RAMPART

Remote Aerial Mission Planning and Radio Tracker

Technological Feasibility Analysis

11/05/19

Team Members

Eric Gault

Samuel Gilb

Keller Mikkelson

Nathaniel Zeleny

Team Sponsor: Dr. Michael Shafer

Team Mentor: Mahsa Keshavarz

Table of Contents

1 Introduction

Wildlife radio telemetry is a very important tool for tracking the movement and behaviour of virtually any animal attached with a transmitter. There are multiple different methods to track animals in this way, but the use of very high frequency (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 alike in the future. 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. Their goal is to decrease the cost of having to locate these animals by using the drone to perform a more rapid, efficient, and complete scan of an area than 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, so that he can plan 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. There are a few 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 envisioned solution for these problems is to create a modified version of an open source flight planning software and integrate features from Dr. Shafer's personal software into it. We will accomplish this by doing the following:

- Modify the open source software to implement a copy of the configuration generator from our sponsor’s software that can connect and send the information to the drone.
- Implement a form of terminal in our own version of the software to allow the receiving of status updates from the drone without having to leave the original software.
- Develop another way of managing configuration data to minimize possibility of data corruption and ensure the drone will not crash if an error from our software arises.

The purpose of this document is to show and document our research into the technical end of our solution before doing any coding. The goal of this document is to outline our thoughts into the development of this project by comparing each of the different routes that a solution can be reached, and any possible problems that we may face along the way. The document will start off in section 2 where we will go into detail about possible technological challenges we may face while building our solution. In section 3 we will go into more detail about each of these challenges, possible alternatives, and how we go about judging which solution fits our needs the best. Section 4 will go over how we will integrate all of our different decisions together and how they will become a true solution to our sponsor’s problem. We will finish off this document in section 5 where we will give our best take on how to accomplish our goal based on the findings from the previous sections.

2 Technological Challenges

As mentioned in the introduction, this section of the document is where we will go into detail about challenges we may face during this project. We are aware that we may encounter other problems as we continue our research, however, the goal of this section is to discover any glaring issues we may run into before we run into them. During our talks about the project with Dr. Shafer, there were some apparent requirements that we would face in the course of development. The requirements originally given to us were programmatically high-leveled and generalized, rather than extremely detailed essentials. The challenges we will be facing are:

- **Open source flight planning software:** This is important as the software we are working with needs to have the functionality that our client needs as well as well written out documentation to ensure that it is readable by people without a strong programming background.

- **Documentation:** Determining how we will document our changes and where we will host our changes is also very important. We will not be supporting the product after our client signs off on it so any changes we make to the initial flight planning software needs to be easily understandable so it can be edited to our client's needs.

- **How configuration data is handled:** Determining a better way to handle configuration data to ensure no errors happen mid-flight. How the configuration is currently being handled is not well written and currently allows for fatal errors to occur while the drone is mid-flight so we need to modify this to ensure the reliability of the software.

- **How to create an FTP[1] connection between the software and the drone**: One of the main goals is to save our sponsors time, since his own software which contains important functionality is slow, we need to be able to implement

[1] File Transfer Protocol (FTP)

pre-processing functions, one of which is an FTP connection to the drone to allow for sending the drone configuration data.

- **If we will recreate the code written by our sponsor or change it to mesh with our software:** Since our sponsor has already written the functions he wants implemented into the open source flight planning software, we need to analyze the code to see if it is easily translatable to the new software, or if we need to write it using a completely different approach.

We will address the three most critical: open source flight planning software, documentation, and how configuration data is handled in the next section.

3 Technological Analysis

In this section we will be going into more detail in regards to a few of the major technological challenges that were briefly mentioned in the previous section. Before we begin any coding, there are a number of problems we need to address before we get started. These problems provide the backbone of our solution and are incredibly important to the feasibility of the product. For the sake of the document we will refer to the problems detailed below as design decisions.The first design decision we need to look into is what open source flight planning software we will be modifying. This is the most mission critical design decision as this defines our framework which we will be using to enable our client. The second design decision regards the best way to document our code, since our program will be used by more than just computer scientists. The documentation is important because we will not be able to support the product after we deliver it to the client (after the end of the capstone course). This documentation will allow for the software to be maintained and used for as long as our client needs it. The third and final design decision is how we will handle configuration data for the drone and its receiver. The handling of the config data is paramount to the drone flying and pathing required to collect data. For each of these different design decisions we will further discuss the metrics, the alternatives, and our decision on which option to utilize as well as our rationale below.

3.1 Drone Flight Planner

One of the most important things we need to decide on for this project is which open source flight planning software we will be using as a base to make our own version on top of. The reason choosing which software to work with is important is because all of our modifications will be on this software. Therefore we need to strike a balance between the aspects that will make our developmental process smooth as well as what aspects are most important to our client. At this time, there are only two software options that can both satisfy our client’s requirements and work with his current hardware: QGroundControl and MissionPlanner. When judging the viability of these two choices, there are a variety of different metrics that are important to us. Overall, each of these metrics are important in choosing a software that fits our needs as developers as well as our client’s needs as a user. These metrics are listed below and will be graded on a scale from one to five with one being the worst and five being the best:

- **Update Frequency:** We want this software to be useful for an extended period of time, therefore we want a software that is constantly up-to-date so that our client does not get stuck with an out of date piece of software.

- **Community Interaction:** We want to choose a piece of software that will have a long life-span and a thriving community of open-source developers. The bigger and more active the community, the larger the chance it will stay relevant and receive updates. This metric can also correlate to how easy it will be to find the answer to a question we may have about the software. If many people are contributing to a piece of software, then there will likely be more developers available to provide guidance when we need it. Whether it is a previously answered question, or one that we cannot find the answer to, a larger community usually accelerates the speed in which we can overcome the inevitable roadblocks that we will encounter.

