Autonomous Airspace

Ekaterina Tolstaya, Rhea Jain, Brent Schlotfeldt

ESE519 Final Project Report

Table of Contents

Motivation

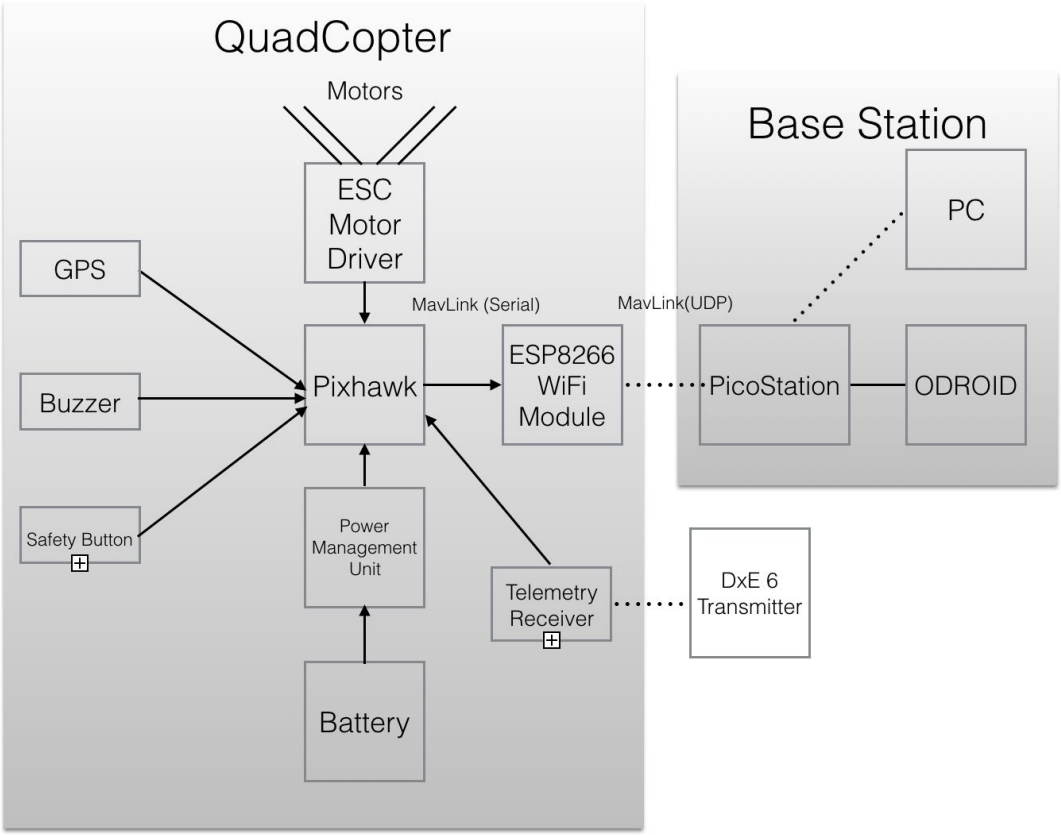
Drones, specifically autonomous quadrotors, are poised to make a huge impact for businesses and society in general. Companies are already using drones for delivery, surveillance, and photography. There are countless other applications like agriculture and communication. In order to operate a dense fleet of quadrotors in a busy urban environment, an autonomous routing paradigm must be developed to allow for safe and efficient operation.

Amazon Prime Air's delivery drone tested in Cambridge, UK on December 14th, 2016.
[source]

We aim to create a quadrotor based testbed for existing and new traffic control algorithms. This system will accelerate the development of modular self-contained autonomous routing controllers for dense low-altitude fleets of quadrotors. This platform will be used by students and researchers as a low-cost, but realistic tool for testing outdoor navigation algorithms.

