Graphic User Interface with Autopilot Features for UAVs

By
Tomas McMonigal

A THESIS

submitted to

Oregon State University

Honors College

in partial fulfillment of
the requirements for the
degree of

Honors Baccalaureate of Science in Electrical & Computer Engineering
(Honors Associate)

Presented September 3, 2020
Commencement July 2020

AN ABSTRACT OF THE THESIS OF

Tomas McMonigal for the degree of Honors Baccalaureate of Science in Electrical & Computer Engineering presented on September 3, 2020. Title: Graphic User Interface with Autopilot Features for UAVs

Abstract approved:

Robin Hess

The purpose of this project was to provide a tool for the Aerial Team of the Robotics Club with an application to control a UAV for automated missions - the Aerial Team's focus is automation of UAVs. This application lays out the foundations to allow future members of the Aerial Team to build on and customize for specific automated missions. UAV university competitions consist on building and automating UAVs for a wide variety of missions. This is why I designed this application to allow multiple processes to connect to one control center, in a client-server type fashion. There are three software components that act as clients: the graphic user interface, the path planning algorithm, and the computer vision algorithm. With this project I am providing the Aerial Team with a control center, as well as a graphic user interface. The path planning and computer vision algorithms are left to other members of the team to develop to be able to deploy a UAV for autonomous missions. The control center I've designed is multi-threaded program that allows other programs to connect to it using sockets. This makes it easy for future additions to be incorporated into the system. The research for this project consisted in finding the the most advanced technology available for educational purposes and integrating it into a practical system for real-world solutions. All of the code used for this project is open-source. Please note that the words UAV and drone are used interchangeably throughout this document.

Key Words: drone, unmanned aerial vehicle (UAV), autopilot, graphic user interface (GUI), application programming interface (API)

Corresponding e-mail address: tomas.mcmonigal@gmail.com

Graphic User Interface with Autopilot Features for UAVs

By
Tomas McMonigal

A THESIS

submitted to

Oregon State University

Honors College

in partial fulfillment of
the requirements for the
degree of

Honors Baccalaureate of Science in Electrical & Computer Engineering
(Honors Associate)

Presented September 3, 2020
Commencement July 2020

Honors Baccalaureate of Science in Electrical & Computer Engineering project of Tomas McMonigal presented on September 3, 2020.

APPROVED:

Robin Hess, Mentor, representing EECS

Eduardo Cotilla-Sanchez, Committee Member, representing EECS

Committee Member Name, Committee Member, representing Committee Member Department

---

Toni Doolen, Dean, Oregon State University Honors College

I understand that my project will become part of the permanent collection of Oregon State University Honors College. My signature below authorizes release of my project to any reader upon request.

Tomas McMonigal, Author

‘

Contents

1 Introduction

At the start of my junior year in the Electrical & Computer Engineering program I became a member of the Robotics Club at Oregon State University. The minute I walked through the doors of Graff Hall for the first time I became fascinated looking at the projects that students were working on. I was greeted by the president of the the Club who gave me a tour of the building and enthusiastically spoke about what each of the different teams were working on. As he talked about the Aerial Team he showed me the UAVs hanging on the wall, that was the moment I knew what team I wanted to be a part of. I have been interested in drones ever since they became popular in the last decade but had not had the chance to fly one. Therefore, I thought this was my opportunity to learn how to build and fly drones.

I joined the team shortly after and started attending the weekly meetings. At the time, the Aerial Team had almost completely changed leadership as the seniors had graduate the term before so the Team was focusing its effort to come together with a clear objective. The projects and achievements left behind by former members gave us the inspiration we needed to work together as a team and build something we could be proud of. There was a lot of work to do if we wanted to get a fair shot at an international university competition for UAVs, but that was the goal. What the team needed most help with was the autopilot and computer vision software, which together would command the drone to perform according to a competition's mission requirements. I shared my interest in automation with the team and told them that I would be happy to help with the autopilot software. This is how I arrived to my thesis project; to build a platform for the Aerial Team that could be used to build on and customize for autonomous missions. Although there is open-source software available for automated UAV missions, such as QGroundControl [7], we needed something that we could use with our own path planning and computer vision algorithms.

To be able to compete at a university level competition on UAVs, we needed customizable software for specific mission requirements. This is the reason that led me to develop an application that could serve as a ground station system for automated UAVs. Therefore, the basic requirements were to establish communication with the drone to receive telemetry data and send commands to it. Additionally, a graphic user interface was needed to be able to visualize the drone's current position throughout the course of a mission, and to display relevant telemetry data. I thought it would be nice to also have some basic autopilot functionality in the GUI to command the drone to arm, disarm, takeoff to a set altitude, land and return home. This system had to be designed in such a way that would allow other software components such as the computer vision and path planning algorithms to easily connect to it.

2 Software Architecture & Background

At the time I began working on this project I had recently gained some experience using Qt to create a graphic user interface for my Junior Design project - a USB oscilloscope. Qt [6] is a free and open source platform for developing graphic user interfaces. One of best features is that it is cross-platform, allowing the user to cross-compile applications to run on a variety of software and hardware platforms such as Linux, Windows, macOS, Android and embedded systems. It allows to develop a reliable lightweight applications with native capabilities and speed. Additionally, I needed something that would provide the means to control the drone, acting as a back end for the graphic user interface. After researching the different technologies available for drone automation I decided to go with MAVSDK [10], which is a set of libraries written in C++ that provide a high-level API to MAVLink [9]. MAVLink is a lightweight communication protocol for UAV systems and ground stations. It covers the basics needed to safely operate a drone and allows for 3rd party customization. For the choice of firmware and hardware I decided to go with the PX4 [11], an open-source autopilot system oriented toward inexpensive autonomous aircraft. It provides a flexible set of tools for drone developers to create tailored solutions for drone applications. PX4 also provides a standard to deliver drone hardware support and software stack, allowing an ecosystem to build and maintain hardware and software in a scalable way. On a side note, PX4, MAVSDK, and MAVLink are part of a non-profit organization, called Dronecode [8], which is administered by the Linux Foundation to foster the use of open source software on flying vehicles. To communicate between the ground station and my drone I decided to use MAVProxy [14], a UAV ground station software package for MAVLink based systems. MAVProxy is a command-line based "developer" ground station software which can be complemented with other ground station components as it was done for this project. With all this in mind, I set out to order the components I would need to build my own drone.

3 Hardware

In terms of hardware, the flight controller is the brain of the drone and its most important component. I decided to go with the Pixhawk 4 [13], the most advanced flight controller for the PX4 autopilot. The flight controller has a gyroscope, accelerometer, magnetometer and barometer. It also receives input from the GPS, and the Radio Control Receiver, as well as transmits data through a Two-way Radio Telemetry antenna. Using these inputs and sensors the PX4 determines the output signal for each motor. All the components are powered by the power distribution board which is connected to a Lithium Polymer battery. The power distribution board also powers the electronic speed controllers which themselves power and regulate the speed of the motors using the input signal from the flight controller.