- **Documentation:** We will be developing software that will be used by more than just programmers, so well-written documentation that is easily understood by layman is needed.

- **Ease of Use:** The software also needs to be easy to use. Our goal is to save time, so if the software we choose is difficult to use, the users will spend unnecessary amounts of time trying to figure out how to use it, our goal will be greatly diminished.

- **Maintainability:** Since we will not be around to constantly update the application and ensure nothing has broken over time, the software needs to be relatively easy to maintain so that those without extensive knowledge about the system can easily continue to use the application if an update is needed from the main software branch.

- **Speed:** We also need to ensure that the software is quicker to launch than our client's current MATLAB implementation. One of the problems we are fixing is a slow launch time, so choosing another software that is also slow to launch just moves our problem to another platform. We will find this by recording the launch times of each program 20 times on the same machine and taking the median of that data set. In the tables below, the speed of the program will be measured in seconds and a score will be assigned to it.

3.1.1 Alternatives

The first possible software choice is QGroundControl which was first released back in May of 2013 and is comprised of C++ and QML. The second choice is Mission Planner, which was written in August of 2010 with a combination of Javascript, C#, and .NET.

3.1.1.1 Choice 1: QGroundControl

QGroundControl is one of the options that was specifically mentioned by our sponsor in the Capstone project proposal. With its first release dating back to May of 2013, QGroundControl was developed to work with ArduPilot, which supports our client's PX4

Pro drone controller. This flight planning software works on all platforms including iOS and Android, making it one of the most portable options available. However, we will determine how well this software fits our needs by relating it back to our previously mentioned metrics.

Pros:

- Frequent updates by many different contributors.
- UI coding can be done relatively easily because it follows the QML format.
- Uses same UI code for all Operating Systems.
- Can be tested on a single platform since the code is functional across all platforms.
    - To check layout flow, you must use on device type to be used.
- Has an architecture built into it which allows custom builds to modify and add to the feature set of the vanilla program.
- Fairly active community.
    - Frequency between posts and replies are good, but usually have one or two days between each post.
- Has an in-depth development guide, but seems unnecessarily wordy.
- Provides a boilerplate object-oriented API for tracking the “heartbeat” pings of the drone.

Cons:

- May not provide best UI layout for all OSes; for example laptop layouts may not be optimized for a tablet or phone layouts.
    - Client uses a laptop when collecting data out in the field.
- Not the current flight planner being used by the client.
- Due to the complexity of the QML used in QGC as well as its reliance on communication with C++ objects to drive the UI it is not possible to use Qt Designer (MVC) to edit this software’s QML.

QGroundControl Metrics

Table 1 - QGroundControl Metrics

	Update Freq	Community	Documentation	Ease of Use	Maintainability	Speed (Median launch time)
QGround Control	4	4	3	4	5	5.32 sec Score: 4

Update Frequency - We gave this a score of 4 due to the amount of stable releases provided on the QGroundControl website [11]. While QGroundControl has about as many stable releases as MissionPlanner, the amount of issues resolved by its surrounding community gives it a considerable edge in terms of active updates.

Community - The Github metrics [6] imply that there are many active and progressing contributors to the program itself. The frequent activity in the official QGroundControl forums[12] show that any questions we may have while using this framework will be answered eventually. We gave it a score of 4 instead of 5 because while the community is very active and knowledgeable, it’s not a resource that is instantly accessible.

Documentation - The interactive “manual” provided online is useful, but can be hard to navigate and understand. This also provides a small developer guide [9] to help get things started, but touts Gitter (a Github-based chat application) as a means of getting direct guidance from fellow developers. While Gitter helps with the score, we decided that a 3 is more deserving instead of a 4 because of the difficulty of navigation.

Ease of Use - From our preliminary tests, QGroundControl appears quite easy to use, the landing page of the application shows your current location as well as the drone’s (if connected). Each tab on the top of this page is large and specific towards what you would find on that tab without having to use any text for naming them. We gave this a 4 instead of a 5 because certain data readouts were not as apparent as the rest of the program.

Maintainability - QGroundControl documentation [11] has a page specifically on keeping custom builds up to date with the main. It has specific tutorials on how to accomplish this and appears straightforward enough for non-computer scientists to be able to easily use. We gave this score a 5 because the builds were displayed with the fixes they did on that version and the simplicity of the guides.

Speed - The mean startup time for the QGroundControl software is 5.32 seconds. This is already 1/60 the startup time for our client's current system, making it a solid contender in the category of sheer speed. We give this score a 4 instead of a 5 because while very fast, it is not the fastest.

3.1.1.2 Choice 2: MissionPlanner

Mission planner was another option listed by our sponsor as well as the software he is currently using. It is a flight planning software that is provided on the ArduPilot website that was was built by Michael Osborne specifically for the autopilot software.

Pros:

- Somewhat frequent updates; mostly by the creator.
- Extensive guide that shows how to do specific tasks and various .
- Very active community.
  - Frequency between posts and replies are very good; almost always the same day.
- Client is already familiar with the software.
- Does not have direct support for a "heartbeat" range tracking system, so one must be implemented from scratch.

Cons:

- Not very intuitive UI implementation.
  - Altering the backend has a steep learning curve.
- Code does not seem to be commented with any consistency.
  - Used as reminders or to temporarily disable code.
- Only runs on Windows and Mac OS X.

- Does not appear to have a development guide to help us understand how to alter the program without breaking it.

