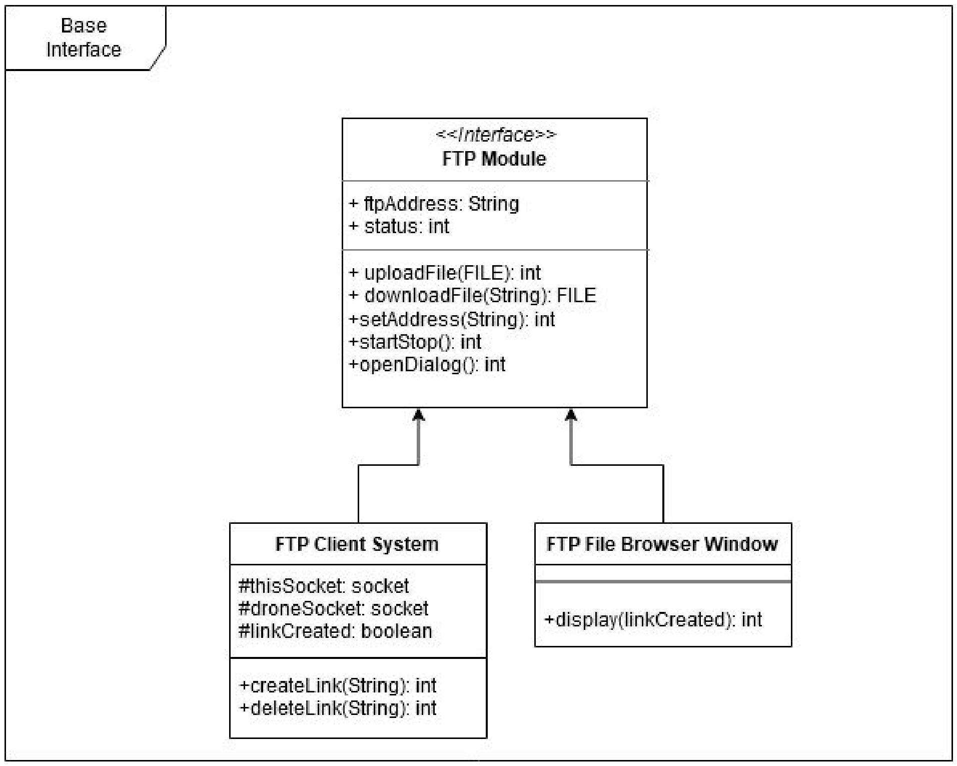

Diagram 2.3 FTP Module Diagram

FTP Module

- ➢ uploadFile: this program takes a FILE object in uploads it to the FTP server. It returns a 0 or a 1 indicating whether the operation was successful with 0 meaning successful and 1 meaning failed.
- ➢ downloadFile: This program takes in a file name as a String and downloads the file if available. On successful operation returns a FILE object
- ➢ setAddress: this function takes in a new address for the FTP server as a String and then updates the ftpAddress field. It then returns a 0 or a 1 based on success 0 meaning successful and 1 meaning failed.
- ➢ startStop: This function starts or stops the FTP server based on its preexisting state turning it on if the server is off and off if the server is active. Returns a 0 or a 1 based on success 0 meaning successful and 1 meaning failed.
- ➢ openDialog: this opens a GUIwhich allows the user to interact with the FTP server. The program returns a 0 or a 1 based on whether or not the GUI was opened 0 meaning successful and 1 meaning failed.

FTP Client System

- ➢ createLink: This function creates a link with the FTP server given the socket of the current device. It then returns a 0 or a 1 based on success 0 meaning successful and 1 meaning failed.

- ➢ deleteLink: This function removes a link with the FTP server given the socket of the current device. It then returns a 0 or a 1 based on success 0 meaning successful and 1 meaning failed.

FTP File Browser Window

- ➢ Display: this function works with open dialog to display the GUI for interacting with the file system to the user. It then returns a 0 or a 1 based on success 0 meaning successful and 1 meaning failed.

