QGroundControl: Ground Station for Land, Water, and Air Autonomous Vehicles

Jessica Hall[1], Bryant Mairs[2], Gabriel Elkaim[2]
[1]*University of Nevada Reno,* [2]*University of California Santa Cruz*
University of California Santa Cruz Autonomous Systems Laboratory

A problem that arises from developing autonomous and remote vehicles is the question of how to communicate with the vehicle. The user needs to be able to send it instructions, gather data from it, or control it in an emergency. This is the role of ground stations like QGroundControl (QGC) which is designed to be fast and capable of running on a variety of platforms and supports many vehicles simultaneously whether they travel by land, water, or air.

This project focused on improving QGC from both a usability and code development standpoint. This included fixing outstanding issues, improving existing unit tests, and extending documentation. Due to the open-source nature of QGC many of its users had filed bugs which outstripped developer support. The existing unit testing framework, which ensures reliable operation of the ground station, was improved. This is of critical importance as these vessels can cost several thousands of dollars. Documentation was added to help make it easier for developers to continue to improve QGC. Additionally documentation was extended for the users who will utilize it to pilot or interact with the vessels in the field.

I. Introduction

QGC is a ground station for use with autonomous vehicles. It is designed to be portable so it can be used from a laptop, desktop, a tablet etc. QGC was first created as a project called PIXHAWK by Lorenz Meier and a group of students at Eidgenössische Technische Hochschule Zürich (ETH Zürich) in 2009 [Pixhawk]. It was originally designed for micro air vehicles (MAVs)**,** then released as an open source project called QGroundControl and has since been worked on to be integrated with land and water vehicles as well as air vehicles [Pixhawk].

Some features of QGC include support for multiple simultaneous land, air or marine vehicles. It provides the user with a pilot view and in addition shows a live 2D map that displays the position of the vehicle. Telemetry provides the user with information on the behavior of the vehicle. QGC supports the use of Google Earth which is used for viewing three dimensional maps. It is customizable by users even by non-developers. The purpose of QGC is to be portable between platforms and devices and easy to use.

II. Methods

GitHub is where the QGC repository is stored. GitHub keeps track of all the changes made to the repository. GitHub is useful because multiple people can be working on their own version of QGC without affecting the main repository. When the developers have completed what they are working on, they put in a pull request which is a request to merge the new code into the main repository.

Valgrind is a program that checks for memory leaks. This was used to fix memory leaks that were found when the unit tests were run.

QGC is developed and run with the development software, Qt.

The problems addressed during the summer were lack of documentation, unit tests, and minor bugs within the code. The lack of documentation makes it difficult for a developer or user to know exactly what the program is doing or what the code is doing. It is important for the unit tests to be working because the code should be tested thoroughly so that the reliability of the code can be proven. The

program should be free of bugs which inhibit performance or usability.

III. Results

Comments were lacking within the code which made it hard to understand what the code was doing. Comments provide developers with a quick and easy way to understand what the code is doing without having to read through every line of code. Comments were added to some of the functions within UAS.cc to provide an overview of what the functions do.

The unit tests for QGC only tested the code in the file UAS.cc. The unit tests at first did not compile. Once they were fixed so that they compiled, they would not run correctly due to a segmentation fault. The segmentation fault was caused by a list of colors. This list was would give a new color to every MAV that was created. A static variable was used as the index to get the next color in the list. There are 19 colors in this list and when 19 MAVs had been created and destroyed, the end of the list had been reached and the variable was used to attempt to access memory past the end of the list causing a segmentation fault. The solution to this problem was to fix the code so that the variable would loop back to the beginning of the list. It is now possible to create more than 19 MAVs though there will be more than one MAV with the same color.

Once the unit tests were compiling and running, the unit tests were fixed so that they all passed. Valgrind was used to find and fix memory leaks that occurred when the unit tests were run.

Documentation on the layout of the repository was added to the README within the root directory. This was done before the root directory was reorganized which had many files and folders which it made it hard to find things. One example of disorganization was that the libraries were scattered in different directories. The repository was reorganized so that all the libraries are in one directory and all source files are in another directory and other files that did not belong in the library or the source directories were put in the appropriate directories. Code which defined paths to certain files had to be changed to reflect the changes made to the repository. In addition to rearranging the files, unnecessary files were deleted. The rearrangement made it easier to find files.

. Doxygen is a tool used to generate documentation using comments that were written in a certain format. Instructions on how to use Doxygen were added to a README file in the doc directory within the repository.

There were many issues reported by users and developers and a few minor ones were addressed and fixed. Some were suggestions on removing a little bit of code or changing a small piece of code. Others were suggestions on how to change the graphical user interface (GUI) or which files to remove because they were unnecessary. These changes were made and since they did not affect the program adversely they were pulled into the repository on GitHub.

IV. Conclusions

The documentation added will help users and developers to use QGC. The now properly working unit tests will ensure that changes will not adversely affect the code. The new layout makes it easier to find files. The issues that were fixed have improved the performance of QGC.

V. Future Work

More documentation needs to be added to the files. The documentation of the repository needs to be redone to reflect the reorganization of the repository. The unit tests cover most but not all of the functions within UAS.cc and more unit tests need to be added to test all of the code within QGC. There are more memory leaks that need to be fixed. There are many other issues reported by users and developers that need to be addressed.

VI. Acknowledgements

I would like to thank Professor Gabriel Elkaim for the opportunity to work in the Autonomous Systems Lab and Bryant Mairs for helping me when I needed it and for answering my questions. I would also like to thank Colt Hangen and Matthew Guthaus for their work in making SURF-IT a wonderful program and to the National Science Foundation for funding the program.

VII. References

Pixhawk (website). Retrieved from https://pixhawk.ethz.ch/overview.