System Design: The Big Picture

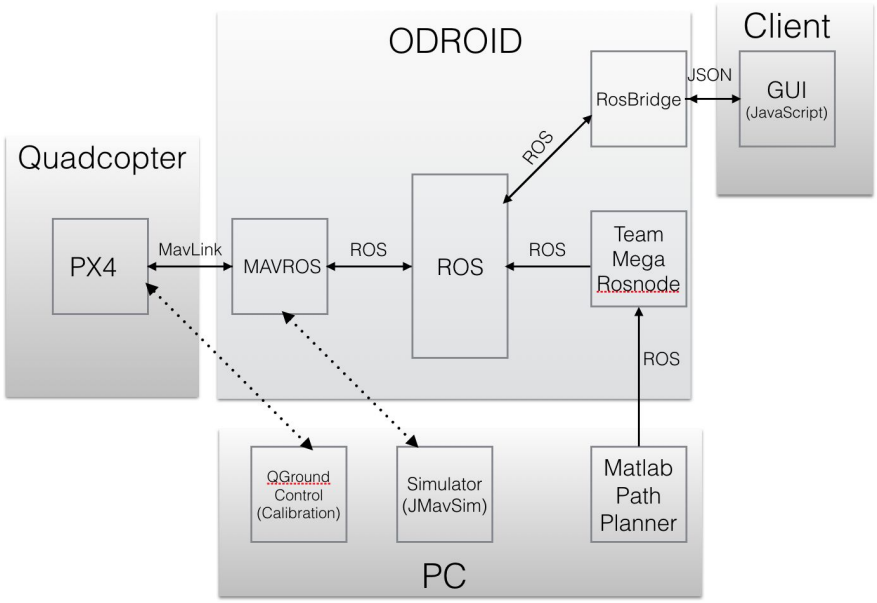
Hardware Architecture



Our goal was to create a simple, inexpensive system to coordinate the motion of several quadrotors concurrently. We chose the PixHawk autopilot to be the only onboard computer as it is the readily available industry standard for open-source UAVs. The PixHawk supports two software/firmware stacks: APM and PX4. We found that PX4 has better support for offboard control mode. We chose to avoid using an additional onboard computer to decrease cost and complexity of the system. Instead, the PixHawk is TELEM2 serial port is connected to the ESP8266 WiFi module, enabling the PixHawk to connect to the ground station over UDP.
We chose the DJI F450 frame as it is reliable, lightweight and came with copatible motors and motor drivers. We chose to use only GPS for localization as our target application was an outdoor air traffic controller. The GPS resolution and drift caused problems later in our project. We did not have any additional sensors onboard the quadrotor.

The base station controller runs on a ODROID computer, which is more powerful than a RPi, which was important for our application. The PicoStation is a great portable, outdoor WiFi station. We used a DXe transmitter to cut costs and meet our requirements.