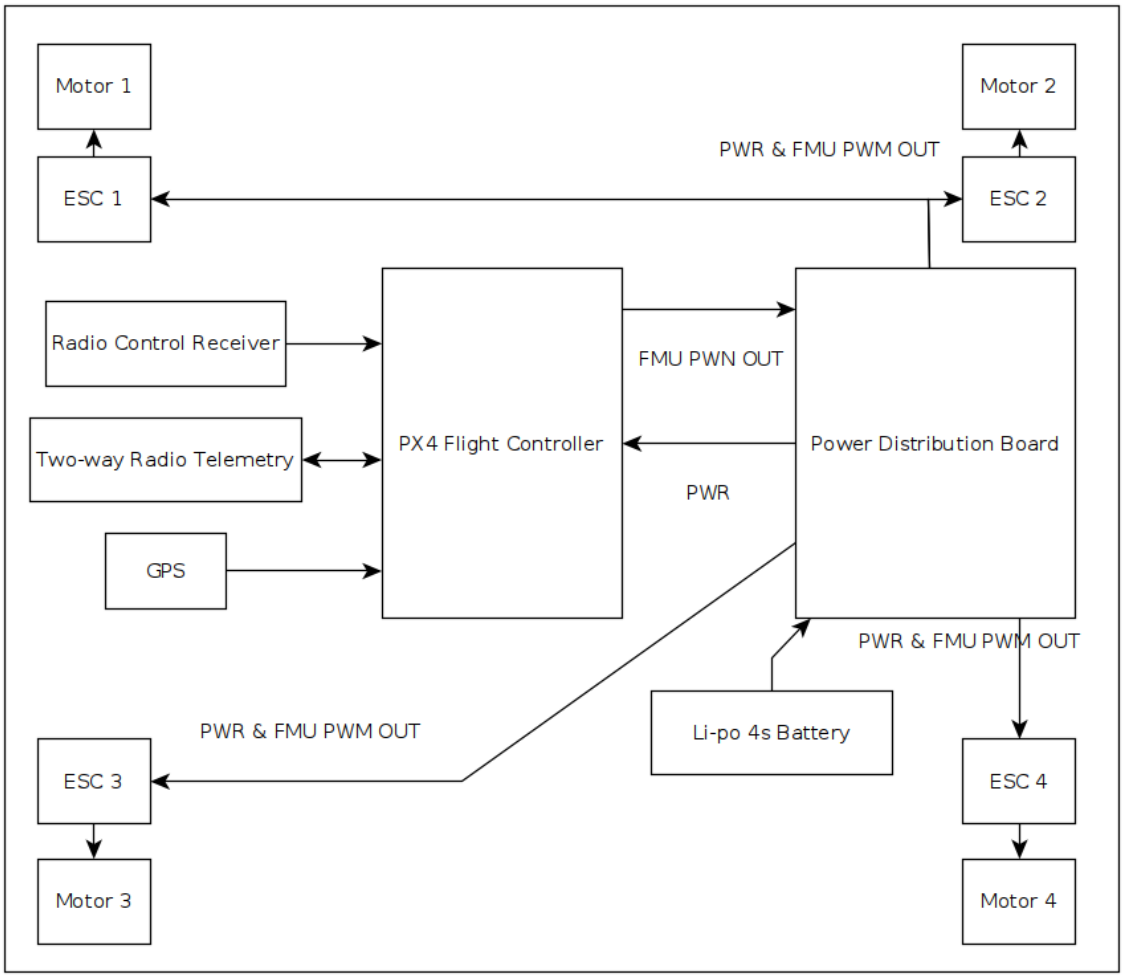

Figure 1: Hardware Block Diagram

.

4 Software

Once I had finished building my drone and was able to fly it manually I began working on a Qt application which I based on one of the examples of the Qt Packages [4]. These examples are intended to serve as tutorials to help new users get started with Qt development. After learning the basics of how the graphic user interface works in an example called Plane Spotter [2], I started designing a control center, which would act as a server and an intermediary between the drone and the graphic user interface, the path planing and computer vision algorithms. I wrote the control center in Python to make use of Python wrapper for the MAVSDK library, which allows to control the drone autonomously, and is more user friendly that directly using the MAVSDK core

which is written in C++. From the start of this project my main goal was to provide the Aerial Team with a system for them to build on and customize for future use. As I mentioned in the introduction, for any university competition, the autonomous system would require a computer vision and path planning programs aside from a graphic user interface. Therefore, I needed to implement one block to take input from these three different programs and communicate with the drone. This is what I am referring to as the control center. The structure of such system is as follows: the front end consists of the Qt graphic user interface, and the back end consists of the MAVSDK libraries to control the drone and the MAVProxy software to communicate with the drone, using the MAVLink protocol. It is worth mentioning that Qt has Python bindings as well, which would have made it possible to have had the server and graphic user interface run as one process. However, having separate software blocks makes a simpler and more fail-proof system architecture. Figure 2 shows the high-level software block diagram which includes all blocks and communication protocols on the ground station as described above.

