RAMPART

Remote Aerial Mission Planning and Radio Tracker

Final Report Version 1.3

May 5, 2020

Team Members

Eric Gault

Samuel Gilb

Keller Mikkelson

Nathaniel Zeleny

Team Sponsor: Dr. Michael Shafer

Team Mentor: Mahsa Keshavarz

Table of Contents

Table of Contents **2**

1 Introduction **3**

2 Process Overview **4**

3 Requirements **5**

3.1 Functional Requirements 5

3.2 Non-Functional (Environmental) Requirements 8

4 Architecture and Implementation **9**

4.1 System Diagrams 10

4.2 Code Files Diagram 10

4.3 Visual Layout Diagram 10

4.4 Interpreting the diagrams 11

4.5 Components & How they work during run-time 11

5 Testing **12**

6 Project Timeline **12**

6.1 User Interface 12

6.2 Configuration File Generator 13

6.3 File Transfer Protocol System 13

6.4 User Datagram Protocol System 13

6.5 Current Progress 14

7 Future Work **14**

8 Conclusion **14**

9 Glossary **15**

10 Appendix A: Development Environment and Toolchain **15**

10.1 Hardware 15

10.2 Toolchain 16

10.3 Setup 16

10.3.1 Downloading and Installing Visual Studio Community 2017 17

10.3.2 Downloading and Installing QtCreator: 18

10.3.3 Cloning the Repository off GitHub: 20

10.3.4 Running the program for the first time: 22

10.4 Production Cycle 23

1 Introduction

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

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

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

- ➢ How the drone configuration is handled, this could result in an error causing the drone to crash while losing all gathered data and potentially damaging valuable hardware.

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

- ➢ Only requiring one program without losing any functionality.
- ➢ Decreasing startup time to under a minute(on average) which saves minutes each time our client wants to start gathering data.
- ➢ Better managing configuration files to ensure data is uploaded correctly.

2 Process Overview

Before diving into more detail about our project, we will first walk through a few things about our process in regards to this project that helped us along the way.
The first step that we took in our process was to figure out what the key features were that we need to implement. We then took these requirements and grouped them into sections so that we could finish all related pieces of code at or around the same time. Version control as well as bug reporting was all handled via GitHub. We only had two code specific development roles when working on this project. Nathaniel handled the front-end while Sam handled the back-end. Keller and Eric both helped on each side of the project and did not have code specific roles.
All of the code that has been written for this project was written within QtCreator as it is the main development environment for all operating systems.
To make sure the development of the project went smoothly, there were a few things that were established at the start of the development cycle. The first is that we established a weekly meeting time that everyone could make so that we could go over things that got done during the week as well as things that still needed to get done. Overall, the goal of these meetings was to get everyone together and touch base so everyone was aware of what was going on. If there were any big decisions to be made we would go with a ¾ majority vote on which option to take. If we were split and could not get out of it, we would reach out to our mentor to get their input in the matter. We also discussed that all documents will be maintained via google docs so that progress can be seen as well as allowing for a group to work on a paper if needed.

3 Requirements

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.

3.1 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 led 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.

3.2 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 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. 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 set up 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 on 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 an experienced system user must be able to do this without any outside help.

4 Architecture and Implementation

To better understand our code structure, we will first discuss which files we have uniquely added and the files which we have modified.

Files we have added

1. qftp(.cpp, &.h)
2. myudp(.cpp & .h)
3. FtpDialog(.cpp & .h)

Files which we have modified

1. QGCApplication.cc
2. qgroundcontrol.pro
3. test.qml
4. MainToolbar.qml

5. MainRootWindow.qml

To distinguish our code in the changed files we have put comments above the changed code that look like this:

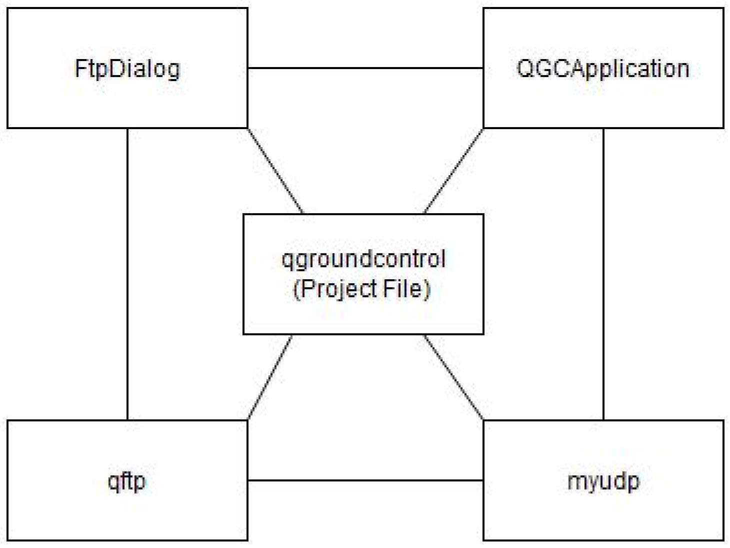
Above: --------UAVRT EDIT--------

With 8 hyphens on each side, to easily locate our changes we recommend using ctrl + f and typing the comment header as seen above.