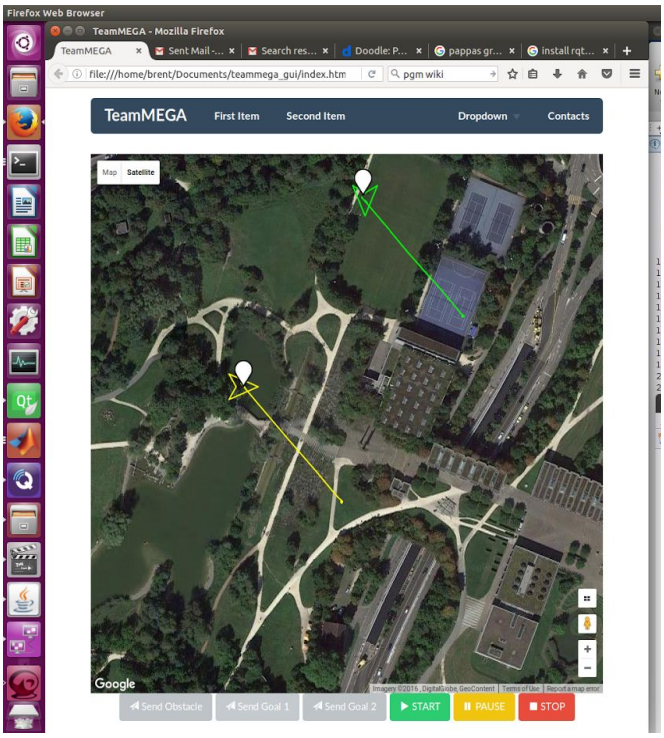
Software Architecture



ROS and MAVROS

Our software system was based on the ROS/ MAVROS platform. We discovered early on in the project that the proper way to communicate to the Quadrotor was through a standardized messaging system called MAVLink. This messaging framework is convenient for navigation, and other purposes because it allows specific waypoints, as well as attitude control commands to be sent to the vehicles. In addition, our group had some previous experience using the Robot Operating System, (ROS) which is a common framework for messaging data and commands in robotic systems. ROS is designed to be modular, and networked, which allows us to effectively use different pieces of software on devices that are all operating on the same network, with seamless communication. ROS is a generic framework, but it has plugins that allow additional functionality. This is where MAVROS comes in. MAVROS is a software bridge between ROS, and MAVLink, that allows us to use ROS style communication (Node Publishers, Subscribers, and Services) to control our Quadrotor which takes in the MAVLink message format. MAVLink commands are then interpreted by the PX4 firmware, and applied to the quadrotor. ROS has classically supported languages such as C++, and Python, and has recently added a toolbox for MATLAB. In summary, using MAVROS allowed for the most seamless and direct communication between all of our computation and path planning services, and the quadrotors themselves, while being able to run on either our laptops, or the ODROID base station on the same network.

Graphic User Interface

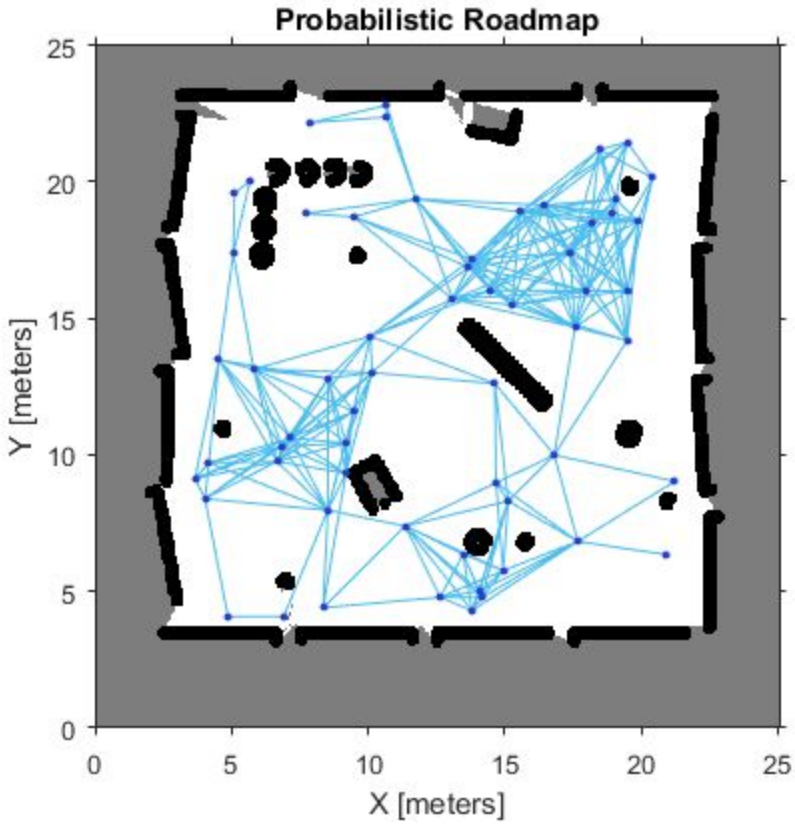


As part of our project, we created a graphical user interface by which users will be able to visualize the quadrotor trajectories in real time, as well as give them commands to land, or create additional weather patterns or obstacles to avoid. The GUI we implemented is an advancement on Rendezvous, an open source platform that visualized GPS coordinates on a map that are given by the Robot Operating System. In other words, our GUI is another piece in the ROS network that receives the data, and through a bridge with JSON called ROSBRIDGE, is able to feed back data about obstacles and landing locations that are determined by us. Once a human user selects this information, the quadrotors are able to fly autonomously, and complete the mission through the planner.