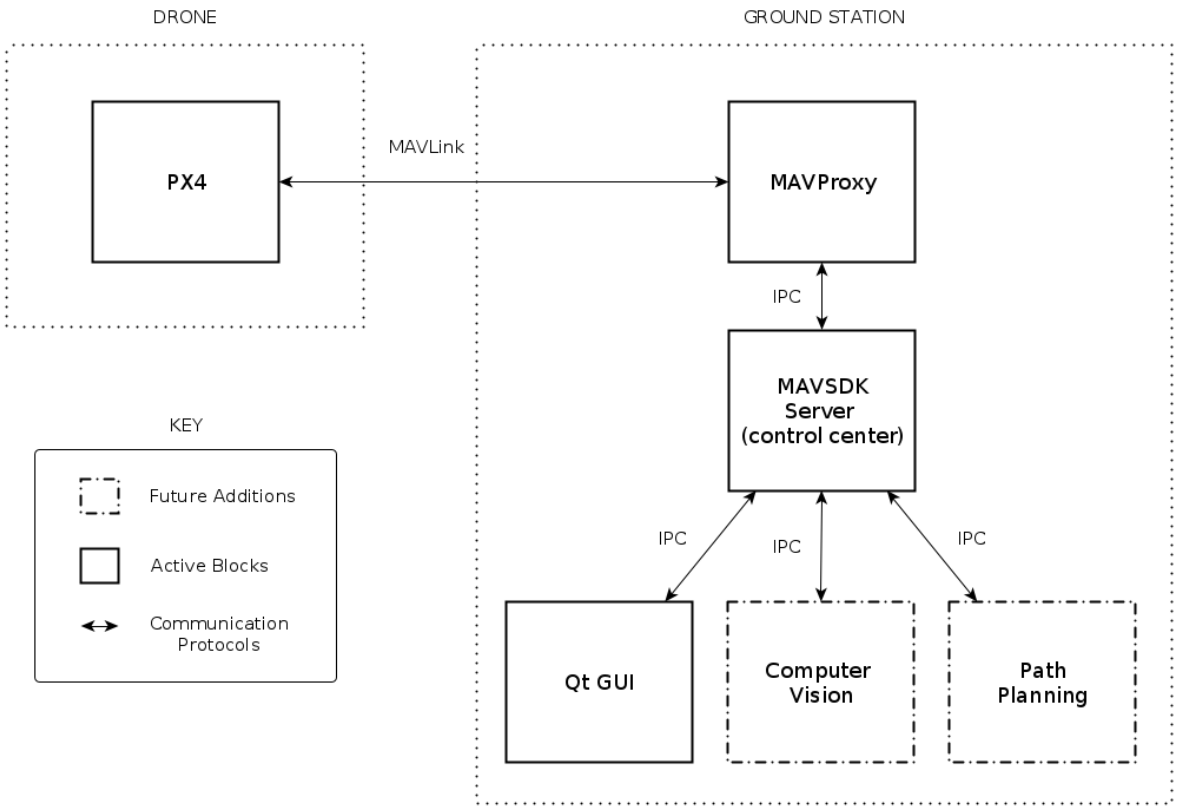

Figure 2: Software Block Diagram

4.1 Graphic User Interface

To get started with my Qt application I went through the examples in Qt Package provided by the Qt Company and found one called Plane Spotter, mentioned above. This example demonstrates the integration of location and positioning data types into QML [3]. QML stands for Qt Modeling Language which is a user interface markup language that uses JavaScript to handle the imperative aspects. The tutorial examples provided by the Qt Company are dual-licensed under commercial and open source licenses [1]. Plane Spotter uses a map and database of coordinates, as well as built in connections between the front end (QML) and the back end (C++). These connections are called Signals and Slots [5]. The reason I chose this example to build on is because one of the most important requirements was to be able to track the drone's current location and display it on the map. Secondly, this application needed the telemetry data to be continuously updated and displayed for the user. Telemetry data includes latitude, longitude, altitude, flight mode, armed/disarmed status, and more. I also thought it would be nice to add some autopilot features to allow the user to takeoff, land, return to home, and set the altitude when using the position-hold flight mode. For each of these user features there is a signal that is triggered in the front end, and consequently, a slot that is activated in the back-end. From a high-level perspective the flow of application goes as follows. On one hand, there is a worker thread that is constantly requesting and receiving telemetry data from the drone. The use of a thread was necessary to avoid the graphic user interface from lagging in case of delays in communication. This avoids delays in sending an action command to the drone, which could be critical and must be sent immediately. The request for the telemetry data is sent once every second to avoid taking up too much bandwidth. This frequency seemed a reasonable time for telemetry data updates but can be modified. When the application is launched, a socket connection is established between the Qt application and the server. Then a thread is created which runs an infinite loop with the sole purpose of requesting and receiving the telemetry data. Once all of the data is received a signal is triggered in the back end, for which a slot is activated in the front end responsible of updating the display. On the other hand, whenever the user uses one of the autopilot features and activates a signal in the front end, the corresponding slot in the back end sends the a request to the server. Figure 3 shows the high-level work flow of the Qt application.

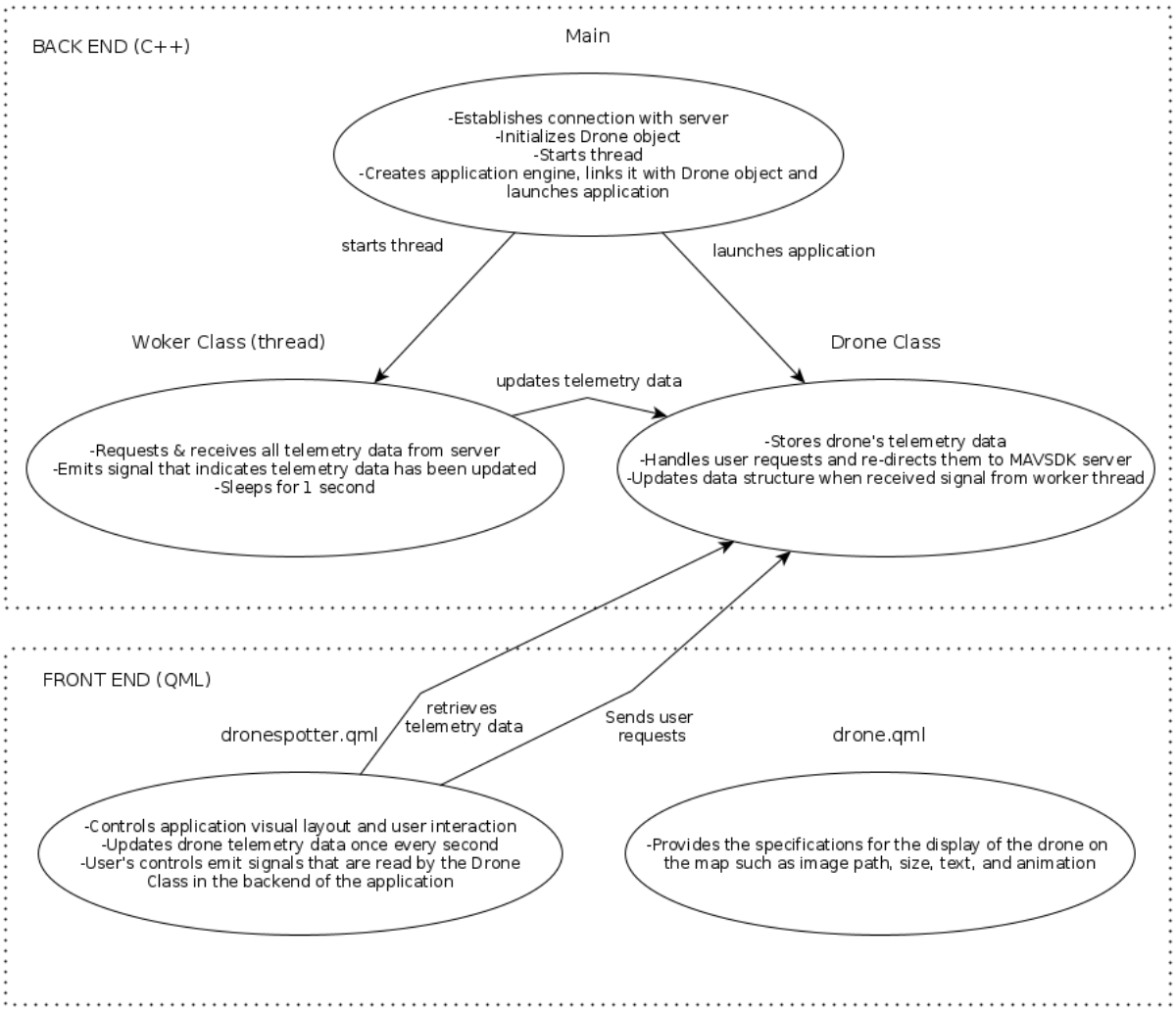

Figure 3: Qt flowchart

4.2 MAVSDK Server

The MAVSDK server, which I have previously referred to as the control center, uses the Python wrapper for the MAVSDK library, whose core is written in C++. This program also uses the Asyncio Python library [12], to write concurrent code. The Asyncio library is intended for I/O-bound and high-level structured network code. The way the server works by first establishing connection with the drone through a third program, called MAVProxy. This connection is established using sockets. When MAVProxy is launched, it uses an antenna attached to a USB port to connect to the drone using the MAVLink communication protocol. Then, it waits until the drone has an accurate global position estimate, and sets its current position as home and its takeoff altitude to five meters. Once confirmation has been received from the UAV,

the program is ready to accept incoming connections from other processes that wish to communicate with the drone. When a client wishes to communicate with the drone and opens a socket connection with the server, a subprocess is created which runs the code written in the echo_server function. This functions reads and decodes each request, determines what is being requested, and forwards the request to the drone. The program may also request the server to send an action command to the drone such as takeoff, land, and return-to-home. The socket will remain open and active as long as data is being received from the application with intervals no longer than 10 seconds, which can be easily changed. The simple structure of the server allows it to run reliably and close any socket connections that is hung or has been closed from the client side.

5 How to Run the Application

This application was developed on Linux and is designed to run on Linux. Although one of the major advantages of Qt is that it is cross-platform, it has not yet been cross-compiled and tested on any other operating systems.

5.1 Dependencies

This is a list of all the dependencies needed for this application to work. For more information on how to download and install these dependencies can be found under References.

1. MAVProxy
2. Qt 5
3. Python 3.6+
4. MAVSDK-Python
5. Asyncio-Python

5.2 Instructions

Before you begin, make sure the drone is on and that the telemetry antenna is connected to the USB port. Type "dmesg" and based on the output determine which USB port the antenna has been attached to, such as ttyUSB0, ttyUSB1, etc. Next, follow the instructions below.