As shown in diagram 2.3 above the FTP module is built within the confines of our base module. The FTP module provides a connection between the base and our FTP system. The ftpAddress field stores a user defined variable of where the FTP server is located. We use the status variable to tell us the current status of the FTP server at any given time. The uploadFile and downloadFile functions allow us to manage files on the FTP server. setAddress defines a way to change the ftpAddress so it can fit the user's needs. startStop is a function for starting and stopping the FTP integration in order to save resources. openDialog works with the function in the FTP File Browser Window: display to create and show an accessible GUI to the user. In the FTP Client System we use thisSocket and droneSocket to help us with the startStop function from the interface. The createLink and deleteLink functions are helper functions for the StartStop function as well.

4.3 UDP Module

Flight planners such as QGroundControl need a way to tell if the drone is "healthy" and working by looking for a drone's "heartbeat" by UDP connection. Users will need this too to determine whether they are able to receive the telemetry data from the VHF tags. In case of a crash or bug, users will be able to start and stop data collection by starting and stopping the "heartbeat"; this will only be for the user, QGroundControl will still be able to receive messages.

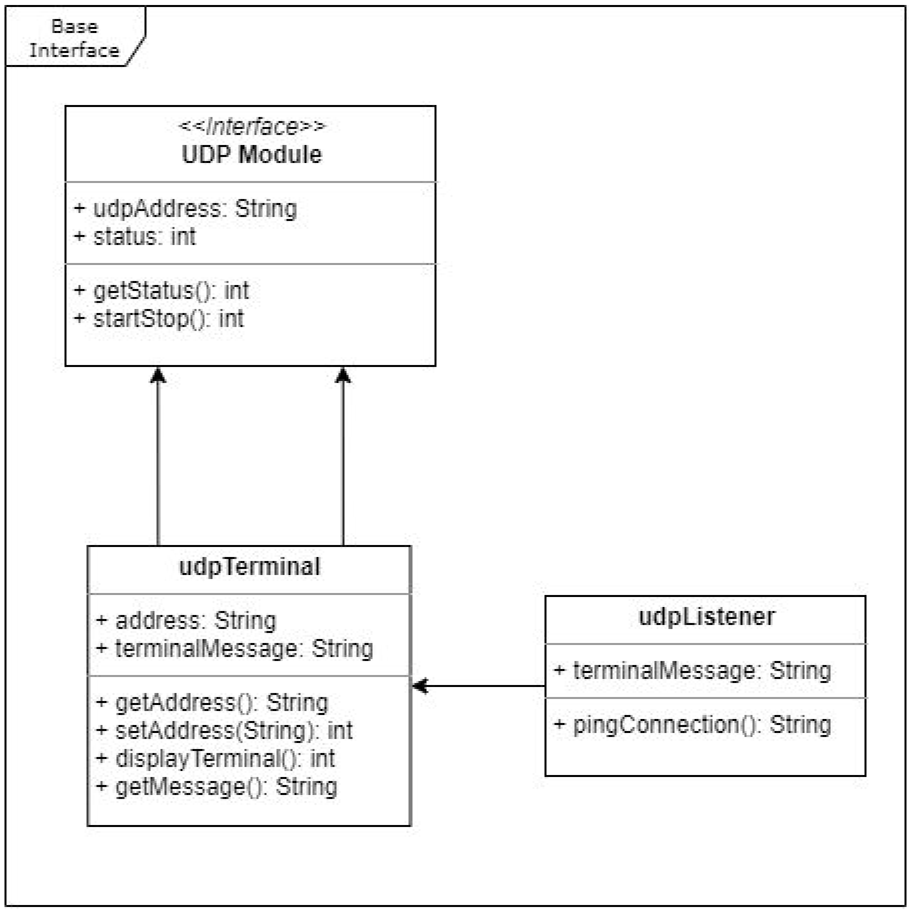

Diagram 2.3 - UDP Module Diagram

UDP Module

- ➢ getStatus: This function gets the status of the UDP connection. It then returns a 0 or a 1 based on success 0 meaning successful and 1 meaning failed.
- ➢ startStop: This function starts or stops the UDP listener based on its preexisting state turning it on if the listener is off and off if the listener is active. Returns a 0 or a 1 based on success 0 meaning successful and 1 meaning failed.