Path Planner

One of the most important pieces of software we developed, is the planner, which generates trajectories for the quadrotors. We used MATLAB to implement this, due to time constraints and our group's prior experience. MATLAB has a toolbox that connects to ROS, so we were able to create additional ROS topics that published the waypoints we generated and MAVROS would then forward these to the quadrotor. The algorithm used is called Probabilistic Roadmap (PRM), which generates a large graph of

connected nodes, and then determines if there is a way to plan a path between start and goal locations, while avoiding obstacles. Other choices for path planning include OMPL, the Open source Motion Planning Library, although we were unable to compile it for Ubuntu 16.04, and given time constraints.



In MATLAB, the planner succeeds by first creating an occupancy grid, as seen above (). This indicates where the obstacles are at each instant, and the planner will attempt to find paths not intersecting these fixed obstacles. For our purposes, we indicate bad weather events that can be selected in the GUI, as obstacles to avoid.

Simulator

The final essential piece of software we included, was the simulator which allowed us to test trajectory planning using complete physics models of the quadrotor. The simulators we looked into included jMavSim, as well as Gazebo. Both of them are effective, and act similar to a quadrotor over a network, except that the IP Address is on the local machine. This allows us to connect our MAVROS system to the quadrotor in the same way it would work when using the Picostation Wi-Fi. Ultimately,we chose to use jMavSim, due to difficulties installing Gazebo correctly on Ubuntu 16.

Hardware Development

Bill of Materials

These parts were used to build four quadrotors and two base stations. There are several optional parts that we did not fully utilize, but may be useful for extensions of this project. Parts from other vendors and manufacturers can be substituted for some items below. The DJI F450 ARF + Naza autopilot kit was used as a proof of concept. The DJI F450 ARF without Naza can be purchased instead.

Quadrotor Components *[Item, Cost, Quantity]*

- CrazePony PixHawk Autopilot $74 x 4
- DJI F450 ARF + Naza $300 x 1, *optional*
- DJI F450 ARF $190 x 3
- DJI F450 landing gear $5 x 3
- Extra Props $10.89 x 9
- Ublox NEO x M8N GPS $28.99 x 4
- Power Module $10.98 x 4
- ESP8266 Serial to Wifi $6.50 x 4

Base Station Components

- PicoStationM2HP $72.52 x 2
- ODROID XU4 with US Plug $74 x 2
- USB GPS Module $25 x 2, *optional*
- 64 GB eMMC module with eMMC and USB adapters $75.9 x 2

Radio Telemetry Components

- DSMX Remote Receiver $34.99 x 4
- DXe Transmitter $59.99 x 2
- DXe Programming cable $17.99 x 2, *optional*

Batteries and Charging

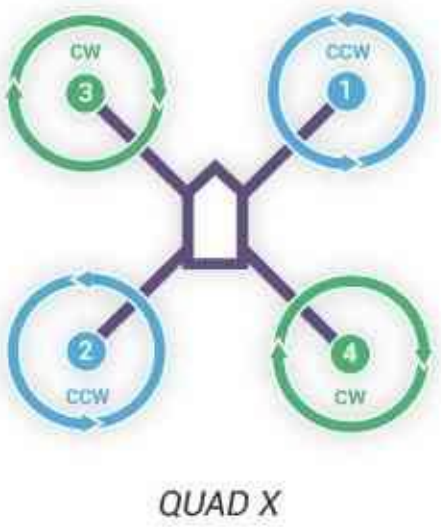
- Accucell S60 AC Charger $29 x 4
- Battery Bag $7.80 x 8
- 5000 mAh battery 4s $19.53 x 8
- Battery Straps 5 pack $7.99 x 1
- Deans Plug 10 Pack $5.95 x 1
- Charger Deans plug adapters (4 pack) $10.99 x 1

BATTERY WARNING: Li-Po fires are extremely dangerous. Use 4S (or greater) batteries for this project. Store batteries in fire-proof bags when not in use. Supervise charging and do not charge overnight. Dispose of batteries properly if they appear puffy. Use a female Deans plug connector on the battery to prevent accidental shorts.