MissionPlanner Metrics

Table 2 - Mission Planner Metrics

	Update Freq	Community	Documentation	Ease of Use	Maintainability	Speed (Median launch time)
Mission Planner	2	3	2	4	1	3.6 sec Score: 5

Update Frequency - MissionPlanner had about 7 to 10 stable releases a year[7], but that number seems to be getting lower and lower as the years go by. Most of the recent updates and fixes have been made directly by the creator, and there seems to be a hemorrhaging of Github issues that are not being addressed[2]. We gave this a 2 because while there were updates, there was almost nothing of substance in those updates.

Community - According to Github metrics[2], there are not many active contributors to the program itself, but the frequent activity in the official MissionPlanner forums show that any questions we may have while using this framework can be answered in a timely fashion. We gave this score a 3 instead of a 4 because while there is an active and helpful forum, there aren't enough people to corroborate what the responder is saying.

Documentation - The wiki hosted on the MissionPlanner website[7] is incredibly useful for learning how to use the software, but gives little insight on how to actually contribute or alter the program without breaking things. No official developer guide is available. It may be easy enough to pick up the code and run with it, but knowing the proper procedures and guidelines for developing this software is incredibly important for our project. Due to the issue with the developer guide, we give this score a 2 even though there are some helpful set-up guides.

Ease of Use - In our preliminary tests, Mission Planner appears quite easy to use as the main page shows your area along with the currently planned flight path. The buttons and their functions can easily be determined on the top of the screen by the contrasting images as well as name descriptors for them. We give this score a 4 instead of a 5 because the UI in whole is fairly dated and needs an upgrade.

Maintainability - Based on the documentation[7] there are no specifics related to updates if you are running a custom build of Mission Planner. The software will notify the user when it is out of date but we have no knowledge regarding how this will affect the modifications we make in the long-term. We gave this score a 1 because of how poorly designed the update system they have in place.

Speed - The mean startup time for the Mission Planner drone flight path software is 3.6 seconds, making it roughly 1/100 of the startup time that our client is currently dealing with. It surpasses QGroundControl by a solid 1.7 seconds, meaning it is the fastest choice listed here. Because it is the fastest by the tests, we will be giving it a score of 5.

3.1.2 Chosen Approach

Table 3 - Chosen Approach for Flight Planning Software

	Update Freq	Community	Documentation	Ease of Use	Maintainability	Speed (Launch Time)	Score
QGround Control	4	4	3	4	5	4	24/30
Mission Planner	2	3	2	4	1	5	17/30

Overall, Team RAMPART believes that QGroundControl will serve our needs the best. Not only do many of our members have experience working with the Qt framework, but the community involved with QGroundControl also seems far more active than what we've observed with Mission Planner. While Mission Planner has been around longer, QGroundControl exhibits better portability and less developmental software requirements. When looking at the median times of both software, there is not a significant enough difference in startup time to pick a clear winner. It is to be noted that Mission Planner had a faster median startup of 1.72 seconds, however the software we are replacing (MatLab) takes anywhere from 1 to 10 minutes. As the GitHub statistics show, QGroundControl has made itself one of the easiest pieces of drone software to pick up and actively contribute to. While we may not end up sending pull requests directly to the official repository, the ability to get our questions answered in a timely fashion by the people building and maintaining the software can be an incredibly important aspect when making our decision. We feel that QGroundControl possesses the community and the accessibility that we need to build our client what he wants.

3.1.3 Proving the Feasibility of our Decision

A way to prove the feasibility of our decision would be to develop a user interface (UI) mock up of what our client wants. We can add placeholder elements to the UI to show what our potential software would look like. While we are working on this we can also determine if we made the right choice for each design decision. A big factor in these decisions being how easy or difficult it was to make the modification requested by our client. Once we have added the elements, we can show this to our client and get their feedback regarding how it looks and if it would suit his needs were we to continue with this design.

3.2 Program Documentation

Another Important facet in our project is the robust documentation of our code. Our client has stressed that he wants engineers and scientists with no computer science background to be able to easily pick up our software and modify it if they need to. The main functionality our wiki needs to provide is a quick and thorough answer about each part of our software. We have decided to measure the options based on some key metrics. These metrics are listed below and will be graded on a scale from 1 to 5 with 1 being the worst and 5 being the best:

- **Maintainability:** This measures how simple adding or updating content is. Maintainability is one of the key metrics because we will not be updating the software once we deliver the product and if our client wants to update features or change wording he needs to be able to with relative ease.

- **User Accessibility:** This measures how technically versed one must be to utilize the software. Accessibility is one of the key metrics in our decision making process because the software will be open source. This means that users should be able to access the functionality of our software with minimal background in computer science.

- **Customizability:** This measures how much the software can be tailored to our needs. Customizability is an important metric because our client wants an aesthetically pleasing form of documentation that is easy to read and unique enough to distinguish from other documentation created with the same framework.

- **Version Control:** This measures how easy it is to both branch and merge versions of the wiki. Version Control is an important metric because our software will be open source and so the wiki should reflect how the different versions and add or remove functionality based on the version and the branch.

- **Ease Of Use:** This measures how simple the software is to use. Ease of Use is a key metric because our software will be open source and if users create new functionality they should have an easy path to documenting the changes.

3.2.1 Alternatives

Since we will be using Github, a widely-used code repository, for our version control and change review management, our code documentation options are fairly limited. However, each of these options have definitive pros and cons, and we have narrowed down the possible frameworks available to: a GitHub Wiki, Source Forge, and a GitHub README file. While there are many more options in this category, we felt that these frameworks would have the largest potential to score high ratings with regard to our metrics, and work well with our current GitHub repository.

3.2.1.1 Choice 1: GitHub Wiki

Founded in 2008, GitHub has solidified itself as one of the largest, most frequently-used open-source code repositories in the world. As GitHub has evolved over the years since its inception, so too has its documentation features. Currently, our client's code is hosted on GitHub, thus, utilizing the pre-established GitHub documentation as a base seems like a cohesive and useful way to document the usage of our software.