1. Open a terminal window and run

```
$ mavproxy.py --master=/dev/<USB Port> --out=127.0.0.1:14555
```

2. Open a terminal window in the application directory and run

```
$ python MAVSDK\_server.py
```

3. (Optional) To test that the server is up and running you can open another terminal window and type

```
$ telnet localhost 5000
```

   this will connect you to the server as a client. You may then retrieve drone telemetry data by typing "altitude", "latitude", "longitude", etc. (to view the full list of options look at the echo_server function in MAVSDK_server.py).

4. Launch the Qt application from Qt Creator by clicking on the play button. Alternatively you may also launch the qt application without Qt creator by opening the terminal on the project directory and running the following commands.

```
$ qmake -project
$ qmake
$ make
$./Hydra
```

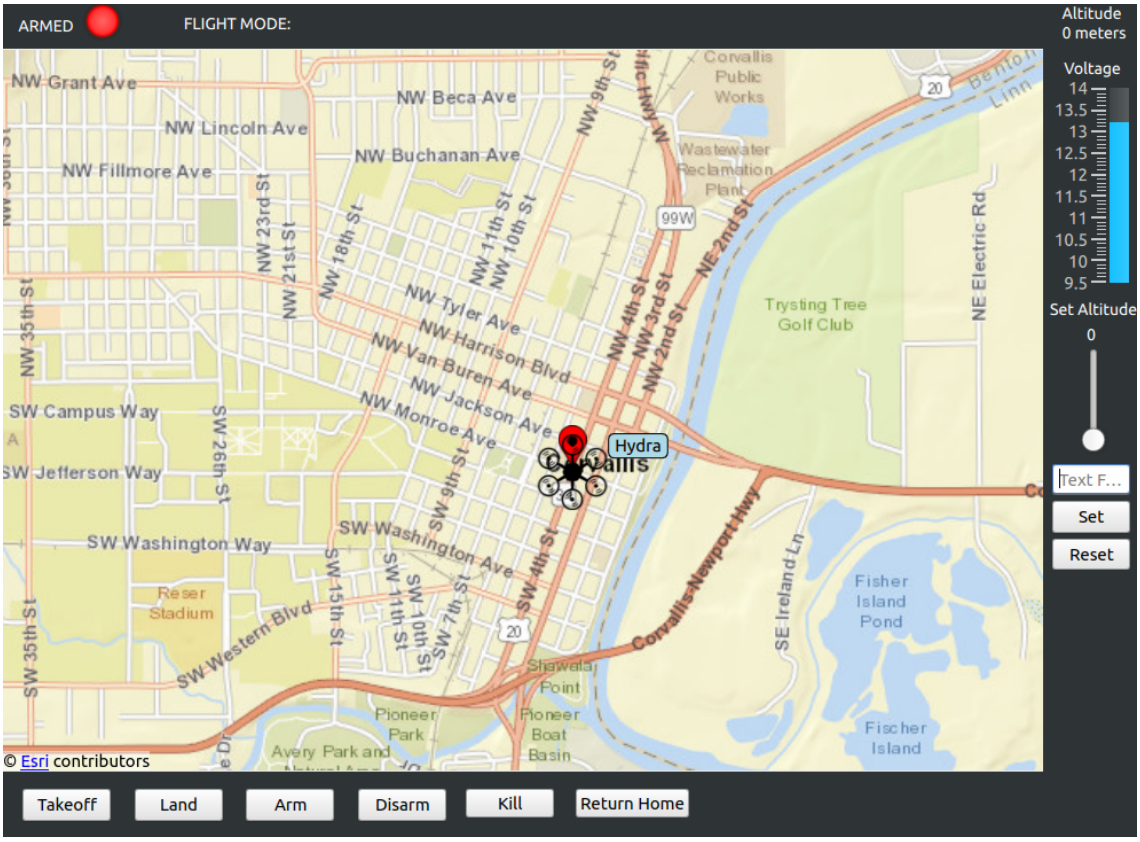

Figure 4: Screenshot of the graphic user interface

6 Conclusion

This project has been tested and works reliably while meeting all the Aerial Team's requirements. This link a video that show the system being tested with a drone (`https://media.oregonstate.edu/media/t/1_mabvin8g`). All the autopilot features on the graphic user interface were tested and performed accordingly. These tests consisted on how accurate the UAV responded to the commands of the user through the graphic user interface. Furthermore, the telemetry data displayed on the GUI was at all times accurate and updated every second, as it intended it to be. This refers to the altitude, gps, battery voltage, and armed/disarmed status. My hope is that it will serve future members of the Aerial Team develop projects, win competitions, and most importantly, learn how turn their ideas into reality.

7 References

[1] The Qt Company. *Licensing*. 2020. URL: https://www.qt.io/licensing/ (visited on 08/28/2020).

[2] The Qt Company. *Plane Spotter (QML)*. 2020. URL: https://doc.qt.io/qt-5/qtlocation-planespotter-example.html (visited on 08/28/2020).

[3] The Qt Company. *QML Applications*. 2020. URL: https://doc.qt.io/qt-5/qmlapplications.html#what-is-qml (visited on 08/28/2020).

[4] The Qt Company. *Qt Examples And Tutorials*. 2020. URL: https://doc.qt.io/qt-5/qtexamplesandtutorials.html (visited on 08/28/2020).

[5] The Qt Company. *Signals & Slots*. 2020. URL: https://doc.qt.io/qt-5/signalsandslots.html (visited on 08/28/2020).

[6] The Qt Company. *WHY Qt? - The Future Is Written with Qt*. 2020. URL: https://www.qt.io/why-qt (visited on 08/28/2020).

[7] Dronecode. *QCG - QGroundControl - Drone Control*. 2020. URL: http://qgroundcontrol.com (visited on 08/28/2020).

[8] Dronecode Foundation. *Dronecode- Building a sustainable open source ecosystem for critical Drone components*. 2020. URL: https://www.dronecode.org/ (visited on 08/28/2020).

[9] Dronecode Foundation. *MAVLink Developer Guide*. 2020. URL: http://mavlink.io/en (visited on 08/28/2020).

[10] Dronecode Foundation. *MAVSDK (develop)*. 2020. URL: https://mavsdk.mavlink.io/develop/en/index.html (visited on 08/28/2020).

[11] Dronecode Foundation. *Open Source Autopilot for Drones - PX4 Autopilot*. 2020. URL: https://px4.io/ (visited on 08/28/2020).

[12] Python Software Foundation. *asyncio - Asynchronous I/O*. 2020. URL: https://docs.python.org/3/library/asyncio.html (visited on 08/28/2020).

[13] Holybro. *Pixhawk 4 - The most advanced development kit for the PX4 autopilot*. 2020. URL: http://www.holybro.com/product/pixhawk-4 (visited on 08/28/2020).

[14] ArduPilot Dev Team. *MAVProxy - MAVProxy documentation*. 2020. URL: https://ardupilot.org/mavproxy (visited on 08/28/2020).