Quadrotor Assembly

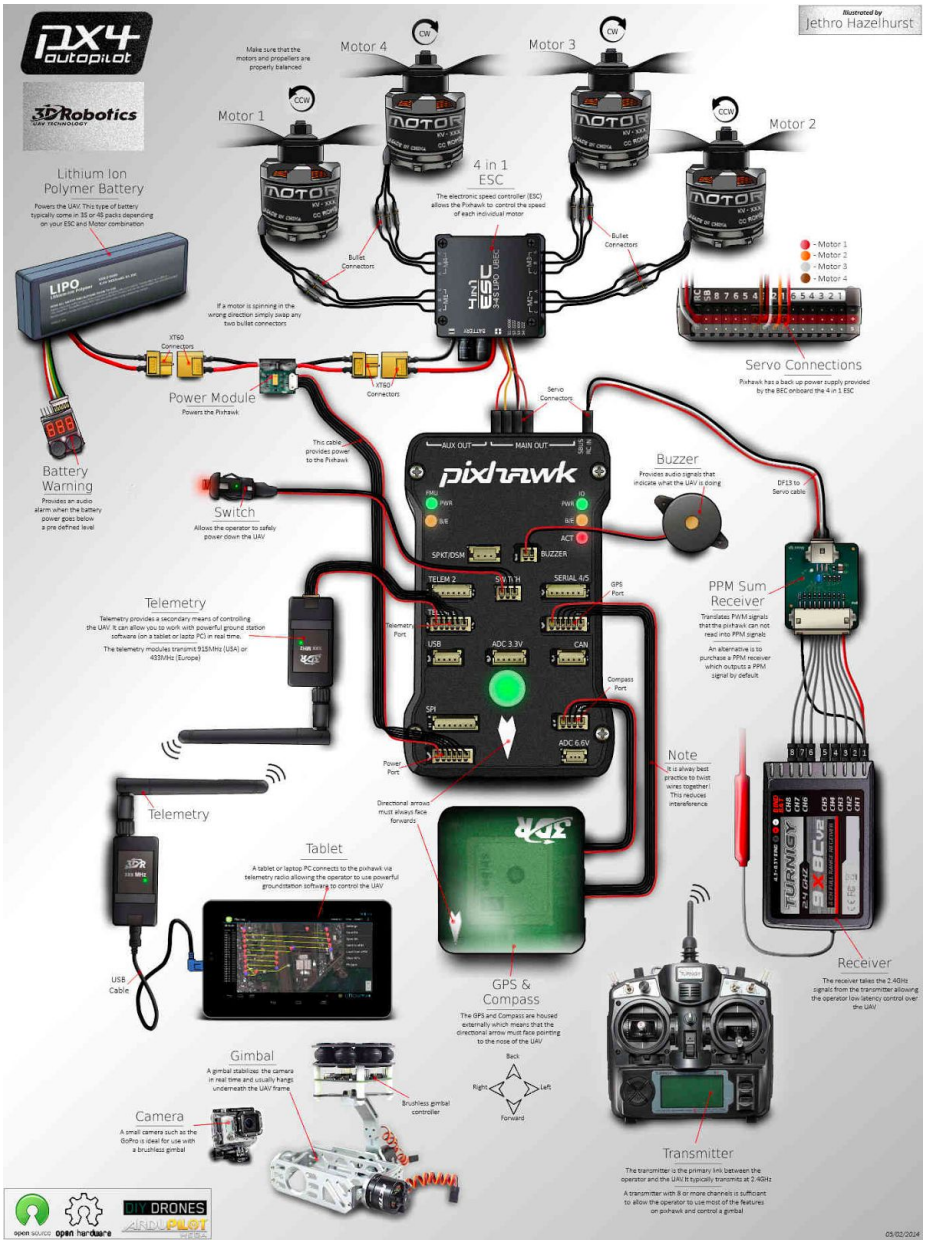
We followed a [video] to do the majority of assembly of the DJI F450 frame.The necessary tools are: a wire stripper, a soldering iron, Hex screwdriver, heat shrink, and zip ties.

