RAMPART

Remote Aerial Mission Planning and Radio Tracker

Requirements Specification Document Version 2.0

12/02/19

Team Members

Eric Gault

Samuel Gilb

Keller Mikkelson

Nathaniel Zeleny

Team Sponsor: Dr. Michael Shafer

Team Mentor: Mahsa Keshavarz

Accepted as baseline requirements for this project: Client:_____

Team:_____

1 Table of Contents

2 Introduction

Wildlife radio telemetry is a very important tool for tracking the movement and behaviour of virtually any animal attached with a transmitter. The reason why scientists want to track these animals is to gather behavioural and movement data of any animal attached with a transmitter. There are two main methods to tracking animals in this way, GPS tracking as well as very high frequency tracking. But 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 more quickly gather transmitted data to 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 cost 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 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 problems with this workflow that we would like to solve:

- ➢ Having to use two separate applications requires constant swapping to ensure maximum benefit from both pieces of software.
- ➢ The custom application developed by Dr. Shafer has a very slow launch time and can take up to five minutes to initially load.
- ➢ 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 this involves taking all the preflight software and moving it into the flight planner. This will take care of startup times and reduce required startup programs to one. This requirements document will provide specifics about our planned solution, the constraints of the solution, as well as our development schedule for the next year.

3 Problem Statement

To collect the positional data of tagged animals, our client follows a specific workflow:

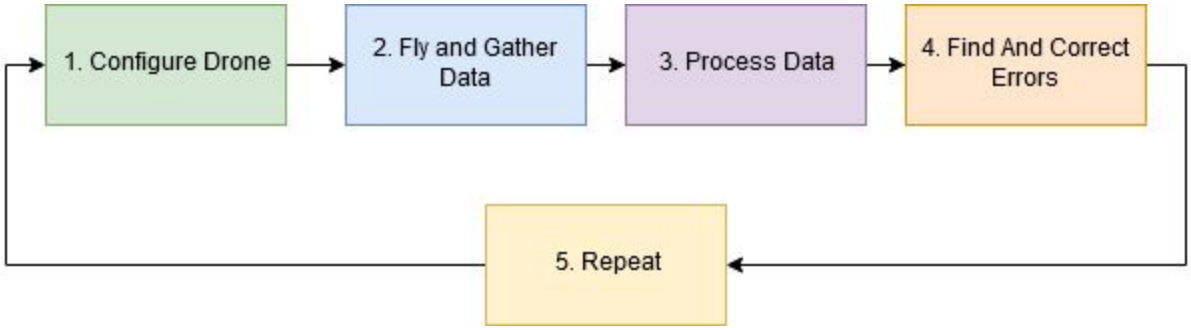

Figure 1: Client workflow

1. He first plots his flight plan in his flight planning software and uploads the configuration data to the drone from a separate MATLAB extension.
2. He then has the drone fly the route and collect data on the VHF tags which he either has scattered along the flight path, or are already attached to the animals being tracked.
3. He then returns to his office where he downloads and processes the flight data using the MATLAB extension.

4. Once the data has been appropriately processed he carefully examines the data and the program for errors. If an error is found he will make measures to remedy it.
5. After making any needed changes, he will then repeat this cycle which is on average repeated once a week.

The main issues in our client's workflow are as follows:

- MATLAB must be opened to fly the drone because the data configuration is only handled in MATLAB currently.

  The MATLAB software can take upwards of five to ten minutes to launch. This ends up being incredibly costly as if you add up the amount of time it takes for MATLAB to open each year, it takes around the same amount of time as 52 extra flights per person.

- QGroundControl cannot currently send the additional radio configuration data to the drone (It controls the flight path but does not have a way to talk to the radio antenna used to pick up VHF).

  Because QGroundControl does not have an option to send specific configuration data to the drone, this required our client to develop his own application so that he could accomplish this. This now means that our client has to swap back and forth between two different applications while he is in the field.

- QGroundControl does not display the status of the drone mid flight.

  Since QGroundControl does not have any ability to access information relating to the VHF receiver, our client would not know if any error with data collection occurs mid flight. He had to build this in his own software, but now he will need to swap between QGroundControl, which shows where the drone is along its path as well as his own software to ensure that onboard functions are working correctly.