Pros:

- Has the same lifetime as GitHub.
- Hosted on the same site as our Client's existing codebase.
- Easily Accessible from the project page on GitHub.
- Allows you to see the history of changes.
- Support for easy formatting.
- Wiki page may be downloaded locally.

Cons:

- Web-based so an internet connection is required to get the most up-to-date version.
- Only for projects on GitHub.

GitHub Wiki Metrics

Table 4 - Github Metrics

	Maintainability	User Accessibility	Customizability	Version Control	Ease of Use
GitHub Wiki	4	5	5	5	5

Maintainability - Fairly easy to maintain[4] however a point is docked for the wiki being exclusive to GitHub as migrating the wiki is next to impossible if our client wants to remove the project from GitHub.

User Accessibility - This is easily accessible as our project will be hosted on GitHub and by clicking the “wiki” tab the user will be brought to it. A very low amount of technical skill is required to access the wiki[1]. We give this score a 5 instead of a 4 because of the extreme ease of use.

Customizability - GitHub wiki’s support a variety of markup languages allowing for a robust selection for text design and styling[4]. Because of this high customizability, we give this a score a 5 instead of a 4.

Version Control - The software allows for multiple versions of the wiki to exist as well as management of editing privileges and provides a tool for users to compare revisions[1]. We give this score a 5 instead of a 4 because it allows us to fix mistakes quickly and update the wiki as needed without fear of erasing important information.

Ease of Use - This software is very user friendly and provides intuitive controls for users[4]. We give this score a 5 instead of a 4 because we cannot find any glaring issues with using the software; it does what we need it to do and it’s easy to do it.

3.2.1.2 Choice 2: SourceForge

SourceForge supports Git as well as GitHub so the documentation is similarly accessible as GitHub and provides similar functionality. Many large open source groups use SourceForge as their version control and documentation because it is an all-in-one software. This means that their documentation, version control, even a forum can be hosted on their SourceForge page.

Pros:

- SF projects can be integrated with Git.
- Allows for “multi ticket” trackers that allows for separate ticket trackers for corresponding groups or subjects.

Cons:

- Hosted by a different website than our main repository.
    - May make it harder to find than other documentation methods.

    - If SourceForge stops supporting their website, our program will lose its main form of documentation.
- The website hosting the documentation may go offline when someone needs it.

SourceForge Metrics

Table 5 - SourceForge Metrics

	Maintainability	User Accessibility	Customizability	Version Control	Ease of Use
SourceForge	4	3	5	5	3

Maintainability - The SourceForge maintainability is extremely similar to the GitHub wiki, because of the same basic skeleton they use[10]. However, instead of being tied to GitHub, it is tied to SourceForge hosting making it the same problem with different variables. We give this a score of 4 instead of 3 because the software is an all-in-one, making changes easy to do because everything is right on your profile.

User Accessibility - After looking at the setup guides[12] and well known SF projects (like notepad++ plugin manager, Apache OpenOffice, and Moodle), the UX of the website is garish almost. We give this score a 3 instead of a 4 because everything that the user needs it there, but there is little consistency with tabs, meaning the user has to re-learn where all the information they need is every time they look at a new project.

Customizability - This is where SourceForge shines, it gives a lot of power to the project holders[12]. Project holders can really make their personal project pages the way that they want and how they want a user to experience their project page. However, this can be seen as a double edged sword because too many choices or variability makes a bad user experience if there are as many separate projects as SourceForge has. We give this score a 5 instead of a 4 because this is focusing on how customizable the site is, and it is very customizable.

Version Control - Again, this is fairly similar to GitHub Wiki even down to non-destructive edits, but just being hosted on SourceForge[12]. The non-destructive edits allows us to revert exactly back to what we want if there needs to be a rollback. We give this a score of 5 because of the extreme similarity of how it works compared to GitHub’s Wiki.

Ease of Use - When trying to create a dummy project[12] and register as a project owner, it was fairly convoluted, the guides were not super easy to find nor were other resources. For example, when we did find the guides to set up our own project, some of the guides on what to do were either not filled out correctly, or simply just missing important information. This isn’t something we want for documentation, but the guides that were filled out were quite helpful, that’s why this score is a 3 instead of a 2.

3.2.1.3 Choice 3: GitHub README File

This is a plaintext file on the Project Page of GitHub. It is mainly used to let the users know what the project does, who it’s creators are, and potentially a small startup guide[13]. They are meant to be brief but informative enough for users to refer back to.

Pros:

- The file is available anywhere the program is installed.
- Linked directly with the GitHub.

Cons:

- Any alterations to the README file must be merged to the main branch in order for the changes to take effect.
- Designed to be used as a “get to know me” guide for users and not a full set of documentation.

README Metrics

Table 6 - README Metrics

	Maintainability	User Accessibility	Customizability	Version Control	Ease of Use
GitHub README File	1	5	2	1	1

Maintainability - This would be nigh impossible to maintain because updating requires that the change be part of the main branch[13]. We gave this a 1 instead of a 2 because while the README is supposed to be a fairly simple file, changing it is that big enough of a headache.

User Accessibility - This format is fairly easy to access regardless of other issues. Since the file is lo-tech[13], almost any computer would be able to run it and it is easily findable on the project page, we give this score a 5..

Customizability - There is not much to customize with this option as it's mainly plain text. However, we can add links or extremely simple visual changes to help with comprehension[13]. The visual changes and linking ability is the reason why the score is a 2 and not a 1.

Version Control - Only the main branch is documented and we have to manually monitor history[13]. This means if a mistake is made, it could be quite difficult to revert our instructions or guide to a previous version of it. Because of this, the score is a 1 due to the difficulty of reversion.