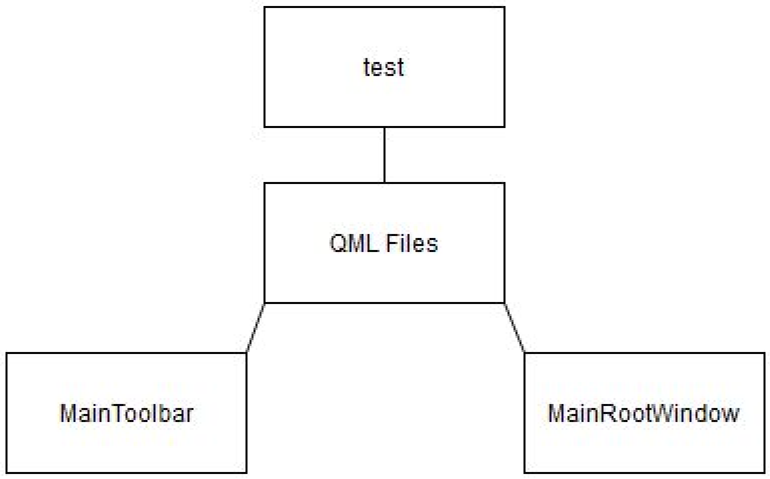
4.1 System Diagrams

For the sake of simplicity, we will diagram our system using the file names but will not reference the file type. To make it clear however .cpp & .cc files contain C++ code, .h files are headers for .cpp & .cc files, .qml files are code for appearance written in the language QML., and the .pro file is a Qt project file. We will also only show the associations that are related to the files that we have changed or made. Finally, we will present this data in two diagrams, first the code files diagram which includes files that end with .cpp, .cc, .h, and .pro. the second diagram is for the visual layout and contains files ending in .qml

4.2 Code Files Diagram



4.3 Visual Layout Diagram



4.4 Interpreting the diagrams

Where a line is drawn the file is associated in some way to the end point's file. Normally this is through a header's inclusion in that file and some sort of usage of the functions. For the QML files they aren't associated with each other, so we added a blanket category to provide structure to the diagram.

4.5 Components & How they work during run-time

Here is a list of components based on their UI representation with a general explanation on how they work during run-time. We also discuss in which files these components have the majority of their code and the nature of their relationships with other files

- Config File Generation: this component handles generating, saving and loading flight settings files. This section operates mainly in the test.qml file as it is written in JavaScript which qml files support inside them. It also communicates with the FTP management component to send flight data to the companion computer.
- FTP management: This component forges and manages the FTP connection between the software and the companion computer. This system allows for files to be uploaded to and downloaded from the companion computer. This component mainly exists in the FtpDialog(.cpp & .h)files and interacts with the qftp library to perform the needed functions during runtime.
- UDP Terminal: This component manages listening for heartbeat messages from the drone as well as sending start/stop commands for the software defined radio via UDP. This system is mainly contained in the myudp(.cpp & .h) files and interacts with the test.qml during run-time to show output in the user interface.

5 Testing

We will be heavily relying on Unit Testing, Integration Testing, as well as Usability Testing to ensure that we develop a bug-free application that our client can rely on. Our client has often stressed that this application needs to not just work, but be functional and reliable. So we know how important this is to him and therefore important to us. For our unit testing we will be utilizing QT Test to test our units. This library is extremely helpful with testing QT functions and since our project is written in QT it was an obvious fit. Our rationale for not testing the units in the user interface is that the units don't provide useful boundaries and the tests would be based upon our visual assent instead of some measurable metric. The units we are testing fall into 3 major categories: The FTP (File Transfer Protocol) Module, UDP (User Datagram Protocol) terminal management, and The Configuration Data Generation/Handling Module. For integration testing, the key components we decided to address were the UDP messages being sent to the terminal in the User Interface, the correctness of files being sent from config file module to the FTP module, and the Network communications of the FTP module. For usability testing, our software is currently being developed for use by our client. Therefore he is the current user of the software and we have kept the changes we have made very small to keep the user interface as close to his old MATLAB program as possible to not cause any problems. He has stated that the user interface can be put on the backburner as he was more interested in the software working correctly rather than looking nice.

6 Project Timeline

The project timeline can be broken up into four distinct categories, each separated based on which part of the project was currently being developed.

6.1 User Interface

The very first thing we worked on for the project was being able to generate the configuration file based on user input from the user interface. However, before we could start we needed to figure out how to modify QGroundControl. The start of this section began by our team trying to figure out how to modify the user interface of QGroundControl. The first big breakthrough with this was when we discovered that within the mainrootwindow.qml (for linux) and maintoolbar.qml (for windows and mac) files contained the code for creating the tabs for swapping between screens. So with a quick modification to these files we were able to implement our own tab where the entirety of our work takes place. Once we found out how to make our tab, the rest of the user interface was easy. Once we replicated the user interface from Dr. Shafer’s MATLAB program, we began working on the configuration file generator.

6.2 Configuration File Generator

With the user interface already developed, the configuration file generator was easy. Originally we thought that we would have to implement a c++ class and have it interact with the qml file we made. However, we quickly realized that JavaScript and QML work quite nicely together. With this breakthrough we were able to use JavaScript to implement the configuration file generator and importer. Once we finished this then we started to work on the File Transfer Protocol system.

6.3 File Transfer Protocol System

Originally, we figured this part of the project would be fairly easy to figure out. We had assumed that there would be a readily available library that had the functionality we needed. However this was not the case. The QT framework did not have an active File Transfer Protocol library that had the functionality that our client requested. We spent a bit of time online looking for c++ File Transfer Protocol libraries that we could use yet none either worked with our program or had the functionality we needed. Eventually we discovered an archived Qt File Transfer Protocol library that had everything we needed. Once we found the library we had to figure out how to get it correctly implemented within the system. Once we had it implemented we needed to figure out how to connect the c++ code with the qml document that was our user interface. This was fairly difficult as documentation online was either not well written or varied widely. We quickly discovered that there were multiple ways to connect qml with c++ and we had to figure out which was the correct way to get it working with our software. We eventually found the correct way to connect them and we were able to get the File Transfer Protocol system up and running after some debugging when testing with the live system.

6.4 User Datagram Protocol System

After completion of the File Transfer Protocol system we began progress on the User Datagram Protocol System. Thankfully, setting up this system followed many of the steps and problems we had gone through when connecting the File Transfer Protocol system. Upon full implementation, we were able to get this system up and running under the localhost system, but upon trying to transfer it to working on a live system using the Drone we have run into the problem of no longer getting any output.