8 Code

Link to github: `https://github.com/OSURoboticsClub/aerial_GUI`

8.1 Qt Application

8.1.1 main.cpp

```
/*****************************************************
 * Filename: main.cpp
 * Author: Tomas McMonigal
 * Date: 08/10/2020
 * ***************************************************/

#include "worker.h"
#define BUFFER_SIZE 1000
#define ANIMATION_DURATION 500

/*****************************************************
* Function: get_in_addr
* Description: returns sockaddr, works for both IPv4 or Ipv6
* Taken from Beej's guide
*****************************************************/
void *get_in_addr(struct sockaddr *sa)
{
    if (sa->sa_family == AF_INET) {
        return &(((struct sockaddr_in*)sa)->sin_addr);
    }

    return &(((struct sockaddr_in6*)sa)->sin6_addr);
}

/*****************************************************
 * Class: Drone
 * Description: Holds all telemetry for the drone.
 * When a Drone object is instanciated a Worker thread
 * is created to continuously retrieve telemetry data.
 * This class also handles any requests from the user
 * graphic interface to the drone.
 * ***************************************************/
class Drone: public QObject
{
    Q_OBJECT
    QThread workerThread;
    Q_PROPERTY(QGeoCoordinate position READ position WRITE
   setPosition NOTIFY positionChanged)
    Q_PROPERTY(QGeoCoordinate from READ from WRITE setFrom NOTIFY
   fromChanged)
```

```
39      Q_PROPERTY(QGeoCoordinate to READ to WRITE setTo NOTIFY
    toChanged)
40
41  public:
42      Drone()
43      {
44          easingCurve.setType(QEasingCurve::InOutQuad);
45          easingCurve.setPeriod(ANIMATION_DURATION);
46          this->currentAltitude = 0;
47          this->batteryVolt = 13.2;
48          this->is_armed = false;
49
50      }
51      ~Drone(){
52          workerThread.quit();
53          workerThread.wait();
54          close(socketFD);
55      }
56
57      void start_client_thread(){
58          this->worker = new Worker;
59          worker->moveToThread(&workerThread);
60          connect(&workerThread, &QThread::finished, worker, &QObject
    ::deleteLater);
61          connect(this, &Drone::operate, worker, &Worker::doWork);
62          // remember update position has an animation gotta get rid
    of
63          connect(worker, &Worker::resultReady, this, &Drone::
    updatedCoordinatesSlot);
64          workerThread.start();
65      }
66
67      void setSocketFD(int socketFD){
68          this->socketFD = socketFD;
69      }
70
71      void setFrom(const QGeoCoordinate& from)
72      {
73          fromCoordinate = from;
74      }
75
76      QGeoCoordinate from() const
77      {
78          return fromCoordinate;
79      }
80
81      void setTo(const QGeoCoordinate& to)
82      {
83          toCoordinate = to;
84      }
85
```

```
86      QGeoCoordinate to() const
87      {
88          return toCoordinate;
89      }
90
91      void setPosition(const QGeoCoordinate &c) {
92          if (currentPosition == c)
93              return;
94
95          currentPosition = c;
96          emit positionChanged();
97      }
98
99      QGeoCoordinate position() const
100     {
101         return currentPosition;
102     }
103     Q_INVOKABLE double altitude() const{
104         return currentAltitude;
105     }
106
107     Q_INVOKABLE double batteryVoltage() const{
108         return batteryVolt;
109     }
110
111     Q_INVOKABLE bool isFlying() const {
112         return timer.isActive();
113     }
114     Q_INVOKABLE void arm(){
115         send(socketFD, "arm", 3, 0);
116     }
117     Q_INVOKABLE void disarm(){
118         send(socketFD, "disarm", 6, 0);
119     }
120     Q_INVOKABLE void kill(){
121         send(socketFD, "kill", 4, 0);
122     }
123     Q_INVOKABLE bool isArmed() const{
124         return is_armed;
125     }
126     Q_INVOKABLE QString flightMode() const{
127         return flight_mode;
128     }
129     Q_INVOKABLE void takeoff(){
130         send(socketFD, "takeoff", 7, 0);
131     }
132     Q_INVOKABLE void land(){
133         send(socketFD, "land", 4, 0);
134     }
135     Q_INVOKABLE void return_home(){
136         send(socketFD, "return", 6, 0);
```

```
137     }
138
139 public slots:
140     void startFlight()
141     {
142         if (timer.isActive())
143             return;
144
145         startTime = QTime::currentTime();
146         finishTime = startTime.addMSecs(ANIMATION_DURATION);
147
148         timer.start(15, this);
149         emit departed();
150     }
151     void updatedCoordinatesSlot(const QGeoCoordinate newCoord,
    double newAltitude, double newBattery, int is_armed, QString
    flight_mod){
152         updateIsArmed(is_armed);
153         updateToCoordinate(newCoord);
154         updateAltitude(newAltitude);
155         updateBattery(newBattery);
156         updateFlightMode(flight_mod);
157         operate(socketFD);
158     }
159     void updateIsArmed(int is_armed){
160         if (is_armed == false){
161             this->is_armed = false;
162         }
163         else {
164             this->is_armed = true;
165         }
166     }
167     void updateToCoordinate(QGeoCoordinate newCoordinate){
168         toCoordinate = newCoordinate;
169     }
170     void updateFromCoordinate(){
171         fromCoordinate = toCoordinate;
172     }
173     void updateAltitude(double altitude){
174         currentAltitude = altitude;
175     }
176     void updateBattery(double battery){
177         batteryVolt = battery;
178     }
179     void updateFlightMode(QString flight_mod){
180         this->flight_mode = flight_mod;
181     }
182
183 signals:
184     void positionChanged();
185     void arrived();
```

```
186     void departed();
187     void toChanged();
188     void fromChanged();
189     // starts thread
190     void operate(int socketFD);
191 
192 private:
193     Worker *worker;
194     QString flight_mode;
195     int is_armed;
196     double currentAltitude;
197     double batteryVolt;
198     QGeoCoordinate currentPosition;
199     QGeoCoordinate fromCoordinate, toCoordinate;
200     QBasicTimer timer;
201     QTime startTime, finishTime;
202     QEasingCurve easingCurve;
203     int socketFD;
204 };
205 
206 /*****************************************************
207  * Function: main
208  * Description: Connects to the server, creates aplication
209  * engine and links,and launches application. Both the ip
210  * address, in this case localhost, and port number can be
211  * modified. The structure of the code is setup to handle
212  * ip address.
213  * ***************************************************/
214 int main(int argc, char *argv[])
215 {
216     QGuiApplication app(argc, argv);
217     Drone HydraDrone;
218 
219     // Connecting to server
220     int socketFD;
221     char buffer[BUFFER_SIZE];
222     char buffer2[BUFFER_SIZE];
223     memset(buffer, '\0', sizeof(buffer));
224     memset(buffer2, '\0', sizeof(buffer2));
225 
226     char hostname[] = "localhost";
227     // connects to port number 5000
228     char portNumberString[] = "5000";
229     char s[INET6_ADDRSTRLEN];
230     int status;
231     struct addrinfo hints;
232     struct addrinfo *servinfo, *p;
233     memset(&hints, 0, sizeof hints);
234     hints.ai_family = AF_INET; // IPv4
235     hints.ai_socktype = SOCK_STREAM; // fill in the IP for me
```

```
236     if ((status = getaddrinfo(hostname, portNumberString, &hints, &
    servinfo)) != 0){
237         fprintf(stderr, "getaddrinfo error: %s\n", gai_strerror(
    status));
238         exit(1);
239     }
240     // loop through all the results and connect to the first we can
241     for (p = servinfo; p != NULL; p = p->ai_next) {
242         // creates socket (taken from Beej's guide)
243         if ((socketFD = socket(p->ai_family, p->ai_socktype,
244                                p->ai_protocol)) == -1) {
245             perror("client: socket");
246             continue;
247         }
248         // establishes connection to server (taken form Beej's guide
    )
249         if (connect(socketFD, p->ai_addr, p->ai_addrlen) == -1) {
250             close(socketFD);
251             perror("client: connect");
252             continue;
253         }
254         break;
255     }
256     bool connection_status = true;
257     if (p == NULL) {
258         fprintf(stderr, "client: failed to connect\n");
259         connection_status = false;
260         //return 2; // uncomment to make application \
261         // quit if connection unsuccessful
262     }
263     if (connection_status == true){
264         // gets the IP address of the hostname and prints it
265         inet_ntop(p->ai_family, get_in_addr((struct sockaddr *)p->
    ai_addr), s, sizeof s);
266         printf("client: connecting to %s\n", s);
267         freeaddrinfo(servinfo);
268         send(socketFD, "latitude", 8, 0);
269         recv(socketFD, buffer, sizeof(buffer), 0);
270         printf("%s", buffer);
271         HydraDrone.start_client_thread(); // starts worker thread
272         HydraDrone.setSocketFD(socketFD);
273         HydraDrone.operate(socketFD);
274     }
275 
276     // creates application engine and links it to the Drone object
277     QQmlApplicationEngine engine;
278     engine.rootContext()->setContextProperty("HydraDrone", &
    HydraDrone);
279     engine.load(QUrl(QStringLiteral("qrc:/planespotter.qml")));
280 
281     // launches application
```

```
282     return app.exec();
283 }
284 
285 #include "main.moc"
```