Ease of Use - This would be a slab of text no matter how it is formatted and features implemented later would become harder and harder to find[13]. Making users read large amounts of text for an answer to a quick question is tedious and boring. We don’t see this format being easy to navigate and that is why we give it a score of 1.

3.2.2 Chosen Approach

Table 7 - Chosen Approach for Documentation

	Maintainability	User Accessibility	Customizability	Version Control	Ease of Use	Total Score
GitHub Wiki	4	5	5	5	5	24/25
SourceForge	4	3	5	5	3	20/25
GitHub README File	1	5	2	1	1	10/25

For documenting our software and its processes, Team RAMPART believes the GitHub Wiki will serve our needs the best. While SourceForge is a good competitor, it encompasses more than what we need as a team and what future users of the project will need. The amount of customization and freedoms SourceForge provides is unnecessary for what the project is about. As for using solely GitHub README files for documentation and walkthroughs, it would be the complete opposite of what SourceForge is in terms of flexibility and ability. The accessibility of the README files are spectacular as GitHub has a place to provide that automatically, but we should see it as a way to show our Table of Contents or provide links instead of a fleshed out documentation choice. The GitHub Wiki provides the best of both worlds for us; we are given the ability to customize our documents but not have it be over the top. The Wiki allows users to easily navigate between topics in the documentation and keeps the design constant without tabs and other visuals moving into other spots. GitHub Wiki appears to be the best choice for us to document our work and easily create walkthroughs or guides for the end users. If we felt like that was not enough, we can provide users a short README file in conjunction with the Wiki.

3.2.3 Proving the Feasibility of our Decision

The way we can prove our decision is the right one can be by creating a mock documentation or guide for our product using the GitHub Wiki format. That way we get even more experience with the process of how the Wiki is made and we could potentially use it as a skeleton for the legitimate documentation for the project. If that does not seem to be enough for a user to get a grasp of the project, we can include a short GitHub README. The README would give a concise summary of the project and where to find the documentation that goes further into detail about it.

3.3 Configuration File Processing

Our sponsor has already implemented configuration handling in his own software, however it is not well designed and allows for errors to occur mid-flight that cannot be recovered from and ultimately resulting in the drone crashing. We need to create a new way to handle the formatting of the configuration data to allow for better management of the data and to ensure all errors are caught and can be handled. The core of this project relies on porting our client's MATLAB-based configuration file processing method to whichever drone flight planning software we inevitably choose. Since the possibilities are tightly coupled with the programming languages we will be using, some of these options are dependent on which flight planning software is ultimately selected. We need to select a processing method that is not only computationally efficient, but also simple to modify, should any aspect of the drone's configuration file change in the future. We will be using the following metrics to judge which file processing method will be ideal for our needs these metrics are measured on a scale of 1-5 with 5 being the best rating:

- **Adaptability:** This measures how easily the configuration processing method can be adapted should the user want to alter their configuration file standards to have more or less information. As the entirety of our client's drone system evolves, so too must the software. Since our software may be utilized long after we have finished this project, the ease of which our processing method can be changed is paramount to the longevity of our contribution.

- **Speed:** Since improving the startup speed of our client's drone system is at the core of why this project exists, we don't want to diminish the effectiveness of our improved startup times with inefficient file handling methods.

- **Readability:** While this metric is similar to adaptability, the future maintainers of this program must be able to quickly and clearly understand how the configuration file processing works before they can effectively make changes. While comments and documentation can help to some extent, having configuration file processing code that is easy to read at a glance can greatly improve the time it takes other developers to understand and change the code if needed.

These metrics are measured on a scale of 1-5 with 5 being the best rating.

3.3.1 Alternatives

There are many options to be considered when deciding on which method of file processing should be used in our program. Considering the languages that are supported by our drone flight planning software, four feasible possibilities will be considered here: Processing files with utility functions, object-oriented file processing, processing files with QML, and file processing with .NET streams.

3.3.1.1 Choice 1: Processing Files with Utility Functions

While both pieces of drone flight planning software support object-oriented languages, the option of implementing standalone utility functions is also available for each. Our first instinct may be to abstract our configuration files with a class in an object-oriented language, it is our job to consider all the available options for this design decision. Developing with the C programming language would require us to utilize this paradigm, but since our drone flight planning software options utilize object-oriented versions of C, the implementation of standalone functions provides a slightly different result.

Pros:

- No need to instantiate a class in order to perform file input and output.
- Not encapsulated in a class, making this method slightly easier to read and write.

Cons:

- Must store configuration data in structs, which have essentially the same overhead as an object in object-oriented variants of C such as C++ and C#.

    - Structs are historically harder to reason about and work with than well-constructed Classes.
- Future developers may add additional functions for configuration processing and utilization, which can be distributed among various different files.
    - Distributing a tightly-coupled set of functions across many different files can be a nightmare to reason about.

Metrics for Processing Files with Utility Functions

Table 8 - Utility Function Metrics

	Adaptability	Speed	Readability
Utility Methods	3	5	3

Adaptability - Relatively easy to add functions as needed, however these functions can end up distributed across numerous files. According to a stackexchange post, that would cause problems with debugging and future modification[3]. Because of the potential debugging problems it could cause, we give this score a 3 instead of a 4.

Speed - The functions can be well optimized by the team to have just what is needed and have no bloat, meaning the functions will not be doing anything extra[3]. If optimization goes correctly, the speed of which these functions can operate is up to us as the developers. Also, since a class would not have to be instantiated to run the functions, the actual file input/ output would be very fast and justifies a score of 5 instead of 4.

Readability - If all functions were to be maintained within one file, readability should be quite easy as everything is located in one location[3]. However, future modifications may be made after our time on this project. This means that problems may occur further down the line. Since we would have less control over the readability of future implementations, we decided to give utility functions a score 3 but not a 4.