6.5 Current Progress

As of writing this document, we are working on debugging the User Data Protocol system as we are having difficulties getting it working with the live system after testing and confirming the functionality is working on a local system.

7 Future Work

If we were able to continue development of this product, we would next focus on these important features

Making the project into an application so less effort is required to build and run the project as well as adding instructions on how to do this conversion for our client

Cleaning up the UI and changing the colors to better match the theme. In order to make a more seamless integration into QGroundControl(QGC) we would need to spend some time color matching and UI work to make our tab seem like it is “supposed” to exist within QGC regularly. Certainly not required for the product but a nice touch nonetheless.

Adding a unique logo for our tab such as the DASL logo. Being able to quickly ascertain which tab represents the special functions we have implemented is a very important part of user experience for us. We were focused on developing a robust backend in this phase but UI development would definitely be a part of the next development cycle.

Changing the UDP section to instead utilize 915MHz custom mavlink messages to further expand the range that messages from the drone can be received and to be able to use start/stop commands in a larger radius.

Automatically downloading data upon flight completion. This streamlines our clients process even further and allows for quick deployments and clean-ups.

Developing live data visualizations on the flight map based on UDP messages received. We think this would provide for a really useful user interface as well as elevate the usefulness of the overall product by allowing for better data visualization. This would be a development cycle all its own as there are a lot of moving parts and would provide a challenge to many teams.

8 Conclusion

To summarize our project, our client, Dr. Michael Shafer of the DASL, is using UAVs to track VHF tags on animals to determine behavior patterns. UAVs reduce the risk of injury and time of collection, but the system can still be improved. Our client’s problem with his workflow involved both a slow start up time for the MATLAB software and the need to utilize two separate programs. Our solution involved remaking the functionality of the MATLAB software into the new flight planner, QGroundControl. This new system will reduce the time spent for setup and flight. Some key features of the implementation are

- A single program encapsulating all needed functionality
- Quick start up time
- A streamlined configuration file generation system with useful extra features

We are confident that our product will streamline our client's work process and save him a decent amount of time to boot. We don't necessarily envision a broader impact from our software alone but together with Our Client's software we imagine that this will revolutionize the way animals are tracked via VHF tags. Overall as a team RAMPART was able to effectively manage and develop the requested product and we are proud of ourselves. The Capstone Class team really helped us achieve our goals by providing the necessary resources to excel. Special thanks to Dr. Eck Doerry, Our wonderful team mentor Mahsa Keshavarz and Our awesome client Dr. Michael Shafer.

9 Glossary

- FTP -- File Transfer Protocol, network protocol used for the transfer of files over a computer network.
- UDP -- User Datagram Protocol, a networking protocol used often for streaming data.
- QGC -- QGroundControl, the flight planner that is the base of our program.
- VHF tag -- Very High Frequency Tag, a type of radio telemetry for use in tracking wildlife.

10 Appendix A: Development Environment and Toolchain

10.1 Hardware

The software was developed using two different hardware environments:

- Ubuntu 19.04 with an Intel i5-8300H CPU with 8GB of ram

We do not believe that there are any minimum specs required for this project, however a better computer will be able to write and build changes faster.

10.2 Toolchain

There are a number of specific tools that are needed to make changes from this software. A number of them are not set by us, but by the developers of QGroundControl.

- Operating System
  - Windows Vista or higher
  - macOS v10.11 or higher
  - Ubuntu 64 bit, gcc compiler
- Visual Studio 2017 (Windows Only, 64bit)
  - This is required as the Visual Studio 2017 compiler is needed for the project if compiling on a Windows operating system.
- Qt Version 5.12.6
  - This is the only build that QGroundControl documents say to use and therefore this specific version must be used even though it is older than current releases.
  - Qt Components
    - There are a number of specific Qt components one must also get while downloading to ensure all parts function correctly. All of these can be found as options when installing Qt.
    - Windows: MSVC 2017 64 bit
    - MacOS: macOS
    - Linux: Desktop gcc 64-bit
    - All:
      - Qt Charts
      - Android ARMv7 (if you wish to build Android)
- Additional Packages
  - There are a few additional packages that must also be installed along with the Qt software.
  - Ubuntu: `sudo apt-get install speech-dispatcher libudev-dev libsdl2-dev`
  - Fedora: `sudo dnf install speech-dispatcher SDL2-devel SDL2 systemd-devel`
  - Arch Linux: `pacman -Sy speech-dispatcher`
  - Windows: USB Driver to connect to Pixhawk/PX4Flow/3DR Radio

10.3 Setup

We will first discuss how to install all the necessary components on the Windows operating system.

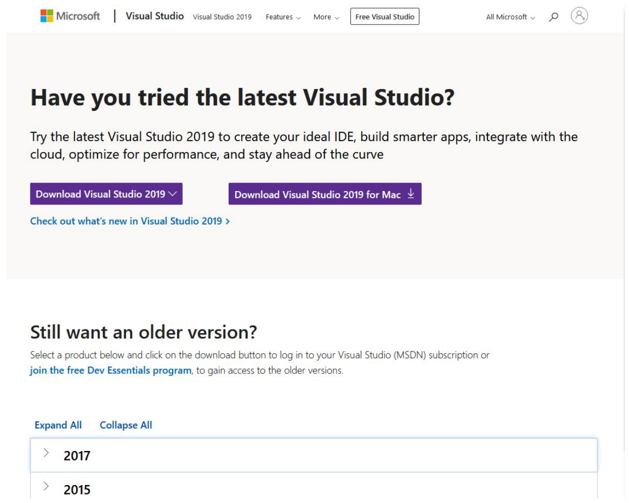
There are 3 steps to installing the product for use:

1. Downloading & Installing Visual Studio 2017
2. Downloading & Installing QtCreator

3. Cloning the RAMPART UAV-RT branch of QGroundControl

10.3.1 Downloading and Installing Visual Studio Community 2017

-First follow this link: [Visual Studio Community 2017]() which will bring you to a page that looks like this:



Microsoft | Visual Studio Visual Studio 2019 Features More Free Visual Studio All Microsoft

Have you tried the latest Visual Studio?