During assembly, keep track of motor orientations and numbering. The video above was made for the Naza autopilot, but PixHawk has a different motor numbering:



PixHawk motor numbering for a X-Quadrotor.

After assembling the body of the quadrotor, the next step is wiring all sensors and components to the Pixhawk. A step-by-step guide for Pixhawk wiring can be found [here]. Use zip ties and velcro for securing sensors on the body of the quadrotor.



Overview of wiring the PixHawk to onboard components. Not all shown components are required for our project. [source].

For our project, made the following connections using the wires included with the CrazePony PixHawk: PMU to Power Port, Buzzer to Buzzer port, Switch to Switch Port, RC receiver to DSM, ESP8266 to TELEM2, GPS to GPS Port and I2C, ESCs to MAIN OUT 1,2,3,4

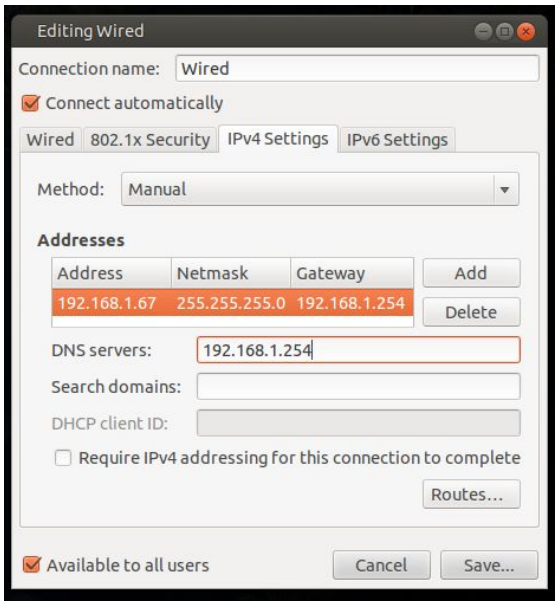
Base Station Setup

The base station includes the PicoStation and ODROID. The PicoStation acts as a WiFi access point, through which the ODROID, multiple PixHawks and any additional PCs can communicate to send information and commands.

We set up the ODROID by connecting a monitor, mouse and keyboard directly to it and using it as a desktop for MAVROS installations. You can also do this via SSH once you can connect the ODROID directly to a PC via ethernet, or to the PicoStation via WiFi. The ODROID eMMC modules come with Ubuntu Mate pre-installed. This OS is very similar to Ubuntu and will run ROS, MAVROS, etc.

To set up the ODROID software stack according to the software installation instructions in later sections. You can also find a guide here. We had trouble compiling MAVROS from source on the ODROID, so just install it from binaries.

We set up the PicoStation as an access point with Static IPs according to the guide here. You need to make sure that your ODROID and any other PCs that you want to connect to to the new network have static IPs. You may have to change this setting multiple times if you need to connect to the PicoStation and then use the Internet.



Change your static/dynamic IP settings. Learn more about IP settings [here]

Quadrotor Calibration and Firmware

We used QGroundControl to set up the firmware on the PixHawk and calibrate the sensors and transmitter. Install QGroundControl from [here].