3.3.1.2 Choice 2: Object-Oriented File Processing

Abstracting configuration files as objects can be a very powerful way to not only keep our file processing method compact and modular, but also provides a logical way to abstract and understand how our program works. Since both pieces of done flight planning software utilize an object-oriented variant of C, these languages are geared towards creating robust and efficient class-based programs.

Pros:

- Provides a modular and intuitive way to abstract successfully uploaded configuration files.
- Easy to reason about, even if the user lacks an extensive programming background.
- Objects can be serialized to easily and efficiently save and reload configuration files for future use.
- Class member variables and methods allow for the use of standardized, highly-readable documentation.
- Inheritance may be utilized if future developers wish to modify the format or specifications of the current configuration file structure while maintaining backwards compatibility with older configuration files.

Cons:

- The programming language associated with our selected drone flight planning software must support the use of objects.
- Additional overhead due to Object instantiation.

Metrics for Handling Files with Objects

Table 9 - Object Metrics

	Adaptability	Speed	Readability
Objects	5	4	5

Adaptability - In the same stackexchange post about utility functions, it states objects can easily be modified as everything stays within the confines of the object itself[3]. The object may end up more complex but will maintain its great adaptability. Even though there could be a level of complexity to the object, we still give this score a 5 because of its virtually endless adaptability.

Speed - As the complexity of the object increases, so too does the size and complexity of the overhead associated with said object[3]. Therefore as long as we can keep the complexity of our object small, we can continue to have fast speeds when working with it. Since we may have to have some degree of complexity there is going to be some slowdown, which is why this score is 4 and not 5

Readability - As everything is within the object itself, everything relating to it is easily readable and allows for comments to be written inside of it to help non-programmers understand the logic of things within it. As long as we keep a good form of commenting, the readability will be fantastic, making this score a 5.

3.3.1.3 Choice 3: File Processing with QML

QML, or the Qt Modeling Language, is usually used to format and structure user interfaces. It is currently the primary language used to handle and display the UI for QGroundControl, meaning it cannot be a valid option if we select MissionPlanner as our software base. While it is unconventional, QML can actually perform file input and output, but it is uncertain what niche this could fulfill within the scope of our project.

Pros:

- May be easier to work with if users have experience in web programming rather than object-oriented C languages thanks to its use of Javascript.

Cons:

- Only feasible with a flight planner that uses the QML, such as QGroundControl.
- Built more for user interface structuring rather than handling files
- Cannot be written with just QML syntax; inline Javascript or Javascript files would have to be utilized to script the file I/O

- Adding the use of additional languages to our program would create additional barriers to entry for those interested in reading, updating or customizing our program.

Metrics for Processing Files with QML

Table 10 - QML Metrics

	Adaptability	Speed	Readability
QML	3	1	2

Adaptability - If good Javascript coding conventions are used, changing certain aspects of the configuration file processing operations wouldn't be exceedingly difficult. However, if advanced Javascript techniques are used, such as arrow functions[8], altering the code can be a dangerous and arduous process. Since this could be a very fine line between good and bad, we give this score a 3 instead of a 4.

Speed - Javascript would be the driver behind uploading and pulling data from configuration files in this case. Compared to even object-oriented C languages, Javascript is historically much slower, as it is a high-level scripting language[8]. In most cases, QML is used as a framework for user-interfaces, and should be as "thin" a layer as possible. Bloating a program's QML with additional functionality could drastically slow the response times of our GUI. Because of the large bloat, we have to give this score a 1.

Readability - Since Javascript would be required to implement File I/O in QML[8], it can be jarring for users and developers to switch between upwards of three different programming languages when examining our code. On top of this, having Javascript potentially mixed into QML files can be increasingly difficult to reason about. Increasing the amount of our languages and mixing code is the reason the score is a 2 instead of 3.

3.3.1.4 Choice 4: File Processing with Microsoft .NET Streams

.NET Streams provide a framework for synchronous and asynchronous I/O in files and data streams using “namespaces”. The system allows for these “namespaces” to communicate using pipes and serial ports as well as compress and decompress files depending on the “namespace” type.

Pros:

- Well written documentation provided by Microsoft.
- Works well on any windows operating system.

Cons:

- Only feasible with a flight planner that uses the .NET framework, such as MissionPlanner.
- Neither the client nor any team member has experience working with .NET

Metrics for Processing Files with .NET Streams

Table 11 - .NET Streams Metrics

	Adaptability	Speed	Readability
.NET Streams	2	3	5

Adaptability - The adaptability of .NET is nonexistent for any system that is not currently using a .NET Framework. Therefore, we gave it a score of 2 because the product would need to be coupled to a .NET framework[5] to work on any operating system. However, we decided against giving it a 1, since the .NET environment could ensure that the code is clean and adaptable thanks to its many supported languages and libraries.

Speed - While .NET can rival or even surpass C++ and C# in terms of raw computational speed, the fact that it utilizes automatic garbage collection[5] makes it overall slower during runtime. Because of the automatic garbage collection, the score is a 3 instead of 4.

Readability - Because of its Common Language Infrastructure, .NET streams can work with a vast array of programming languages[5]. This gives programmers the ability to write code using whichever language they feel will be the easiest to understand for those who need to read and understand the code. Because of this functionality, we can give this score a 5 instead of 4.

3.3.2 Chosen Approach

Table 12 - Chosen Approach for File Handling Metrics

File Handling	Adaptability	Speed	Readability	Score
Utility Methods	3	5	3	11/15
Object	5	4	5	14/15
QML	3	1	2	6/15
.NET Streams	2	3	5	10/15

For managing our configuration data, Team RAMPART believes that storing the information in a C++ object will suit our needs the best. .NET streams fill all of our requirements in a config manager, however due to it requiring a flight planner using the same framework and our previous mention of choosing to work with QGroundControl, we will not be able to implement this into the program. While QML is integrated within QGroundControl, it still requires some form of backing with either more C++ or javascript, in which case there is no more need to be using QML and we could work without it. Utility functions also were able to handle what is needed, however since they still require an overhead related to those of C and C++, we would be better off using those along with the structure to help give them better readability. Overall, handling files with an Object best suits our needs as it is easily adaptable, easy to read, and is not slow enough to cause any problems.