UDP Terminal

- ➢ getAddress: This function returns the address field
- ➢ setAddress: This function sets the address field. It returns a 0 or a 1 based on success 0 meaning successful and 1 meaning failed.
- ➢ displayTerminal: This function displays the terminal. It returns a 0 or a 1 based on whether the terminal is displayed 0 meaning successful and 1 meaning failed.
- ➢ getMessage: This function listens for a message and returns a message as a String upon receiving a message.

UDP Listener

- ➢ pingConnection: This function pings the Drone to see if it still is in range. It returns a String, if the ping is returned the String will not be empty but it will be empty if the ping times out

Looking at diagram 2.3, we can see that the Base Interface builds the UDP Module inside of it as it is one of the parts of our whole modification. The UDP Module Interface provides intermediate connection between the Base and the “heartbeat” controls. In the interface two variables are held as they are needed in the Base and UDP Modules, the UDP connection address (String). The address is a string because it will hold an ip of the UDP and it may change during runtime so it is not hard coded in. The two functions that the interface holds are getStatus and startStop which communicate with the udpTerminal class. In getStatus there is an int return type that shows if the “heartbeat” is healthy or not (1 or 0 respectively). This is done by asking the udpTerminal to translate the health of the last known message from a string into an int. The startStop function toggles the user’s terminal status to receiving or not (1 or 0). This is done by telling the udpTerminal object to switch a private int to either 1 or 0 and returns what state it is in after the switch. In order to keep a dialog with the connection, the classes udpTerminal and udpListener are needed. UdpTerminal communicates between the interface and the udpListener. The terminal contains two variables, the UDP address (String) and the terminalMessage (String). The address sill must be a string for simplicity sakes because that is how the interfaces hold it. The most recent “heartbeat” message from the drone is kept as a string in terminalMessage so it can be accessed for display on the UI. The getter and setter functions for the UDP address (getAddress and setAddress) are to be used by the UDP UI so then the user can change the address when needed. For getAddress(), the udpTerminal object will communicate with the udpListener object and ask for the current address that it is using in a String format. The setAddress function has a return type of int to let us know if the address was successfully changed or not (1 or 0 respectively). The functions displayTerminal and getMessage pertain to what the user will see on the UI. The displayTerminal function returns an int of the drone connection’s health, 1 is healthy and 0 is not. We find the health status by parsing the terminal message provided by getMessage. The function getMessage has a return type of String as it changes the local variable terminalMessage. In order to receive these statuses and messages, there needs to be a class that constantly pings the ether for a UDP response. That is where udpListener comes in; this listener uses the address provided by the user to try and receive a valid message from it. The pingConnection function does just that by taking the address as input, and returning a message as a string. The message does not get parsed here but is sent back to the udpTerminal class for processing. In case of a crash or bug, the most recently received message is stored here for redundancy. The UDP Module is quite important as it is how the user can make sure they are collecting data and not wasting a flight because of a connection error.

4.4 Config Module

The user needs a way of telling the drone the data collection settings required for each flight, that's where the Config Module comes into play. The Config Module takes in user input, double checks it for any errors, and spits out a config file if there isn't anything wrong with it. We use this instead of a hard coded solution because users might want to change the frequency or collection settings due to hardware constraints.