Quadrotor Firmware and Software setup:

- Firmware upload: install the PX4 stack
- Airframe: select Quadrotor X, F450
- Flight Modes: map switches on the radio to AltHold, Loiter, and Offboard modes
- RC Transmitter binding and calibration: current version of QGroundControl can't bind transmitter to receiver, but the radio calibration works. Can use Mission Planner software to bind receiver to transmitter instead.
- Sensor calibration - calibrate gyro, accel, GPS and level horizon
- Safety Settings (details below)
- WiFi telemetry setup [guide]

Additional instructions for WiFi telemetry setup:

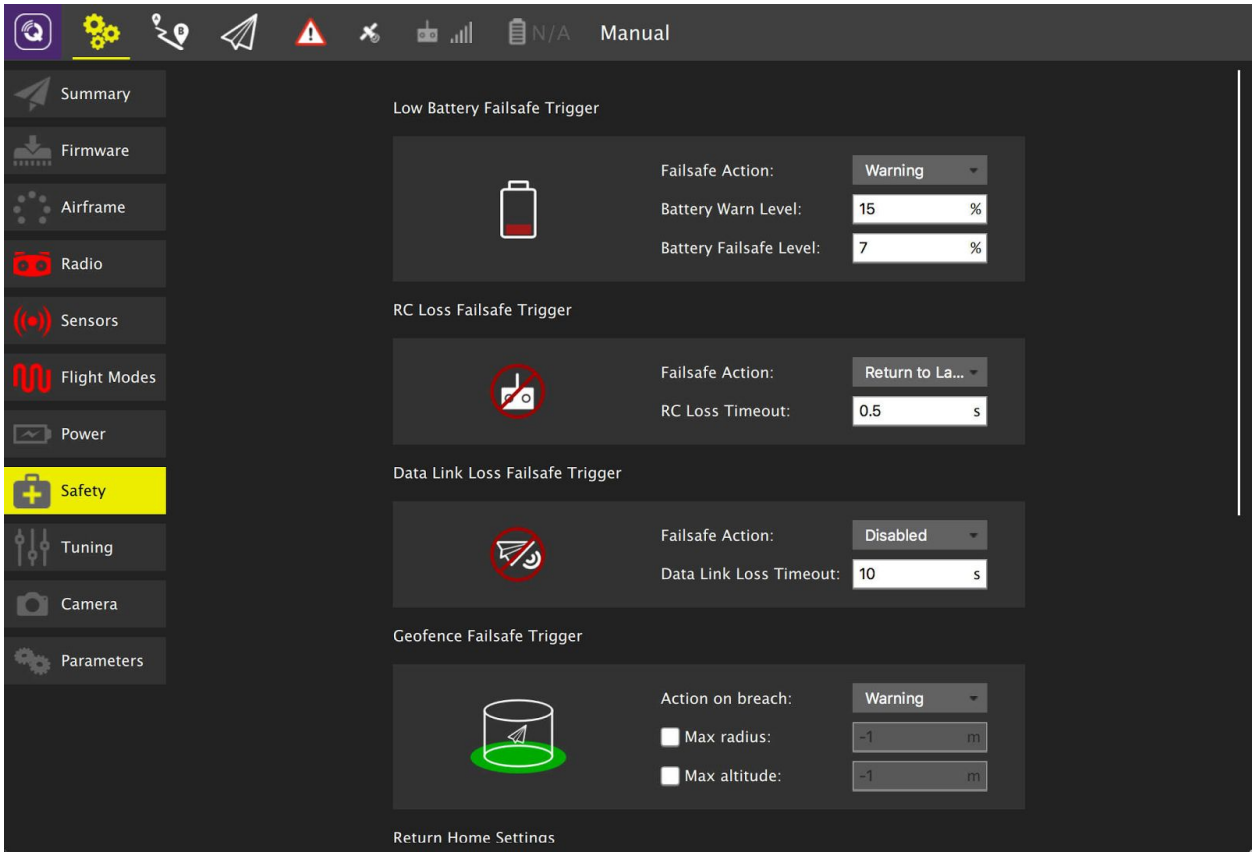
- Upload the [firmware] to the ESP82699 using a FTDI Friend module.
- Connect the WiFi module to the TELEM2 port
- In QGroundControl, change the SYS_COMPANION parameter to 921600.
- Turn off the PixHawk and turn it back on. See if a PixHawk WiFi network now exists.
- Connect to the PixHawk network using a PC, password is pixhawk. To connect to a typical existing network with a static IP, go to the following URL to set parameters on the ESP8266. Ipsta is the PixHawk's new IP. you may need to change this to avoid conflicts. The networkname, pwdsta, subnetsta and gatewaysta come from your PicoStation settings.