3.3.3 Proving the Feasibility of our Decision

The way we can prove that we made the right decision in regards to our file handling by creating a sample file built by our choice. Our sponsor should easily be able to read the code that was written as well as the output. Although he may not understand all of the coding at first glance, it should be fairly simple to walk through line by line and able for non-programmers to be able to see what it is doing. [Figure 1]

3.4 Proving Feasibility

Overall we can prove the feasibility of our design decisions by creating a quick demo showcasing the implementation of our choices. We can show we made the right decision in regards to which open source flight planning software we chose. We can implement a basic configuration generator as well as a mock terminal to show that we are able to implement UI changes into the software. We can show that we made the correct decision in regards to configuration handling by implementing a skeleton version, it can have all of the required fields for the configuration as well as be able to generate it. Although it may not be connected to the drone at the time, it will show that we are able to generate the needed configuration from inside the application. We can then prove we made the correct choice in regards to documentation by documenting everything we did for the previous two demonstrations.

4 Technology Integration

Now that all of our design decisions have been laid out in the previous section, we can put each of our choices into a simple system diagram to analyze our solution on a broader scale. The Drone is controlled by the computer that our client brings into the field. On the computer our client will launch our modified version of QGroundControl With a GUI allowing for the C++ config file object to be generated and uploaded to the drone through QGroundControl. Finally all documentation on how to use our unique version of QGroundControl will be created and stored in a GitHub Wiki.

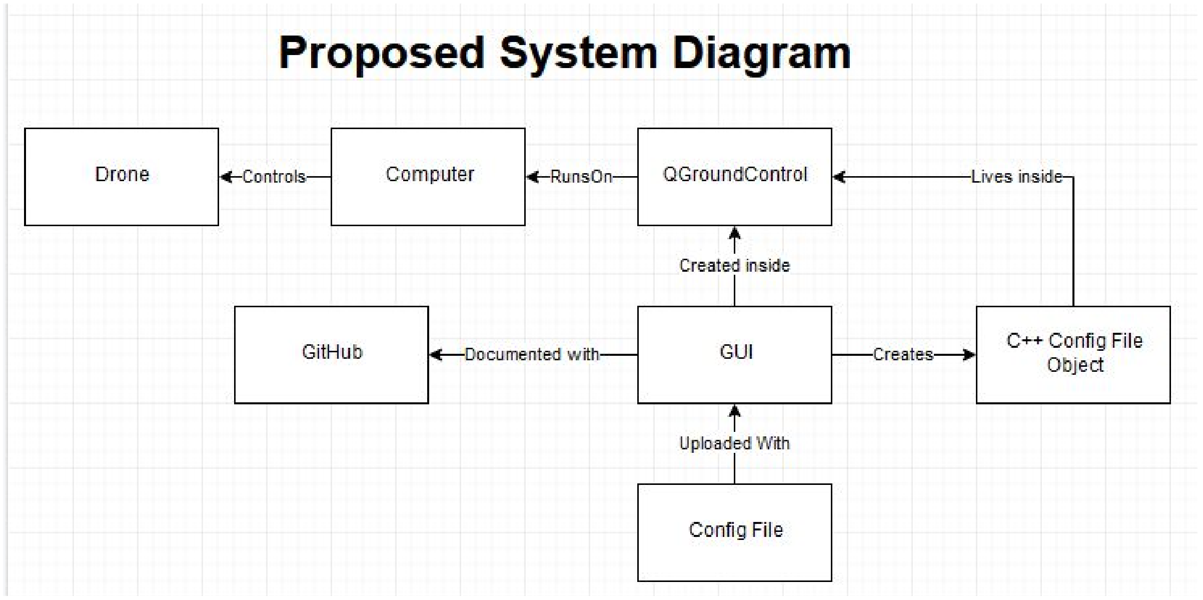

Figure 1 Proposed System Diagram

To further elaborate on the proposed system diagram, our task is to port the existing GUI from our client's current MATLAB program in order to streamline and speed up the uploading and tweaking of his configuration files. To do so, we will implement the GUI within QGroundControl: an open source drone flight planner that uses the C++ and QML programming languages. Our modifications will be extensively documented with GitHub's wiki system, and upon successful upload, the contents of the configuration file will be stored in a C++ object to allow for simple, modular access to the data. Both the GUI and the configuration file object will live inside the QGroundControl environment, meaning that our client and his colleagues will be able to handle more of their drone data collection workflow with one singular program that has a much faster startup time than his current implementation. Thus, our users will not need to start up the MATLAB runtime environment in order to send their drone on a mission, considerably simplifying and accelerating the rate of which data can be collected by the drone.