Try the latest Visual Studio 2019 to create your ideal IDE, build smarter apps, integrate with the cloud, optimize for performance, and stay ahead of the curve

Download Visual Studio 2019 | Download Visual Studio 2019 for Mac

Check out what's new in Visual Studio 2019 >

Still want an older version?

Select a product below and click on the download button to log in to your Visual Studio (MSDN) subscription or join the free Dev Essentials program, to gain access to the older versions.

Expand All Collapse All

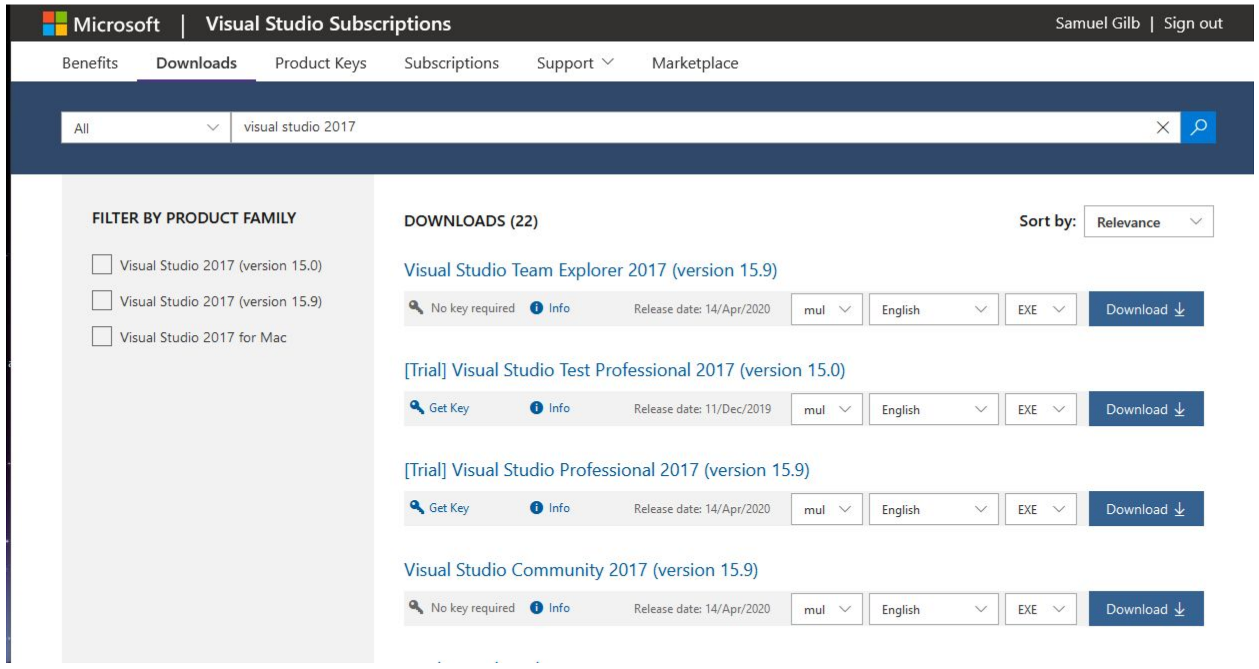
> 2017

> 2015

-If you do not have a Dev Essentials account, click the hyperlink titled “join the free Dev Essentials program”. After creating an account return to the page by clicking the hyperlink in this document again. If you already have an account ignore this and continue to the next step.

-Next expand the 2017 tab by clicking the arrow to the left of 2017 and then click the button labeled “download” in the expanded tab.

-This will bring you to a page that looks like this:



Microsoft | Visual Studio Subscriptions Samuel Gilb | Sign out

Benefits Downloads Product Keys Subscriptions Support Marketplace

All | visual studio 2017

FILTER BY PRODUCT FAMILY

- Visual Studio 2017 (version 15.0)
- Visual Studio 2017 (version 15.9)
- Visual Studio 2017 for Mac

DOWNLOADS (22) Sort by: Relevance

Visual Studio Team Explorer 2017 (version 15.9)
No key required | Info | Release date: 14/Apr/2020 | mul | English | EXE | Download

[Trial] Visual Studio Test Professional 2017 (version 15.0)
Get Key | Info | Release date: 11/Dec/2019 | mul | English | EXE | Download

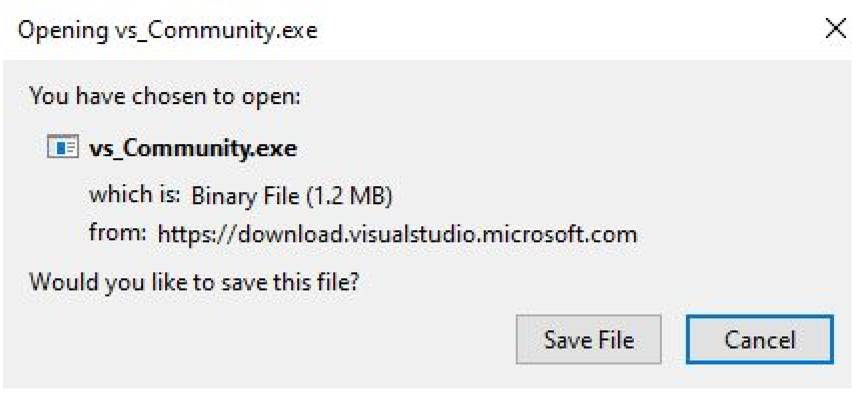
[Trial] Visual Studio Professional 2017 (version 15.9)
Get Key | Info | Release date: 14/Apr/2020 | mul | English | EXE | Download

Visual Studio Community 2017 (version 15.9)
No key required | Info | Release date: 14/Apr/2020 | mul | English | EXE | Download

-In the Filter by product family area click the box for “Visual Studio 2017 (version 15.9)”.