8.1.2 worker.h

```
1  /*************************************************************
2   * Filename: worker.h
3   * Author: Tomas McMonigal
4   * Date 08/10/20
5  *************************************************************/
6  
7  #ifndef WORKER_H
8  #define WORKER_H
9  
10 #include <QGuiApplication>
11 #include <QQmlApplicationEngine>
12 #include <QQmlContext>
13 #include <QObject>
14 #include <QTime>
15 #include <QBasicTimer>
16 #include <QDebug>
17 #include <QEasingCurve>
18 #include <QGeoCoordinate>
19 #include <QtPositioning/private/qwebmercator_p.h>
20 #include <QPointF>
21 #include <QInputDialog>
22 #include <QtWidgets>
23 #include <iostream>
24 #include <iomanip>
25 #include <stdio.h>
26 #include <stdlib.h>
27 #include <unistd.h>
28 #include <string.h>
29 #include <sys/types.h>
30 #include <sys/socket.h>
31 #include <netinet/in.h>
32 #include <netdb.h>
33 #include <fcntl.h>
34 #include <stdbool.h>
35 #include <errno.h>
36 #include <netinet/in.h>
37 #include <arpa/inet.h>
38 
39 /*************************************************************
40  * Class: Worker
41  * Description: Thread that is created when a Drone object
42  * is instantiated. Retrieves telemetry data from the drone
43  * every one second.
44 *************************************************************/
```

```
class Worker : public QObject
{
    Q_OBJECT
public slots:
    void doWork(const int socketFD) {
        sleep(1);
        QGeoCoordinate newCoord;
        // ***************** Position Request ****************************
        qDebug() << "Thread: requesting current location from server";
        char buffer[100];
        memset(buffer, '\0', sizeof(buffer));
        // request latitude from drone
        send(socketFD, "latitude", 8, 0);
        recv(socketFD, buffer, sizeof(buffer), 0);
        double latitude = atof(buffer);
        memset(buffer, '\0', sizeof(buffer));
        // request longitude from drone
        send(socketFD, "longitude", 9, 0);
        recv(socketFD, buffer, sizeof(buffer), 0);
        double longitude = atof(buffer);
        QGeoCoordinate current_location(latitude, longitude);

        // ****************** Altitude Request ****************************
        // request altitude from drone
        send(socketFD, "altitude", 8, 0);
        recv(socketFD, buffer, sizeof(buffer), 0);
        // cuts string at 5th char
        buffer[5] = '\0';

        // ****************** Battery Status Request ************************
        double altitude = atof(buffer);
        send(socketFD, "battery", 7, 0);
        memset(buffer, '\0', sizeof(buffer));
        recv(socketFD, buffer, sizeof(buffer), 0);
        double battery = atof(buffer);

        // ****************** Getting Armed/Disarmed Status *****************
        send(socketFD, "is_armed", 8, 0);
        memset(buffer, '\0', sizeof(buffer));
        recv(socketFD, buffer, sizeof(buffer), 0);

        int is_armed;
        qDebug() << buffer;
        if (buffer[0] == 'F'){
            qDebug() << "disarmed";
            is_armed = 0;
```

```
91          }
92          else {
93              is_armed = 1;
94          }
95          qDebug() << "is armed:";
96          qDebug() << is_armed;
97
98          // ***************** Gets flight mode
    ****************************
99          send(socketFD, "flight_mode", 11, 0);
100         memset(buffer, '\0', sizeof(buffer));
101         recv(socketFD, buffer, sizeof(buffer), 0);
102         QString flight_mode = buffer;
103
104         // ****************** Sends flight mode to master
    ****************
105         qDebug() << "Thread: sending location to master";
106         emit resultReady(current_location, altitude, battery,
    is_armed, flight_mode);
107     }
108
109 signals:
110     void resultReady(const QGeoCoordinate &result, const double &alt
    , const double &batt, const int &is_armed, const QString &
    flight_mod);
111 };
112 #endif // WORKER_H
```