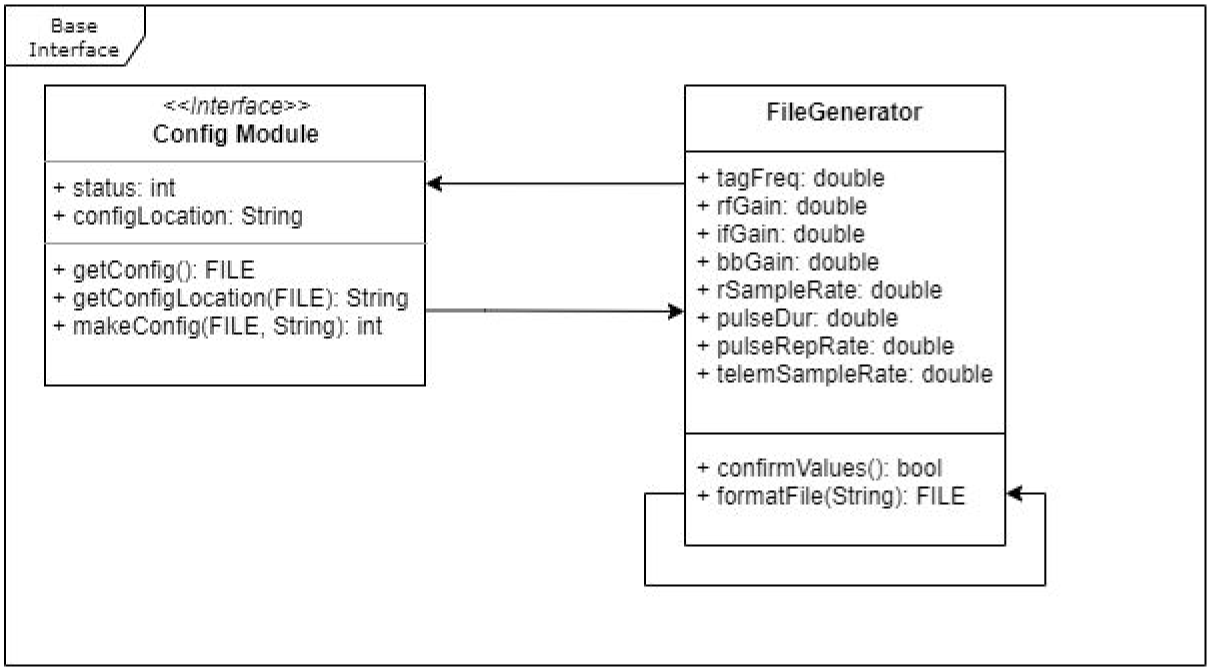

Diagram 2.4 Config Module Diagram

Config Modules

- ➢ getConfig: This function gets the config file and returns it as an object of type FILE.
- ➢ getConfigLocation: Given a FILE object the function gets the location of the config file and returns the path as a string
- ➢ makeConfig: given a FILE object and an address as a String the function then creates the appropriate file and stores it at the path location

File Generator

- ➢ confirmValues: This function verifies each configuration value entered. It returns a True of False based on whether the inputs are valid
- ➢ formatFile: this File takes in a String with inputs listed in it and formats the string into the preferred file. It then returns the formatted file as a FILE object.

In diagram 2.4 we are shown how this module is implemented as with the other modules this is made in the base interface. The Config Module interface provides a connection between the base module and our Config System. The fields for the Config module interface status and configLocation both store important information. The status variable holds a value either 1 or 0 which indicates if the system is running. The config location is where the config file is saved upon validation. The methods of Config Module provide useful functions starting with getConfig which allows for us to modify a pre existing config file by locating it and returning it so we have access to it. The getConfigLocation allows us to determine what value configLocation should hold. The makeConfigFile function serves as a function that runs the FileGenerator functions required to make a config file. For the FileGenerator class we have many fields which store important data. The tagFreq, rfGain, ifGain, bbGain, rSampleRate, pulseDur, pulseRepRate, and telemSampleRate fields store config data entered by the user. These data points are needed to ensure that the user will be collecting viable telemetry data from the VHF tags on the animals. Without these parameters, the drone would not know what signals to look for. The confirmValues function makes sure all of the fields listed above are within the correct value ranges for each value. The formatFile takes in a String with a specific format and formats the string into a finished config file ready to be uploaded to the drone.