-After that in the downloads area locate and click the download button for the item titled “Visual Studio Community 2017 (version 15.9)”.

-At this point a pop-up should appear that looks like this.



Opening vs_Community.exe

You have chosen to open:

vs_Community.exe

which is: Binary File (1.2 MB)

from: https://download.visualstudio.microsoft.com

Would you like to save this file?

-Click “Save File”.

-Locate the file in your downloads folder or wherever you download files to and click on the file titled vs_Community.exe to run the installer.

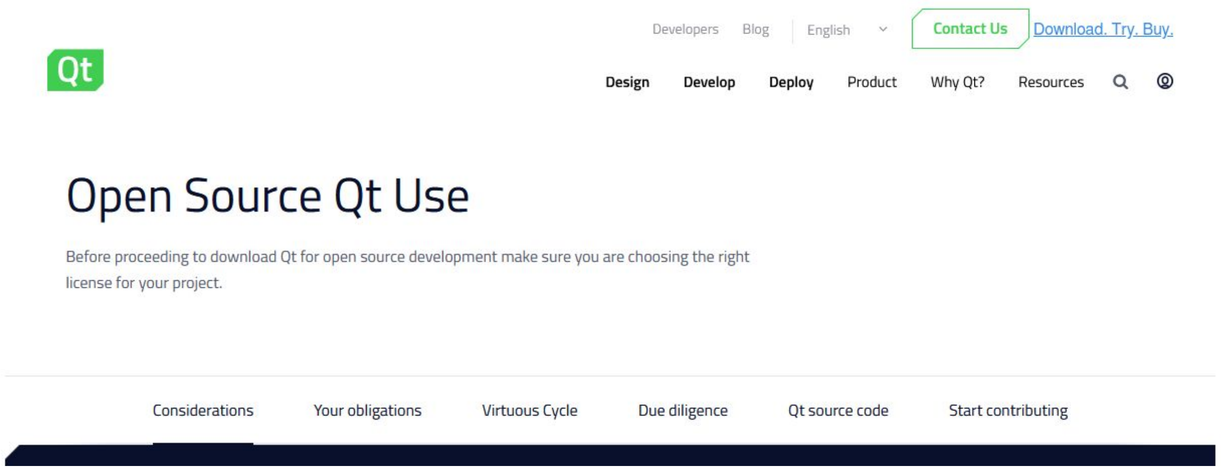
-Follow the instructions in the wizard and wait for the installation to complete.

-Congratulations you have installed the necessary version of Visual Studio 2017.

10.3.2 Downloading and Installing QtCreator:

-To start, open this link: [Qt Online Installer Download] in your web browser.

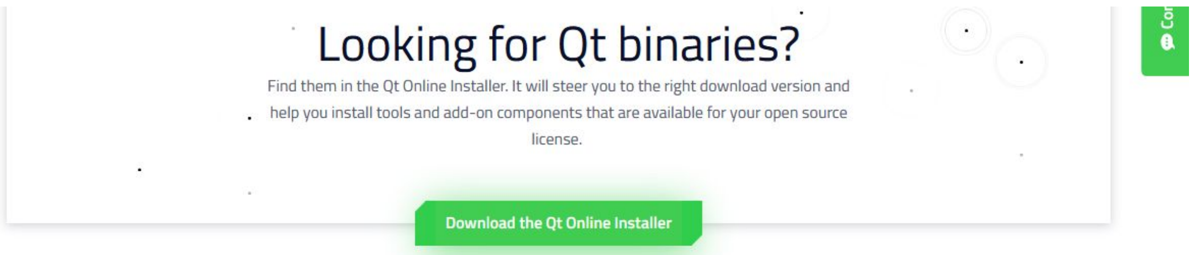
-You will be directed to a page that looks like this:



Open Source Qt Use

Before proceeding to download Qt for open source development make sure you are choosing the right license for your project.

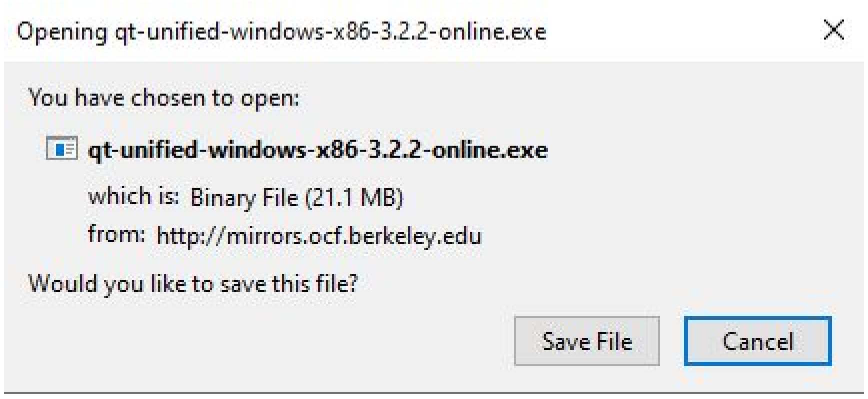
-Scroll to the bottom and click on the button which reads “Download the Qt Online Installer” as seen below:


Looking for Qt binaries?

Find them in the Qt Online Installer. It will steer you to the right download version and help you install tools and add-on components that are available for your open source license.

-This will take you to another page with the title "Install Qt". Make sure the operating system detected is windows and click the "Download" button.