4 Solution Vision

With the problem provided, our team needed to find a way to reduce the amount of time of our client's setup and consolidate the pre-flight steps into one program. The solution we are planning, as seen in Figure 2 below, involves remaking the configuration software of our client's MATLAB code in QGroundControl so that our client's total setup time will be significantly less. The figure also shows that it will also be able to send all needed configuration data through one program instead of two, introducing more ease of use. Our solution will also address QGroundControl's nonexistent status message display by using the modified GUI to display these messages.

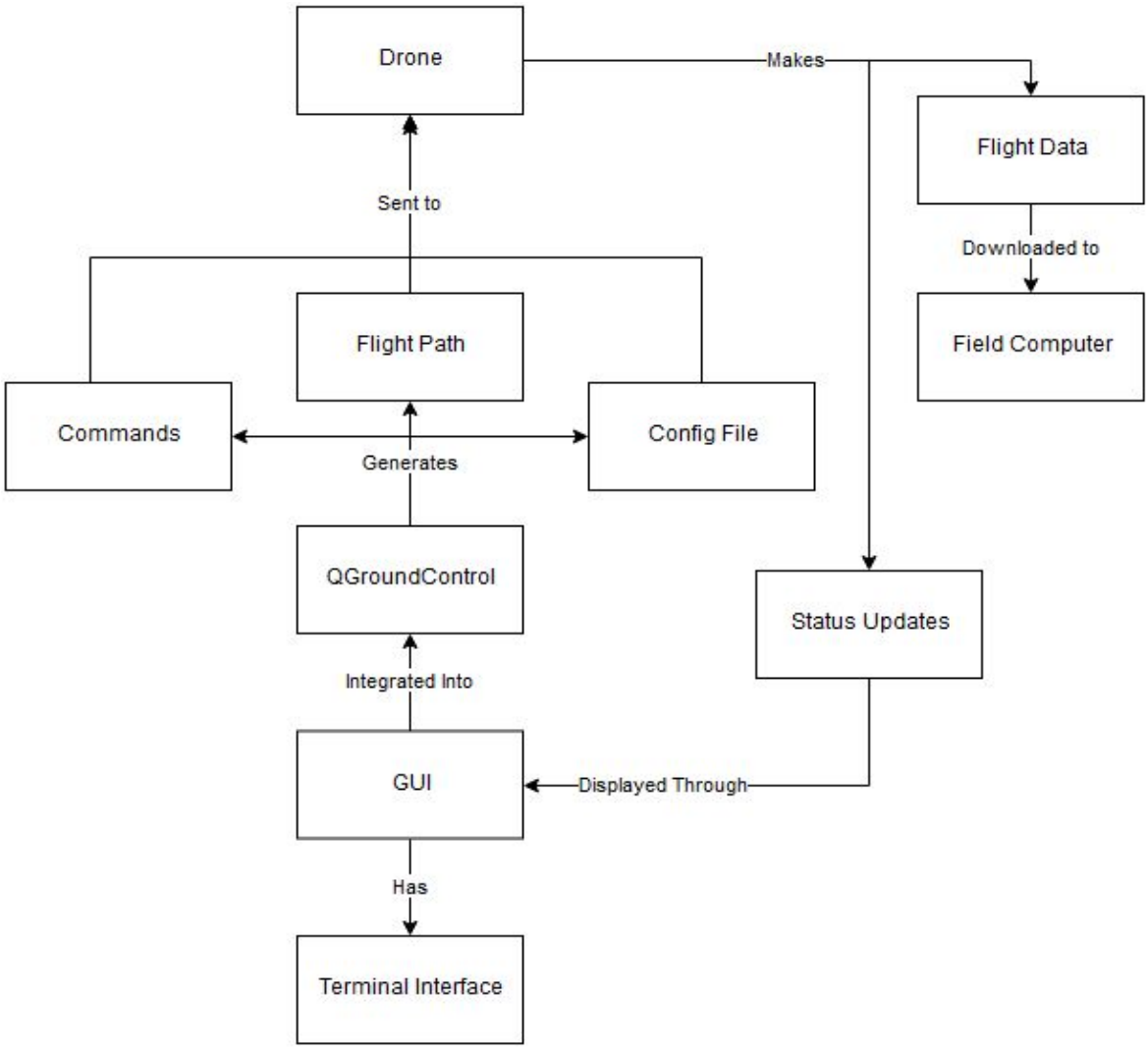

Figure 2 - Solution Overview

Our Solution will provide these key features:

- ➢ Complete configuration management in QGroundControl.
    - This is displayed in figure two where QGroundControl Generates a config file and sends it to the drone.
- ➢ No MATLAB required for drone flight planning.
- ➢ Terminal display to show drone's status messages.
    - This is represented in figure two through the status messages the drone makes and sends back to the GUI which then displays the message through the terminal interface.
- ➢ Ability to communicate with the drone to upload configuration files and download flight data.
    - As shown in figure two, QGroundControl generates the config file and the config file is sent to the drone. The flight data is created by the drone and then downloaded to the field computer for processing.
- ➢ A Graphical User Interface (GUI) to manage the creation of configuration files and how status messages are displayed.
    - As shown in figure 2 the GUI is integrated into QGroundControl and will control both how configuration files are made and how the terminal interface appears.

Our system will use data from the drone mostly flight data and status messages to let the client know the status of the drone and provide useful results through the flight data. It will also use data from the user such as the manually input configuration data, start/stop commands, and a manually inputted flight path. This allows our client to be able to use QGroundControl for setting up the drone for data collection. Our system will generate a configuration file based on the user input. This file is used to control the functionality of the VHF antenna located on the drone. With the implementation of these functions, our client will no longer need to use MATLAB in addition to his flight planning software for pre-processing data. This in turn means that he will only need to use QGroundControl when out in the field as it will now possesses all the functionality of the MATLAB code as well as some quality of life changes to QGroundControl.

5 Use Cases

To better explain our solution vision we have created some use cases. These will explain the currently expected functionality of our product in the context of a user.

Use Case: Create a configuration File.
Primary Actor: User.
Brief: The client will be able to input the correct data into fields and be able to create a configuration file from this data.
Postcondition: Successfully created configuration file.
Precondition: Empty form template for configuration data.
Basic Flow:

1. Client inputs data into the numerous fields.
2. Client inputs a name for this configuration data.
3. Client saves the configuration file.

Use Case: Upload configuration File.
Primary Actor: User.
Brief: The client will be able to send the configuration file from the computer to the drone.
Postcondition: Successfully uploaded configuration file to the drone.
Precondition: Different or no configuration file on the drone.
Basic Flow:

1. Client inputs data into the numerous fields.
    a. Client may choose to import configuration file instead.
2. Client names the configuration file if needed.
3. Client uploads the file to the drone.

Use Case: Start Drone Radio.
Primary Actor: User.
Brief: The client will be able to start the collection of data from the drone.
Postcondition: Drone radio is now on.
Precondition: Drone radio is not on.
Basic Flow:

1. Client clicks the button to start collection of data.

Alternate Flow:

1. Client clicks button to stop the collection of data.

a. Requires the drone to currently be collecting data.

6 Project Requirements

Now that we have briefly listed out what we are aiming for in our solution, as well as mentioning a few user stories to give a better understanding on what our client wants, we will now be going into more detail on the specific requirements our client wants for this project and how these requirements can break down into smaller parts.

For overall domain level requirements, our software needs to be able to be the following:

- A modified version of an open source flight planning software that implements the GUI and functionality of our sponsors MATLAB program.
- A robust and bug free software that is highly reliable and easy to reason about.

Although we do not have many domain level requirements, we can break these down into smaller subsections which we will go into in the next sections of this document. Our client has also stressed numerous times that our final product has to work. Due to the importance to our client, and therefore us, we believe that ensuring our product works well and will not have any problems is a high-level requirement, we can later split this into more lower level and specific requirements later in this document. As of now the following is the current listing of requirements for this project. But we do understand that there is a chance that things may change down the line as development starts.