5 Implementation Plan

Now that we have talked about the modules, we will go into our implementation plan to help show how long we believe implementation will take as well as who will be working on which part.

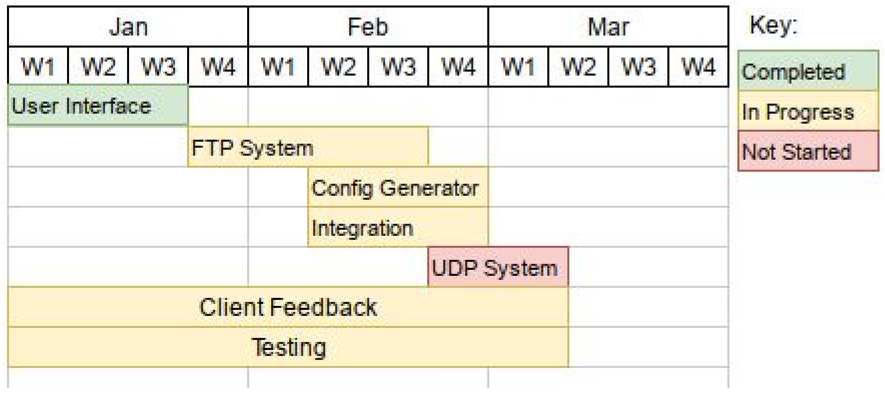

Figure 3.1 - Gantt Chart of Schedule

One module that we have already completed is the base module, which involves creating the GUI of the program which was developed for our technical demo last semester. Our future implementation plans for our project is split into three main phases: the first is getting the FTP setup, followed by the configuration file generator and then finished with the development of the UDP system. The first major development phase is getting a working FTP system up and running. There is a fair amount of time dedicated solely to the FTP system because we believe that this is something that everyone should be working on at once to ensure it gets completed and is working well fairly quickly. After developing the basics of an FTP system, we will then begin on implementing the configuration generator which Sam and Eric will be in charge of. While development with this is going on, we will also be spending resources making sure that this does not cause a problem with the previously developed system and to ensure that they both continue to work, this will be done by Keller and Nathaniel to ensure that the FTP system still functions. After ensuring there are no integration problems with the in-progress configuration generator and the FTP system, Nathaniel and Keller with them begin developing the UDP system as as the configuration generator is being wrapped up. We will then continue with the UDP system while still ensuring we do not cause any problems with the two other components. Our client has stressed that this program has to work, therefore we will constantly be testing for bugs and other problems throughout the entirety of the development cycle. We will also be meeting weekly with our client during this development phase to get his feedback as things are developed. There is a chance that during this phase he may see something he will want tweaked or changed so there is a chance that this development plan changes as we progress further along it but we do not believe that any big changes will be added further down the road.

6 Conclusion

In conclusion, the positional data obtained by scanning for VHF tags can massively inform researchers about the behavioral patterns of animals around the world. However, as mentioned before, the traditional method of hand-held receivers can be incredibly time consuming as well as occasionally dangerous. Using a drone to collect this data solves these problems, but our client's current implementation of this solution can still be improved. Dr. Shafer still has to use two different programs for his pre-flight setup: the flight planner QGroundControl and a custom MATLAB data collection configurer. The MATLAB program is very slow to start and doesn't need to run in the MATLAB runtime environment. By porting his MATLAB program's collection configuration functionality to the drone's flight planning software, we can save him and his colleagues hours upon hours of time when collecting this critical wildlife data for ecologists and biologists alike. Our proposed solution is to develop a singular application built off the back of QGroundControl. This software will have all the flight planning capabilities as well as now contain preprocessing tools such as configuration file generation and FTP capabilities to send this information to the drone out in the field. Currently, we have completed a version of the GUI that we are happy with and now are working on implementing an FTP system. Once we believe we have a working FTP system, we will then swap to ensure we can create a configuration file from within the program and then we will work on creating the UDP module. In the end we are happy with our current progress and are confident we can provide a good product to our client that will greatly help him with his research.