8.1.3 planespotter.qml

```
1  import QtQuick 2.4
2  import QtQuick.Window 2.2
3  import QtPositioning 5.5
4  import QtLocation 5.6
5  import QtQuick.Controls 1.6
6  import QtQuick.Extras 1.4
7  import QtWebEngine.Controls1Delegates 1.0
8  import QtQuick.Layouts 1.3
9  import Qt.labs.calendar 1.0
10 import QtGraphicalEffects 1.0
11 import QtQuick.Controls.Styles.Desktop 1.0
12 import QtQuick.Dialogs.qml 1.0
13 import QtTest 1.2
14 import QtQuick.Layouts 1.0
15 import QtQuick.Dialogs 1.2
16
17 ApplicationWindow {
18     id: window
19     width: 1000
20     height: 500
21     visible: true
```

```
22
23      property variant topLeftCorvallis: QtPositioning.coordinate
    (44.599480, -123.326150)
24      property variant bottomRightCorvallis: QtPositioning.coordinate
    (44.552963, -123.221276)
25      property variant viewOfCorvallis:
26          QtPositioning.rectangle(topLeftCorvallis,
    bottomRightCorvallis)
27      property variant corvallis: QtPositioning.coordinate(44.5646,
    -123.2620)
28      property variant albany: QtPositioning.coordinate(44.6365,
    -123.1059)
29
30      Rectangle {
31          id: rightRectangle
32          x: 916
33          width: 84
34          color: "#2e3436"
35          anchors.right: parent.right
36          anchors.bottom: parent.bottom
37          anchors.top: parent.top
38
39          Text {
40              id: element4
41              x: 4
42              width: 80
43              height: 16
44              color: "#ffffff"
45              text: qsTr("Set Altitude")
46              anchors.right: parent.right
47              anchors.rightMargin: 0
48              anchors.top: parent.top
49              anchors.topMargin: 259
50              horizontalAlignment: Text.AlignHCenter
51              font.pixelSize: 14
52          }
53
54          Button {
55              id: resetAltitudeButton
56              x: 8
57              width: 68
58              height: 27
59              text: qsTr("Reset")
60              anchors.right: parent.right
61              anchors.rightMargin: 8
62              anchors.top: parent.top
63              anchors.topMargin: 468
64          }
65
66          Text {
67              id: altitudeDisplay
```

```
            x: 29
            width: 81
            height: 22
            color: "#ffffff"
            fontSizeMode: Text.FixedSize
            anchors.right: parent.right
            anchors.rightMargin: -1
            anchors.top: parent.top
            anchors.topMargin: 18
            verticalAlignment: Text.AlignTop
            textFormat: Text.AutoText
            horizontalAlignment: Text.AlignHCenter
            wrapMode: Text.WordWrap
            font.pixelSize: 14
        }


        Text {
            id: element1
            x: 35
            width: 67
            height: 19
            color: "#ffffff"
            text: qsTr("Altitude")
            anchors.right: parent.right
            anchors.rightMargin: 8
            anchors.top: parent.top
            anchors.topMargin: 0
            verticalAlignment: Text.AlignVCenter
            horizontalAlignment: Text.AlignHCenter
            font.pixelSize: 14
        }


        Text {
            id: element2
            x: 41
            width: 54
            height: 23
            color: "#ffffff"
            text: qsTr("Voltage")
            anchors.top: parent.top
            anchors.topMargin: 46
            anchors.right: parent.right
            anchors.rightMargin: 14
            horizontalAlignment: Text.AlignHCenter
            verticalAlignment: Text.AlignVCenter
            font.pixelSize: 14
        }

        Gauge {
```

```
            id: voltageGauge
            x: 47
            width: 54
            height: 185
            value: 13.2
            anchors.top: parent.top
            anchors.topMargin: 67
            anchors.right: parent.right
            anchors.rightMargin: 8
            minimumValue: 9.5
            maximumValue: 14
            tickmarkStepSize: 0.5
        }

        Button {
            id: setAltitudeButton
            x: 8
            width: 68
            height: 28
            text: qsTr("Set")
            anchors.right: parent.right
            anchors.rightMargin: 8
            anchors.top: parent.top
            anchors.topMargin: 435
        }

        Slider {
            id: setAltitudeBar
            x: 33
            width: 22
            height: 89
            anchors.right: parent.right
            anchors.rightMargin: 29
            anchors.top: parent.top
            anchors.topMargin: 303
            orientation: Qt.Vertical
            value: 0
        }

        TextField {
            id: enterAltitudeField
            x: 8
            width: 68
            height: 26
            anchors.right: parent.right
            anchors.rightMargin: 8
            anchors.top: parent.top
            anchors.topMargin: 403
            placeholderText: qsTr("Text Field")
            validator: IntValidator{bottom: 0; top: 15;}
            focus: true
```

```
            onAccepted: {
                setAltitudeValue.text = text
                setAltitudeBar.value = text
                HydraDrone.setAltitude(text)
            }
        }

        Text {
            id: setAltitudeValue
            x: 8
            width: 68
            height: 16
            color: "#ffffff"
            text: qsTr("0")
            anchors.right: parent.right
            anchors.rightMargin: 8
            anchors.top: parent.top
            anchors.topMargin: 281
            horizontalAlignment: Text.AlignHCenter
            font.pixelSize: 14
        }

    }

    Dialog{
        id: armDialog
        title: "Are you sure you want to arm?"
        standardButtons: StandardButton.Ok | StandardButton.Cancel
        onAccepted: HydraDrone.arm()
    }

    Dialog{
        id: takeoffDialog
        title: "Are you sure you want to takeoff?"
        standardButtons: StandardButton.Ok | StandardButton.Cancel
        onAccepted: HydraDrone.takeoff()
    }

    Rectangle {
        id: mapRectangle
        color: "#ffffff"
        anchors.top: parent.top
        anchors.topMargin: 42
        anchors.bottom: parent.bottom
        anchors.bottomMargin: 58
        anchors.right: parent.right
        anchors.rightMargin: 84
        anchors.left: parent.left
        anchors.leftMargin: 0


```

```
221         // Qml class that handles the map interface
222         Map {
223             id: mapOfEurope
224             anchors.rightMargin: 0
225             anchors.bottomMargin: 0
226             anchors.centerIn: parent;
227             anchors.fill: parent
228             plugin: Plugin {
229     // other possible open-source plugins for
230     // maps are mapboxgl and osm
231                 name: "esri" // "mapboxgl", "esri", "osm"
232             }
233 
234             MapQuickItem {
235                 id: marker
236                 anchorPoint.x: imageMarker.width/2
237                 anchorPoint.y: imageMarker.height
238     // initial location for drone on map is
239     // center of corvallis, this is set before
240     // current location from drone is retrieved
241                 coordinate: corvallis
242                 sourceItem: Image {
243                     id: imageMarker
244                     source: "marker.png"
245                 }
246             }
247 
248             // Qml class that handles the display of the drone
249             Plane {
250                 id: cppPlane
251                 pilotName: "Hydra"
252                 coordinate: HydraDrone.position
253 
254                 MouseArea {
255                     onClicked: {
256                         if (cppPlaneAnimation.running || HydraDrone.
    isFlying()) {
257                             console.log("Hydra still in the air");
258                             return;
259                         }
260                         cppPlaneAnimation.rotationDirection =
    HydraDrone.position.azimuthTo(HydraDrone.to)
261                         cppPlaneAnimation.start();
262                         cppPlane.departed();
263                     }
264                 }
265             }
266             visibleRegion: viewOfCorvallis
267         }
268     }
269 
```

```
    Timer {
        interval: 1000; running: true; repeat: true;
        onTriggered: {
            console.log(HydraDrone.altitude())
            // retrieves altitude from Drone class
            altitudeDisplay.text = HydraDrone.altitude() + " meters"
            console.log("battery:")
            console.log(HydraDrone.batteryVoltage())
            // retrieves voltage from Drone class
            voltageGauge.value = HydraDrone.batteryVoltage()
            // retrieves drone status (armed/disarmed) from Drone
    class
            if (HydraDrone.isArmed() == 0){
                statusIndicator.color = "red"
            }
            else if(HydraDrone.isArmed() == 1){
                statusIndicator.color = "green"
            }
            // retrieves flight mode from Drone class
            element5.text = HydraDrone.flightMode()

        }
    }

    Rectangle {
        id: bottomRectangle
        y: 443
        height: 57
        color: "#2e3436"
        anchors.right: parent.right
        anchors.rightMargin: 84
        anchors.bottom: parent.bottom
        anchors.bottomMargin: 0
        anchors.left: parent.left
        anchors.leftMargin: 0

        Button {
            id: takeoffButton
            x: 17
            y: 15
            text: qsTr("Takeoff")
            anchors.verticalCenter: parent.verticalCenter
            anchors.bottom: parent.bottom
            anchors.bottomMargin: 14
            onClicked: takeoffDialog.open()
        }

        Button {
            id: landButton
            x: 115
            y: 15
```

```
320             text: qsTr("Land")
321             anchors.verticalCenter: parent.verticalCenter
322             anchors.bottom: parent.bottom
323             anchors.bottomMargin: 14
324             onClicked: HydraDrone.land()
325         }
326 
327         Button {
328             id: armButton
329             x: 212
330             y: 15
331             text: qsTr("Arm")
332             anchors.verticalCenter: parent.verticalCenter
333             anchors.bottom: parent.bottom
334             anchors.bottomMargin: 14
335             onClicked: armDialog.open()
336         }
337 
338         Button {
339             id: disarmButton
340             x: 310
341             y: 15
342             text: qsTr("Disarm")
343             anchors.verticalCenter: parent.verticalCenter
344             anchors.bottom: parent.bottom
345             anchors.bottomMargin: 14
346             onClicked: HydraDrone.disarm()
347         }
348 
349         Button {
350             id: killButton
351             x: 404
352             y: 15
353             text: qsTr("Kill")
354             onClicked: HydraDrone.kill()
355 
356         }
357 
358         Button {
359             id: returnHomeButton
360             x: 500
361             y: 15
362             text: qsTr("Return Home")
363             onClicked: HydraDrone.return_home()
364         }
365     }
366 
367     Rectangle {
368         id: topRectangle
369         height: 41
370         color: "#2e3436"
```

```
        anchors.right: parent.right
        anchors.rightMargin: 84
        anchors.left: parent.left
        anchors.leftMargin: 0
        anchors.top: parent.top
        anchors.topMargin: 0

        StatusIndicator {
            id: statusIndicator
            x: 70
            y: 0
            color: "#008000"
            active: true
        }

        Text {
            id: element
            x: 0
            y: 12
            width: 75
            height: 21
            color: "#ffffff"
            text: qsTr("ARMED")
            horizontalAlignment: Text.AlignHCenter
            font.pixelSize: 14
        }

        Text {
            id: element3
            x: 152
            y: 10
            width: 105
            height: 21
            color: "#ffffff"
            text: qsTr("FLIGHT MODE:")
            horizontalAlignment: Text.AlignHCenter
            font.pixelSize: 14
        }

        Text {
            id: element5
            x: 270
            y: 10
            width: 105
            height: 21
            color: "#ffffff"
            horizontalAlignment: Text.AlignHCenter
            font.pixelSize: 14
        }
    }
}
```