6.1 Functional Requirements

Our client has been very clear on what our new piece of software needs to be able to do. As mentioned in the previous section, we noted that we need to implement the GUI and functionality of our sponsor's MATLAB code into an open source flight planning software. In our Technical Feasibility document, we discussed the pro's and con's of different flight planning software as well as the mentality that lead us to choose QGroundControl as our base for this project. We can now start from the top-level requirement of creating a modified version of an open source flight planning software that implements the GUI and functionality of our sponsors MATLAB program and break it down into more steps.

➢ Ability to delete files:

One of the main reasons that our client has to use his own software is because there is currently no system setup in his flight planner to be able to create or edit the configuration file for VHF tracking on the drone. Therefore we need to implement this and we can split it up into smaller subsections.

➢ Configuration file generator:
    - GUI Form to create the configuration file.

The first is that there needs to be some sort of GUI form that he can input the values he needs to create the configuration file. This is straightforward as all that is needed in this part is to ensure that we are using the correct fields as well as implementing them in the same order to not mess with the workflow he has been using in conjunction with his other application.

➢ Be able to import previously created files:
    - Be able to choose which file to import from a form of list.
➢ Save configuration files to computer for reuse:
    - Ability to name these saved files.

Our client also wants to have a system setup so that he can reuse previously created configuration files. He would like this paired up with a way to choose which configuration file to import, this means that we will also have to store names for configuration files so that the client can choose which to import. Because of this we will also need a way to save the configuration file within the system with also being able to name this file for ease of use. Our client wants to be able to choose which configuration file to import so we will have to create a way to see all configuration files saved on the device in a specific location so whichever file is wanted can be chosen.

➢ Show FTP system connection status:

He also wants a visual indicator to ensure that the FTP system that is being used for file upload and download is connected or not. This is to ensure that he is not trying to transfer files when he isn't connected and he will be aware that he needs to reconnect to the server.

➢ Heartbeat Terminal:
    - Ability to receive status messages from the drone.
    - Logging of messages received and sent.

The next high-level requirement that our client wants within our GUI is a heartbeat terminal. With this terminal he wants to be able to see incoming status messages from the drone while it is in-flight. It is vital that this terminal shows the correct information and does not miss anything. We also need to make sure that any messages that we send from the ground to the drone are also recorded in this terminal. This part is incredibly important to our client as if we do not ensure that the message was received successfully this can lead to a lot of time wasted on behalf of our client.

➢ Display UDP connection status:

Our client also wants to have a signal indicator indicating the current status of the UDP connection being used for the terminal information. This is to be used alongside terminal updates to check the status to see if the drone is connected or not.

➢ Ability to start and stop the radio for VHF tracking:

It is also necessary to send radio messages from our flight planning software to the drone's radio system to start and stop the collection of data. With this terminal we also need to have buttons that tell the receiver on the drone to start or stop receiving data. This is to ensure that the drone will only be collected data when it is needed. It is also needed to convey the starting or stopping within the terminal as well as ensure that these messages are fully sent.

➢ Reliability:
    - Ensure messages sent to the drone are received correctly.
    - Ensure messages sent from the drone are handled correctly.

Due to the possibility of any message not being properly sent, we need to put in measures to ensure that the drone receives the correct messages and acts accordingly. If any message sent or received is incorrectly handled, this will cost our client time and money. Therefore, we need to ensure that anything we send to the drone is sending the

same information that the user wants to send. We also need to make sure that any information that is received from the drone is handled the correct way. If there is ever an error when uploading or downloading files with the drone, our client wants to have some sort of notification so that he is aware if a problem occurred so he can repeat his last action. Our client has mentioned that this software must work and that he needs to be able to fully rely on this product. Therefore we must take all possible measures within the code we write to give him the reliability.

- File Transfer Protocol system
    - Ability to send and receive files from the drone
    - In a tree format

Our client has an FTP system setup on the drone to allow for sending and receiving data. Therefore, we will need to implement this system into our modified program. Our client must send configuration files to the drone to ensure proper data receiving as well as downloading all the information after the flight has been completed. With the FTP system setup with the drone, our client has a preference on this being shown in a tree format.