-You will see a window that looks like this, click the "Save File" button.


Opening qt-unified-windows-x86-3.2.2-online.exe

You have chosen to open:

qt-unified-windows-x86-3.2.2-online.exe

which is: Binary File (21.1 MB)
from: http://mirrors.ocf.berkeley.edu

Would you like to save this file?

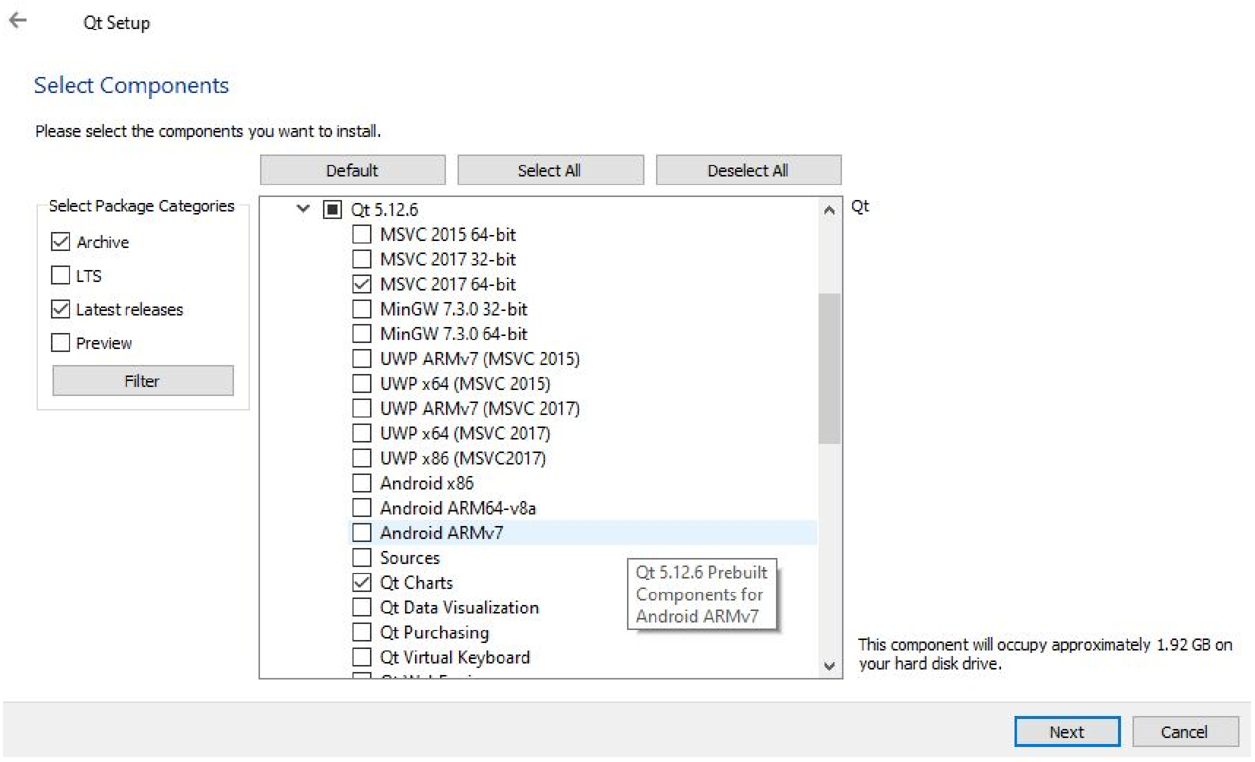
-Locate the file in your download directory or wherever you downloaded the file to and click it to begin the Qt Installation process.

-You will be asked to enter Qt Account credentials either enter your preexisting credentials or to make a free Qt account.

-After this progress through the next steps in the wizard and then set up the installation directory to where you want on your device.

-Upon reaching the select components dialog in the leftmost box click the square next to "archive" and then click the "filter" button, this will prompt another meta information download.

-After this download is completed click the arrow next to the Qt name and click the arrow next to "Qt 5.12.6" to expand the installation options, you will see a screen like the one as shown below. Make sure to check the MSVC 2017 64-bit box and the Qt Charts box 3.


Qt Setup
Select Components
Please select the components you want to install.
Default
Select All
Deselect All
Select Package Categories
Filter
Qt
Next
Cancel

☑ Archive
☐ LTS
☑ Latest releases
☐ Preview

- ■ Qt 5.12.6
  - ☐ MSVC 2015 64-bit
  - ☐ MSVC 2017 32-bit
  - ☑ MSVC 2017 64-bit
  - ☐ MinGW 7.3.0 32-bit
  - ☐ MinGW 7.3.0 64-bit
  - ☐ UWP ARMv7 (MSVC 2015)
  - ☐ UWP x64 (MSVC 2015)
  - ☐ UWP ARMv7 (MSVC 2017)
  - ☐ UWP x64 (MSVC 2017)
  - ☐ UWP x86 (MSVC2017)
  - ☐ Android x86
  - ☐ Android ARM64-v8a
  - ☐ Android ARMv7
  - ☐ Sources
  - ☑ Qt Charts
  - ☐ Qt Data Visualization
  - ☐ Qt Purchasing
  - ☐ Qt Virtual Keyboard

Qt 5.12.6 Prebuilt Components for Android ARMv7

This component will occupy approximately 1.92 GB on your hard disk drive.

-Proceed through the rest of the installation following the steps in the wizard.

-Congratulations you have installed the necessary version of Qt Creator and all required plugins.

10.3.3 Cloning the Repository off GitHub:

-To start, navigate to the GitHub repository right here. Keep this page open for either cloning case as you will need it for both.