8.1.4 Plane.qml

```
import QtQuick 2.4
import QtLocation 5.6

MapQuickItem {
    id: plane
    property string pilotName;
    property int bearing: 0;

    anchorPoint.x: image.width/2
    anchorPoint.y: image.height/2

    sourceItem: Grid {
        columns: 1
        Grid {
            horizontalItemAlignment: Grid.AlignHCenter
            Image {
                id: image
                rotation: bearing
                source: "airplane.png"
            }
            Rectangle {
                id: bubble
                color: "lightblue"
                border.width: 1
                width: text.width * 1.3
                height: text.height * 1.3
                radius: 5
                Text {
                    id: text
                    anchors.centerIn: parent
                    text: pilotName
                }
            }
        }

        Rectangle {
            id: message
            color: "lightblue"
            border.width: 1
            width: banner.width * 1.3
            height: banner.height * 1.3
            radius: 5
            opacity: 0
            Text {
                id: banner
                anchors.centerIn: parent
            }
            SequentialAnimation {
                id: playMessage
```

```
                running: false
                NumberAnimation { target: message;
                    property: "opacity";
                    to: 1.0;
                    duration: 200
                    easing.type: Easing.InOutQuad
                }
                PauseAnimation  { duration: 1000 }
                NumberAnimation { target: message;
                    property: "opacity";
                    to: 0.0;
                    duration: 200}
            }
        }
    }
    function showMessage(message) {
        banner.text = message
        playMessage.start()
    }
}
```

8.1.5 MAVSDK Server

8.1.6 hydra_server.py

```
#!/usr/bin/env python3
"""
* Program: hydra_server.py
* Author: Tomas McMonigal
* Date: 2/15/2020
* Description: Multi-threaded server that accepts incoming connections to send
    telemetry data to GUI, CP, and PP programs. Receives telemetry data from
    ttyUSB0. To run it provide the port number as a command line parameter.
 The port number for MAVProxy is 14555.
"""

import asyncio
from mavsdk import System
from concurrent.futures import TimeoutError
from mavsdk import (Attitude, OffboardError)

async def echo_server(reader, writer, drone):
 """Function that is run as a thread for
 every connection established with a client
 """
    while True:
        print("waiting on data requests")
        marker = 1
```

```
24
25  # waits 10 seconds to receive data from client
26  # if no data is received, connection is closed
27        try:
28            data = await asyncio.wait_for(reader.read(100), timeout
     =10)
29            if not data: # client has disconnected
30                print("connection closed by client: closing
     connection with client")
31                break
32        except TimeoutError:
33            print("Timeout error: closing connection with client")
34            break
35
36        ********** if data is valid and connection active
     ****************
37        data_in = data.decode() # decodes utf-8 only
38        data_in = data_in.replace('\r\n', '')
39
40  ********** processes the request accordingly **********************
41        if data_in == "latitude":
42            print("latitude requested")
43            async for position in drone.telemetry.position():
44                data_out = str(position.latitude_deg) + '\n'
45                writer.write(data_out.encode())
46                await writer.drain()
47                break
48
49        elif data_in == "altitude":
50            async for position in drone.telemetry.position():
51                print("altitude requested")
52                data_out = str(position.relative_altitude_m) + '\n'
53                writer.write(data_out.encode())
54                await writer.drain()
55                break
56
57        elif data_in == "longitude":
58            async for position in drone.telemetry.position():
59                print("longitude requested")
60                data_out = str(position.longitude_deg) + '\n'
61                writer.write(data_out.encode())
62                await writer.drain()
63                break
64
65        elif data_in == "battery":
66            async for battery in drone.telemetry.battery():
67                print("battery percentage requested")
68                data_out = str(battery.voltage_v) + '\n'
69                writer.write(data_out.encode())
70                await writer.drain()
71                break
```

```
72
73          elif data_in == "arm":
74              print("-- Arming")
75              await drone.action.arm()
76
77          elif data_in == "disarm":
78              try:
79                  print("-- Disarming")
80                  await drone.action.disarm()
81              except:
82                  print("error disarming")
83
84          elif data_in == "kill":
85              print("-- Killing")
86              await drone.action.kill()
87
88          elif data_in == "is_armed":
89              async for is_armed in drone.telemetry.armed():
90                  print("Is_armed requested:", is_armed)
91                  data_out = str(is_armed) + '\n'
92                  writer.write(data_out.encode())
93                  await writer.drain()
94                  break
95
96          elif data_in == "flight_mode":
97              async for flight_mode in drone.telemetry.flight_mode():
98                  print("flight mode requested:", flight_mode)
99                  data_out = str(flight_mode) + '\n'
100                 writer.write(data_out.encode())
101                 await writer.drain()
102                 break
103
104         elif data_in == "takeoff":
105             print("-- Taking off")
106             await drone.action.takeoff()
107
108         elif data_in == "land":
109             print("-- Landing")
110             await drone.action.land()
111
112         elif data_in == "return":
113             print("-- Returning home")
114             await drone.action.return_to_launch()
115
116         else:
117             print("error: data requested not supported, closing
    connection")
118             output_string = '|' + data_in + '|'
119             print(output_string)
120     writer.close()
121
```

```
async def main(host, port):
    drone = System()
 # connects to drone through MAVProxy using port number 14555
    await drone.connect(system_address="udp://:14555")
    print("Waiting for drone to connect...")
    async for state in drone.core.connection_state():
        if state.is_connected:
            print(f"Drone discovered with UUID: {state.uuid}")
            break

 # waits for gps to be accurate
    print("Waiting for drone to have a global position estimate...")
    async for health in drone.telemetry.health():
        if health.is_global_position_ok:
            print("Global position estimate ok")
            break

 # sets current location as initial point
    print("-- Setting initial setpoint")
    await drone.offboard.set_attitude(Attitude(0.0, 0.0, 0.0, 0.0))

 # sets takeoff altitude to 5m
    print("-- Setting takeoff altitude to 5m")
    await drone.action.set_takeoff_altitude(5)
    print (await drone.action.get_takeoff_altitude())

 # server is now ready to accept incoming connections from clients
    print("Ready to accept incoming connections")
    server = await asyncio.start_server(lambda r, w: echo_server(r,
   w, drone), host, port)
    async with server:
        await server.serve_forever()

asyncio.run(main('127.0.0.1', 5000))
```