6.2 Non-Functional (Environmental) Requirements

The functional requirements that create the shape and body of our project are important to get started with the GUI, heartbeat terminal, and FTP connection design, but there needs to be requirements that allow us to verify that the system is being used properly and performing as expected. These requirements are the environmental requirements that have measurable or quantifiable outcomes based on the system the team has created. This currently this ranges from the performance time of an algorithm in the code to if a user can properly operate the system with or without outside help.

The addition to QGroundControl (QGC) should not increase the software's startup time by 40%. In order to keep the new startup time to fly the drone below the time of 5 to 10 seconds, all the modifications to QGC cannot exceed the startup time past 14 seconds. This means loading the QGC flight planner and opening the GUI widget we have created must be able to load in that 14 second time frame. This is taking into account

that the GUI will load on its own and not require user input to load the widget every time they want to fly the drone. In the initial installation and setup of our widget will take longer than this 14 second time restraint but, it only needs to be done once. The time it takes will be ignored in testing because it does not represent use in the field.

Since the software is open source, someone that is not a developer on the project may want to change or modify our system's code. Our documentation in the code should be easily traversable and understandable for anyone interested in modifying it. Easy traversion and understanding of our code can be constituted as having a person who has little experience with software development be able to restate a code block's specific purpose by code comments. The person should not have to read any actual code to understand the basic function, but just the corresponding comments. The important part of the comment for this is the plain English description of the algorithm that the person must read. If they cannot give an explanation of the code based off of the comment, then it must be revised and rewritten.

In correspondence to documentation in the code, a person who wishes to use (but not modify) our modification to QGC should be able to use it by reading the documentation that will be provided on the Dynamic and Active Systems Lab (DASL) website. The minimum experience required for complete comprehension is a novice user of QGC so the guide does not have to start from the very beginning. When someone reads our out-of-code documentation, they should be able to set up and run the drone on default settings for gathering the VHF tag telemetry data. The setup will include downloading, installing, and verifying the QGC program with our addition can be used without outside help.

After the user has downloaded and correctly setup the QGC flight planner and our modification, the user required functions of the modification needs to be able to be understood by a user with at least a novice understanding of QGC. When the user opens our GUI, they must be able to correctly identify all user inputs that they can utilize. This includes text field inputs, start/stop buttons, drop down menus, and import/export data buttons. For complete understanding to be validated, a novice user should be able to send configured collection settings, upload the settings to the drone,

start/stop data collection, and download the VHF tag telemetry data with the out-of-code documentation as mentioned previously while an experienced user should not have to use the out-of-code documentation. This requirement does not use the ability to fly or plot a drone flight plan as a validator because those are not in the scope of the modified GUI.

Another requirement based off user comprehension is that a user needs to know exactly what data they are downloading from the companion computer and where it is being downloaded to. To validate this a user must be able to locate where the data has been saved, open and identify the data being telemetry records of while the system was running. A novice user of the system must pass this while being able to look at documentation and similar telemetry data examples while and experienced system user must be able to do this without any outside help.

6.3 Environmental Requirements

With the functional and non-functional requirements established, this document will now discuss the environmental requirements that this project contains. Our client has a very specific system, most of which cannot be immediately changed without radically changing the hardware. This means that we are restricted not only by certain requirements established by the client, but also the physical system that this project revolves around.

That being said, the first, and one of the most important environmental requirements is the fact that we must design our program to work specifically with the PixHawk 4 flight controller. This is essentially a small computer that enables the drone to fly around a set path via an auto-pilot. While there are many different variations on this device, this is the one our client chose specifically, and changing said hardware would require additional development time as well as additional funding.

Since we are required to use the PixHawk 4 flight controller, we are also restrained by the fact that this device is only compatible with certain pieces of flight planning software. As discussed in our technical feasibility document, we have chosen a program called

QGroundControl to be the flight planning software we will be modifying to implement our solution. As such, we will be restrained by the frameworks implemented by this flight planning software. To be specific, QGroundControl uses the QML modelling language for structuring its user interface and the C++ programming language to implement its programmatic functionality. This means that while we are working with QGroundControl, we will be restrained to using these two languages in order to craft our ultimate solution.