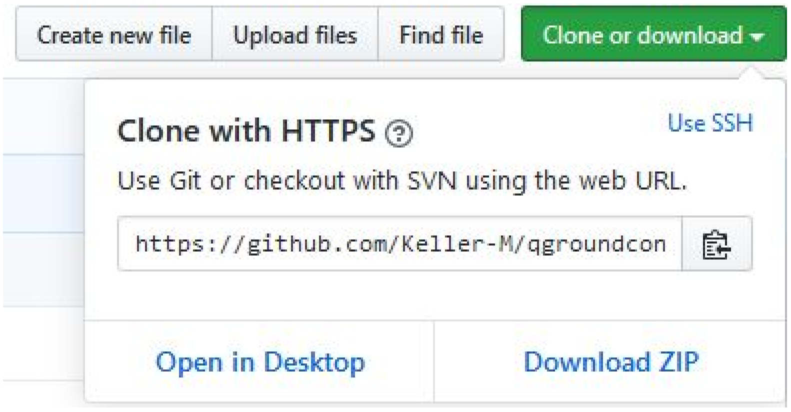
If you have a version of Git for Windows feel free to utilize the To Clone Using Command Line interface section otherwise scroll down to the “To Clone with GitHub Desktop “ section instead.

To Clone using Command Line Interface (CLI)

-First open your command line interface of choice.

-Next navigate to the desired location of the cloned directory.

-At this point head back to the repository website and click the green button “Clone or download” and copy the URL by either clicking the clipboard button or by highlighting the URL and copying it. An example of the screen you will see can be seen below.



Clone with HTTPS

Use Git or checkout with SVN using the web URL.

```
https://github.com/Keller-M/qgroundcon
```

-In the Command Line Interface type the following commands (they are in italics):

git pull (the URL copied goes here) uavrt

at this point the repository should download.

cd ./uavrt/

git submodule update

-After running those command, the repository should be good to go.

-Congratulations you have downloaded the repository and are ready to build and run the product.

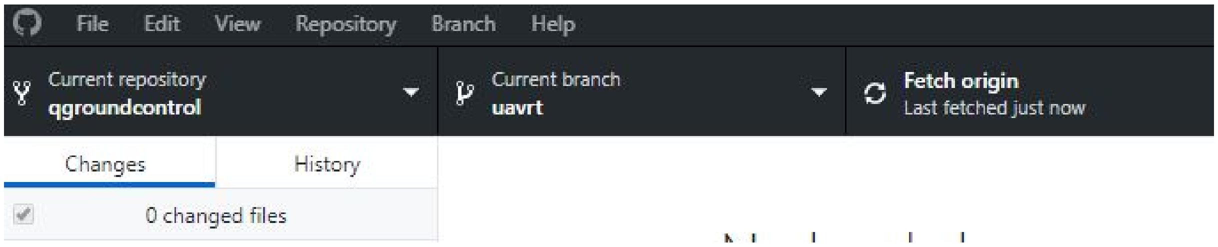
To Clone with GitHub Desktop

-First if you do not have GitHub desktop follow this link to download it https://desktop.github.com/

-Next open the webpage with the repository and click the clone or download button and then click on the “Open in Desktop” button on the bottom left. The window can be seen above in the command line interface set of instructions.

-This should open a dialog in the GitHub desktop app, make sure the local path is set to where you want to store the repository and click clone. This process may take a couple of minutes to complete.

-After the download has completed make sure to change the branch to uavrt (if you are having trouble locating the branch type “uav-rt” in the filter area). Then press the fetch origin button to make sure the files have downloaded properly. The fetch origin button looks like the image below.



Current repository: qgroundcontrol

Current branch: uavrt

Fetch origin — Last fetched just now

0 changed files

-Congratulations you have downloaded the repository once the fetch origin bar says “last fetched now” and are ready to build and run the product

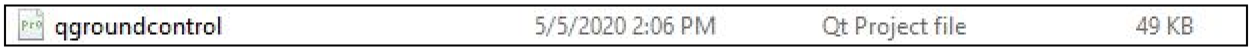
10.3.4 Running the program for the first time:

-After installing all the components as instructed above you should be able to build and run the project.

-First start by opening Qt Creator.

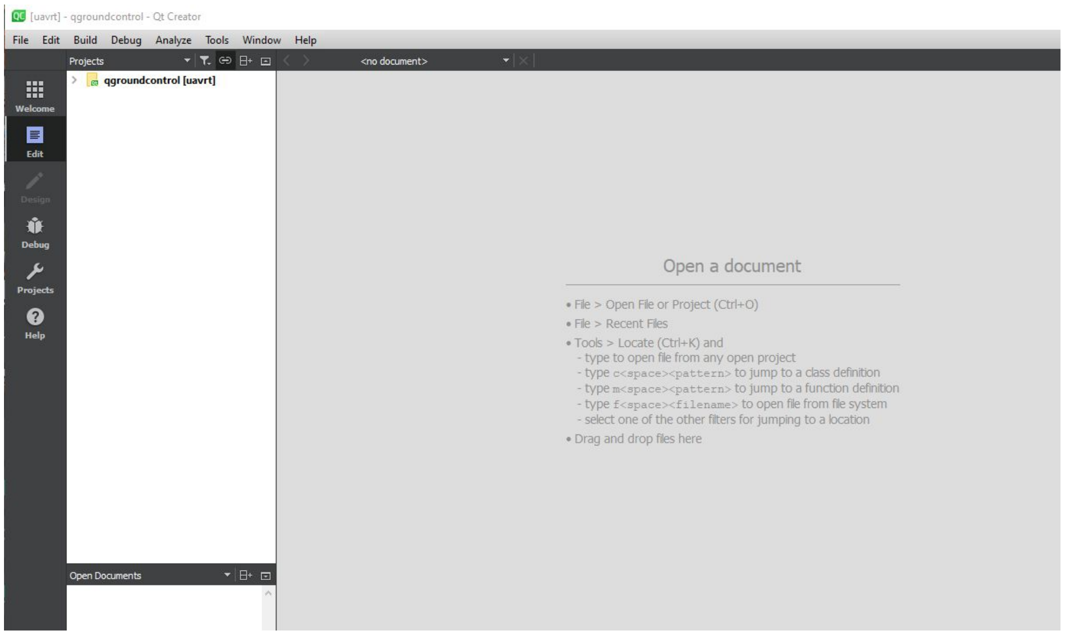
-Then on the projects tab click the open button.