5 Conclusion

By integrating the technological choices we have listed throughout this document, we feel that it will be very possible to achieve our ultimate goal of improving the efficiency and usability of our client’s software. Our team feels that if our goals are attained, we will contribute not only to NAU’s Dynamics and Active Systems Lab, but also the field of environmental Biology as a whole. As more VHF radio tags make their way onto more wildlife specimens, the amount of biological data to be collected from frequently scanning these tags goes up exponentially. Our client, Dr. Shafer, has found an elegant solution to the problem of safely and efficiently collecting the data, but it is our job to improve upon his current solution. To do this, Team RAMPART has been tasked with adding additional features with robust documentation and error handling to a piece of open source drone flight planning software that is compatible with our client's UAV-RV system. Overall, our project is to modify an open source flight planning software and implement a configuration generator and a terminal to allow heartbeat messages from the drone to be shown in one application. We also need to ensure that everything is well-documented to ensure the viability of this software as time progresses. In order to accomplish this, we have selected which open source flight planning software to use, the kind of documentation to employ, and how the configuration file should be processed and stored when using our customized branch of said flight planner. The purpose of this document was to explain our decision making process when picking each of these important items by listing the important metrics for each category and examining how each option adheres to them. By doing so, we have documented why we feel each of our chosen design decisions are correct, solidifying the foundation on which we will start one of the most important projects of our academic careers. The following tables quickly outline the metrics and corresponding ratings (from zero to five points) that each possible choice received for each design decision listed here:

5.1 Open Source Drone Flight Planning Software

Our confidence level is 4/5 for this choice. While MissionPlanner has the edge on raw startup speeds, we are fairly confident in our choice. However, we have yet to delve into modifying either piece of software in the way we will have to in the future. There may be unknown issues with QGroundControl that could hinder our development, but the thriving community of open source developers contributing to the software gives us a feeling that this will not be the case. Overall we feel that QGroundControl better suits our needs for this project.

Table 13 - Flight Planning Software Metrics comparison; our selection is in bold print.

	Update Freq	Community	Documentation	Ease of Use	Maintainability	Speed (Launch Time)	Total Score
QGround Control	**4**	**4**	**3**	**4**	**5**	**5.32 sec**	**20/25**
Mission Planner	2	3	2	4	1	3.6 sec	12/25

5.2 Program Documentation Framework

We are extremely confident that the GitHub Wiki will be the best solution to our documenting needs; enough to give it a confidence rating of 5/5. It beats SourceForge in user accessibility and ease of use which are very important metrics that our client has stressed.

Table 14 - Documentation Metrics comparison; our selection is in bold print.

	Maintainability	User Accessibility	Customizability	Version Control	Ease of Use	Total Score
GitHub Wiki	**4**	**5**	**5**	**5**	**5**	**24/25**
SourceForge	4	3	5	5	3	20/25
GitHub README File	1	5	2	1	1	10/25

5.3 Configuration File Processing

We are confident that file handling and processing via an object is the best choice for our project as it is easily adaptable to whatever may be needed in the future. On top of this, class files are generally easy to reason about, since everything pertaining to it is located in one location. The speed of it varies depending on the size of the object itself, but for what we need it for, the speed reduction from class instantiation should not be a problem. Since each member on our team also has extensive experience with object-oriented programming, we are giving our choice a confidence rating of 5/5.

Table 15 - File Handling Metrics comparison, our selection is bold

File Handling	Adaptability	Speed	Readability	Score
Utility Functions	3	5	3	11/15
Object	**5**	**4**	**5**	**14/15**
QML	3	1	2	6/15
.NET Streams	2	3	5	10/15

Overall, after rigorous research regarding each topic, our team feels that we made sound decisions when assessing the options for each design decision. While our options were limited in some categories, we made sure to properly outline each of the possible options we came across, and carefully weighed the pros and cons of each before coming to unanimous consensus for each of our choices. There are still a few open questions, such as “what will be the best way to handle errors?” and “what kind of testing suite should be utilized” the answers to these questions are tightly coupled with our selection of flight planning software, since we want to use the standards that are currently being used by the original developers of said software. While we may discover more requirements and difficult hurdles as we continue working on this project, our team is confident that the important design decisions we outlined previously will remain the best choices as we constantly seek to adapt our solution to fit our client’s ultimate vision.

6 References

1. Anon. About wikis. Retrieved October 22, 2019 from https://help.github.com/en/github/building-a-strong-community/about-wikis
2. ArduPilot. 2019. ArduPilot/MissionPlanner. (October 2019). Retrieved October 22, 2019 from https://github.com/ArduPilot/MissionPlanner
3. Cybergibbons et al. 1964. What overheads and other considerations are there when using a struct vs a class? (June 1964). Retrieved October 22, 2019 from https://arduino.stackexchange.com/questions/658/what-overheads-and-other-considerations-are-there-when-using-a-struct-vs-a-class
4. Anon. Documenting your projects on GitHub. Retrieved October 22, 2019 from https://guides.github.com/features/wikis/
5. Dotnet-Bot. Stream Class (System.IO). Retrieved October 22, 2019 from https://docs.microsoft.com/en-us/dotnet/api/system.io.stream?view=netframework-4.8
6. Mavlink. 2019. mavlink/qgroundcontrol. (October 2019). Retrieved October 22, 2019 from https://github.com/mavlink/qgroundcontrol
7. Anon. Mission Planner Home. Retrieved October 22, 2019 from http://ardupilot.org/planner/
8. Christopher PetersonChristopher Peterson. 1962. How do I read in FILE contents in QML? (January 1962). Retrieved October 22, 2019 from https://stackoverflow.com/questions/7773994/how-do-i-read-in-file-contents-in-qml
9. Anon. QGC - QGroundControl - Drone Control. Retrieved October 22, 2019 from http://qgroundcontrol.com/
10. Anon. 2019. The Complete Open-Source and Business Software Platform. (October 2019). Retrieved October 22, 2019 from https://sourceforge.net/
11. Anon. 2019. QGroundControl User Guide. Retrieved October 30, 2019 from https://docs.qgroundcontrol.com/en/releases/release_notes.html
12. Anon. 2019. SourceForge.net Documentation. Retrieved November, 1 2019 from https://sourceforge.net/p/forge/documentation/Docs%20Home/
13. Anon. 2019. About READMEs. Retrieved November, 1 2019 from https://help.github.com/en/github/creating-cloning-and-archiving-repositories/about-readmes