We also had some environmental requirements requested directly by our client. Since changing code within the drone's computer itself is out of the scope of this project, we must treat the drone as a "black box" of sorts. As such, any configuration files we generate and send up to the drone's computer must contain all of the same information and formatting as our client's current implementation. Similarly, our client has requested that the interface used for downloading the data from the drone must be similar to the interface he is currently using. Specifically, the client wants to be able to traverse the files within the drone's computer using an explorable file tree structure which allows the user to download whatever file they have highlighted.

The final environmental variable we had to consider was the fact that our client wants our work to be added to his own code repository. This means that our in-code[1] and out-of-code documentation[2] must adhere to the standards used by the DASL. An example of the in-code documentation is shown in Figure 3 on page 15. These are comments done in MATLAB, so the exact format will be slightly different but the explanations should be almost the same. A comment should include: the function name (line 3), current version (line 4), description of what the function does (lines 7-13), authors (line 16), input descriptions(lines 19-29), and output descriptions (lines 32-onward).

While this mostly boils down to matching the formatting of their documentation and coding practices, it is important that we begin development with this in mind from the start. This is because going through and changing the formatting of our code and documentation after the fact would be an unnecessary time sink. On top of this, submitting code that does not adhere to these standards would make our program look out of place, making his code repository look less professional.

```
function [] = uavrt_main()

%% uavrt_main
% Version 1.0
% 2019-02-19
%
%UAVRT_MAIN processes flight and radio data from the UAV-RT system and
%displays results in a Matlab figure window, a .kml file for GoogleEarth,
%and a text output file. The program assumes that flight logs and radio
%data begin at the same time. Note that in the current system their
%timestamps in the filenames may look different, but these are the times
%when the files are initialized, not when the recordings starts. The
%writing to those files does begin at the same time.


%Author: Michael Shafer
%%*************************************************************************

%%Input parameter:
%flt_data_flnm          Location of the flight data record
%sdr_data_flnm          Location of the IQ radio data record
%v_thresh_at_wypt       Velocity below which the vehicle should be at a
%                       waypoint
%Fs                     Sample rate of the IQ data
%pulse_dur              Expected pulse duration (s)
%pulse_rep              Expected pulse repetition rate (s)
%filt_band              Width of filter (Hz)
%savestring             a char array that will begin each output filename
%out_data_path       place where you want the results saved.


%OUTPUTS
```

Figure 3 - In code Documentation Example

7 Potential Risks

When assessing the requirements and production of a system, recognizing and prioritizing the potential flaws can help make preemptive safeguards during production instead of after. Risks can appear in three different categories: Technical, Social, and Market. Technical risks are problems that can arise due to poor implementation of a system and are found by testing and “walkthroughs” of a software’s logic. Social and Market risks are usually an outside body’s reaction to the software and its purpose,

whether it is legal action or community action they can still affect the software development. These are mitigated by researching and evaluating the opinions and rules set in place for similar systems.

Much like any project, there are a number of potential risks that we can run into during the development of our project. The next few sections will talk about the risks that we can currently see as possible problems in the future and how we will try to ensure these problems do not happen, and if they do, how we can fix it.

7.1 Technical Risks

There are three different technical risks we need to watch out for. Physical drone crashes are an issue can happen, the probability of it happening is extremely low because even though we are not changing any flight pathing directly, our edits could inadvertently create an issue. Even with the low likelihood, the setback can be very long and expensive, which makes the concern high. In the event of a crash, there will be records displayed in the GUI's terminal in order to help diagnose what went wrong and prevent future crashes.

Issues that could happen more frequently, but would have less repercussions are the data not being downloaded wirelessly and incorrect data collection. There is a possibility that there could be connection issues to the client’s laptop and the offload of data from the drone cannot be done wirelessly. Since our team is trying to reduce the amount of time needed, the simple solution is to perform a wired download from the drone to the laptop. Such a quick and simple fix makes this a low concern.

Another issue that arises from data collection is that the data from the VHF tags could be incorrect. This would typically be due to user error in that the user could accidentally input the wrong collection settings when configuring the drone, which could force the user to re-run the drone's flight path. A way to combat this is to have a confirmation of the collection settings which indicates the settings stored on the drone's computer. The prompt will also provide an option to resend the configuration file if the user finds that

the settings are incorrect. Since this relies on the user to pay attention to what they have entered, we feel that this has a medium chance of happening.