-The navigate to the folder where QGroundControl is stored. Navigate into the qgroundcontrol folder and locate the file named qgroundcontrol which should be of the type “Qt Project File”. The file should look like the image below.



qgroundcontrol	5/5/2020 2:06 PM	Qt Project file	49 KB

-Click this file and then click open, the project should open to a screen like the image on the next page.

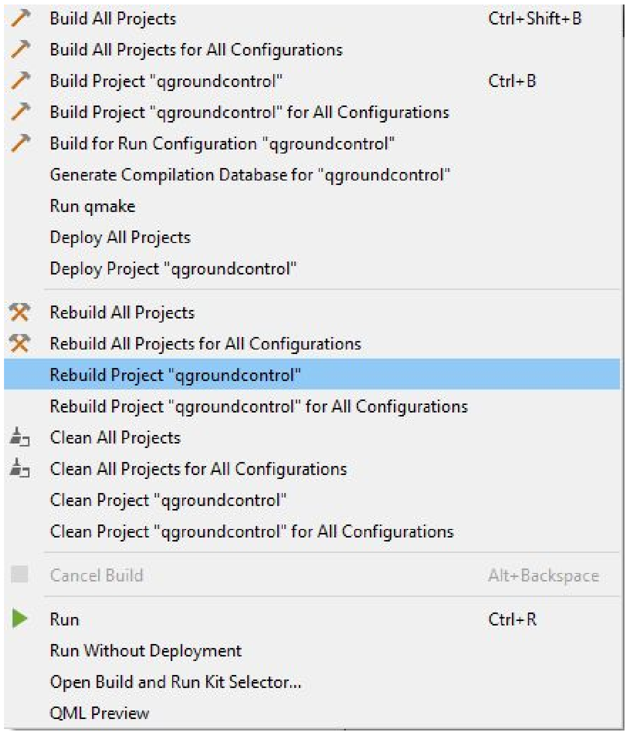


Open a document

- File > Open File or Project (Ctrl+O)
- File > Recent Files
- Tools > Locate (Ctrl+K) and
  - type to open file from any open project
  - type `c<space><pattern>` to jump to a class definition
  - type `m<space><pattern>` to jump to a function definition
  - type `f<space><filename>` to open file from file system
  - select one of the other filters for jumping to a location
- Drag and drop files here

If you see an error stating something similar to[1] “Project ERROR: MAVLink folder does not exist at 'libs/mavlink/include/mavlink/v2.0'! Run 'git submodule init && git submodule update' on the command line.” Open the build tab on the right and select “Rebuild project qgroundcontrol” option as seen in the below image highlighted in blue.

[1] An error in this case is a red octagon, a warning which is a yellow triangle is not applicable and the program has been built correctly. Our program doesn’t introduce any new warnings and there should be 17 warnings in total on a successful build.


Build All Projects Ctrl+Shift+B
Build All Projects for All Configurations
Build Project "qgroundcontrol" Ctrl+B
Build Project "qgroundcontrol" for All Configurations
Build for Run Configuration "qgroundcontrol"
Generate Compilation Database for "qgroundcontrol"
Run qmake
Deploy All Projects
Deploy Project "qgroundcontrol"
Rebuild All Projects
Rebuild All Projects for All Configurations
Rebuild Project "qgroundcontrol"
Rebuild Project "qgroundcontrol" for All Configurations
Clean All Projects
Clean All Projects for All Configurations
Clean Project "qgroundcontrol"
Clean Project "qgroundcontrol" for All Configurations
Cancel Build Alt+Backspace
Run Ctrl+R
Run Without Deployment
Open Build and Run Kit Selector...
QML Preview

-The build should take roughly 5-15 minutes to complete.

-After the initial build completes select run on the build menu and the program should run.

If any of this was confusing or you are still having problems setting up the development environment, QGroundControl has their own developer guide with installation steps which can be found here: https://dev.qgroundcontrol.com/en/.

10.4 Production Cycle

We will now walk through an example on how to modify the user interface to show how modifications to the program can be accomplished.

- The first step will be to launch QtCreator, this is the main environment that will be used for all development.
- You will now be greeted with the main menu of QtCreater, navigate to the top left where it says File, click the button and then hit Open File or Project.
- You will now need to navigate to the folder in which you cloned QGroundControl. Once you have navigated there click on the .pro file and then hit open.
- You are now greeted with the main screen that you will be seeing while developing any changes. On the left is the file tree which allows you to open and view all files which will show up on the right side.

- If you click within the file tree on the left you are able to CTRL + F to find any specific files, for this example we will look for the “test.qml” file and double click it to open it on the right side of the application.
- We can also CTRL + F within the code view to find specific pieces of code, for this example we will look for “tagFreqLabel”.
- Right under where the label is you will see qsTr(“Tag Frequency (MHz)”), for the sake of this demo, feel free to change the text inside the qsTr parenthesis to whatever you want.
- Once you have finished your changes, on the bottom left of the application there is a hammer, click this hammer to build the application.
- This may take some time, the first build is the longest out of all of them. The bottom right side of the screen will have status messages, as well as the Compile Output tab on the bottom. You will know the build is complete when the status bar on the bottom right is completely green.
- You can launch the application with the play button (without the bug symbol) above the build button.
- When the application launches, click the circle tab icon to swap to the modified tab, and on the configuration module on the left, one of the labels will have changed to the text you input earlier.