7.2 Social Risk

While our group does not have any issues at this time, there is a low possibility that could change. For every large decision, we have a discussion on what needs to be done and split it up in a way so that everyone believes that the work is even. For any disagreements, the team will follow a majority-rules mindset. If the team is split on a decision, we will go to our mentor for help on making a decision.

7.3 Market Risks

If this software modification comes to market, there could be possible risks in competition or being made futile by integration from QGroundControl. While QGroundControl is an open source software, the licensing they use does not require developers to relinquish copyright of contributions if they choose not to. Because of this, the likelihood of this happening is fairly low.

Regarding competition, there is an Australian company called Wildlife Drones that uses UAVs to track animals with VHF tags. Since this is a fairly niche field, they are taking some potential clientele from the DASL making this a medium concern. The likelihood of Wildlife Drones overtaking our market is low because they only rent out the drones and the DASL has a utility patent on the UAV system, meaning Wildlife Drones cannot legally bring their business to the states.

7.4 All Risks

Table 1 - All Risks Assessment

Risks	Concern Level	Likelihood	Plan
Tech: Physical crashes	High	Low	Create crash report
Tech: Data not downloaded wirelessly	Low	Medium	Download through wired connection
Tech: Data is incorrect	Medium	Medium	Confirm connection
Social: Group disagreements	High	Low	● Majority rules ● Confer with Mahsa
Market: Obsolete through integration	High	Low	● Licensing ● Keep IP of project
Market: Wildlife Drones	Medium	Low	U.S. utility patent

In order to properly plan for the development path in a whole, our team found seven different risks that could potentially hinder development and deployment of our project. Three involved technical risks within the software itself: Physical drone crashes with a high concern and low likelihood, wireless connection failure with a low concern and medium likelihood, and finally incorrect data collection with a medium concern and likelihood. One risk was a social risk where disagreements between the group can stall development; we assessed that with a high concern but a low likelihood because of communication redundancy. The last two were market risks: Being made obsolete by integration with QGroundControl and an Australian company, Wildlife Drones, taking potential clients from the DASL. QGroundControl is a high concern with a low likelihood because of their licensing agreements and Wildlife Drones is a medium concern with a low likelihood due to location and the DASL having a U.S. utility patent.

8 Project Plan

Currently, we are starting to work on our demo by implementing changes into the GUI of our chosen flight planner, QGroundControl, so that we can get feedback on the layout to ensure it fits our client's needs. Some milestones to point out and expand upon in Figure 4, are: Finished GUI, Config Generation, FTP Connection, and Quality Assurance. The finished GUI can be split up into two sub-milestones. First, the completed skeleton for the GUI must be placed in QGroundControl and approved by our client. Second, the code stubs and comments will be used to bridge the gap between the configuration and FTP milestones. The configuration generation milestone is adding the functional requirements pertaining to configuration inside the skeleton GUI we previously created. This involves the inputting, loading, and saving of user configuration settings for data collection. Connecting to the FTP is another milestone that can be expanded upon into two sub-milestones: terminal heartbeat and wireless data transfer. Terminal heartbeats are already in the background of QGroundControl, so we will need to have a live update of the heartbeat on the GUI by calling on that background process. Wireless data transfer will be finalized when the drone does not require a wired connection for any data download or upload. Our client has stressed multiple times that this has to work, therefore, as we work on our final project we will be constantly testing our code to ensure it will have no problems in the field.

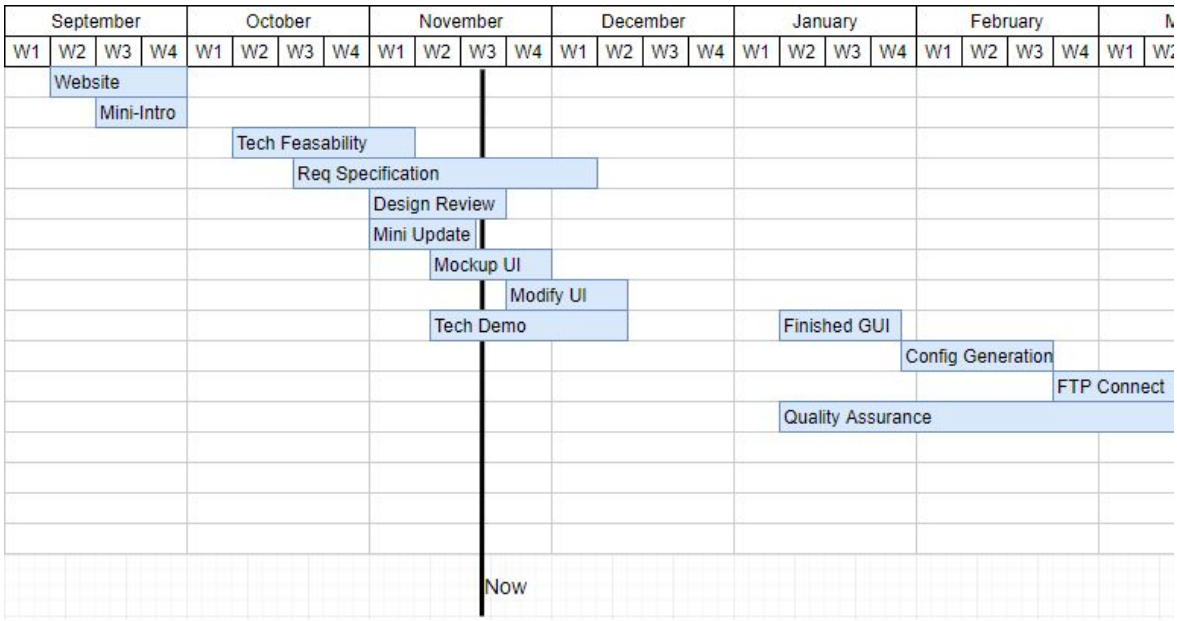

Figure 4 - Gantt Chart Schedule

9 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. After continuously obtaining feedback from our client, we have boiled our requirements down to those discussed in section 5 of this document. By continuing this regular communication, our requirements may change over time, but we are confident that the foundation we have laid out in our requirements acquisition document will exceed our clients expectations. While some of the risks we have discussed previously can certainly put a hindrance on our development cycle, we feel that our

solutions to each risk should suffice in mitigating their negative effects, should they occur. In the next developmental phase we will be finalizing the modified GUI on QGroundControl, as well as porting the configuration subsystem out of MATLAB and into QGroundControl.

10 Glossaries and Appendices

1) Schafer Michael, In-code documentation example. https://github.com/dynamic-and-active-systems-lab/UAV-RT/blob/master/POST_PROCESSING/uavrt_main.m
2) Out of code documentation example: https://uavrt.nau.edu/index.php/docs/control/

Glossary

Configuration File: In this application, a formatted text file that holds the radio frequencies and other settings that the drone uses to collect the radio telemetry data from the VHF tags.

UDP: Otherwise known as User Datagram Protocol is a protocol designed to manage a connection over the internet, unlike its cousin TCP (Transmission Control Protocol) UDP allows for a connection which has less overhead but less guarantee of data being received whole.

Flight Planning Software: a piece of software designed to control a drone much more precisely and consistently than a controller or other user-controlled device. Normally used to reduce crashes.

File Transfer Protocol (FTP): A set of rules that computers use to talk to one another in a network.

Graphical User Interface (GUI): A visual model for easily accessible input and output for a user.

Heartbeat: A periodic signal from the drone to a user's computer to indicate that the connection is stable and the drone is functioning properly.

Radio Telemetry (RT): A specific use of radio signals to aid in determining the location of VHF tags in relation to the drone.

Very High Frequency Tags (VHF Tags): A specific designation for radio frequency tags that broadcast between the ranges 30 and 300 megahertz (MHz).

MATLAB Runtime Environment: MATLAB is a programming language developed by mathworks while the MATLAB Runtime Environment is an environment built by the language for which its software is capable of running.
Mission Planner: An open source flight planning software developed by ArduPilot.
QGroundControl (QGC): An open source flight planning software by Dronecode.
QML: A programming language much designed to be used for building user interfaces quickly and easily.