http://192.168.4.1/setparameters?mode=1&ssidsta=networkname&pwdsta=thepassword&ipsta=192.168.1.123&gatewaysta=192.168.1.1&subnetsta=255.255.255.0

- Now, you can connect your PC running QGroundControl to the same network. Connect to the quadrotor from your PC by creating a new UDP connection.

Making a connection to PixHawk over UDP.

FAILSAFE WARNING: Correct failsafe settings can prevent damage and injury. All types of failsafes are important: low battery, RC loss, data link loss. For our project, our failsafe settings were set to Land Immediately. Return to Land can also be used, but make sure that your RTL altitude is set correctly, or your quadrotor will crash into ceilings and roofs. A kill switch is another key safety feature that can be set in the Flight Modes menu.



Set Failsafe settings using QGroundControl. [source]

First flight and beyond

Test Arming

Take the propellers off. Switch on the Radio. plug in the battery. Switch the mode to AltHold. Hold the safety switch for about 3 seconds. The PixHawk LED should be flashing green. Arm the quadrotor by holding the throttle stick down and to the right (or to the left depending on your radio mapping). As it arms, the buzzer will beep and the PixHawk LED light transitions to solid green. Motors should start spinning. If LED is not green, check this website for error codes. Once you are able to arm the quadrotor, turn everything off and put propellers on.

Test Flight

With the propellers this time, arm the quadrotor again, and this time gently and slowly push the throttle up. If the quadrotor flips immediately, then the ESC wires are swapped. Change wire orientations and try again. If it doesn't, you can try the Loiter mode (if GPS lock is achieved). We found that it was easiest to control the quadrotor when the autopilot was in Loiter or AltHold modes. Eventually, you can learn to use Manual mode and others.

Autonomous Flight

Next, we moved on to testing autonomous flight and MAVLink protocol using the QGroundControl software. There is a detailed guide that we followed here. In this mode, the quadrotor follows given GPS waypoints autonomously.

FLIGHT SAFETY WARNING: Ensure all quadrotor components are secure. Ensure that there are no obstacles or people nearby. Ensure that radio and base station settings are correct. Allow time for GPS lock if necessary. If there are more than one quadrotors bound to the same receiver, always secure or turn off quadrotors not in use.

Software Development

Installation Requirements

See each individual link for installation instructions.

1. A simulator of choice (jMavSim or Gazebo) (http://dev.px4.io/simulation-sitl.html)

2. MAVROS (http://dev.px4.io/ros-mavros-installation.html)
3. ROS (Tested with Kinetic) (http://wiki.ros.org/kinetic/Installation/Ubuntu)
4. MATLAB Planner (https://github.com/brent8149/Matlab_planner)
5. teamMEGA ROS Node (https://github.com/brent8149/teamMEGA)
6. GUI Application (https://github.com/katetolstaya/teammega_gui)
7. ROSBridge (https://github.com/RobotWebTools/rosbridge_suite)

Otherwise, simply note the IP Address of your quadrotor on the network.

To run MAVROS: (Check the port / IP)
For simulator:

```
roslaunch mavros px4.launch fcu_url:="udp://:14540@127.0.0.1:14557"
```

For Serial / USB:

```
roslaunch mavros px4.launch fcu_url:="serial:///dev/ttyACM0:921600"
```

For real drone over WIFI:

```
roslaunch mavros px4.launch fcu_url:="udp://:14550@192.168.4.1:14555"
```

Launch ROSbridge

```
roslaunch rosbridge_server rosbridge_websocket.launch
```

Launch web-app

```
open teammega_gui/index.html
```

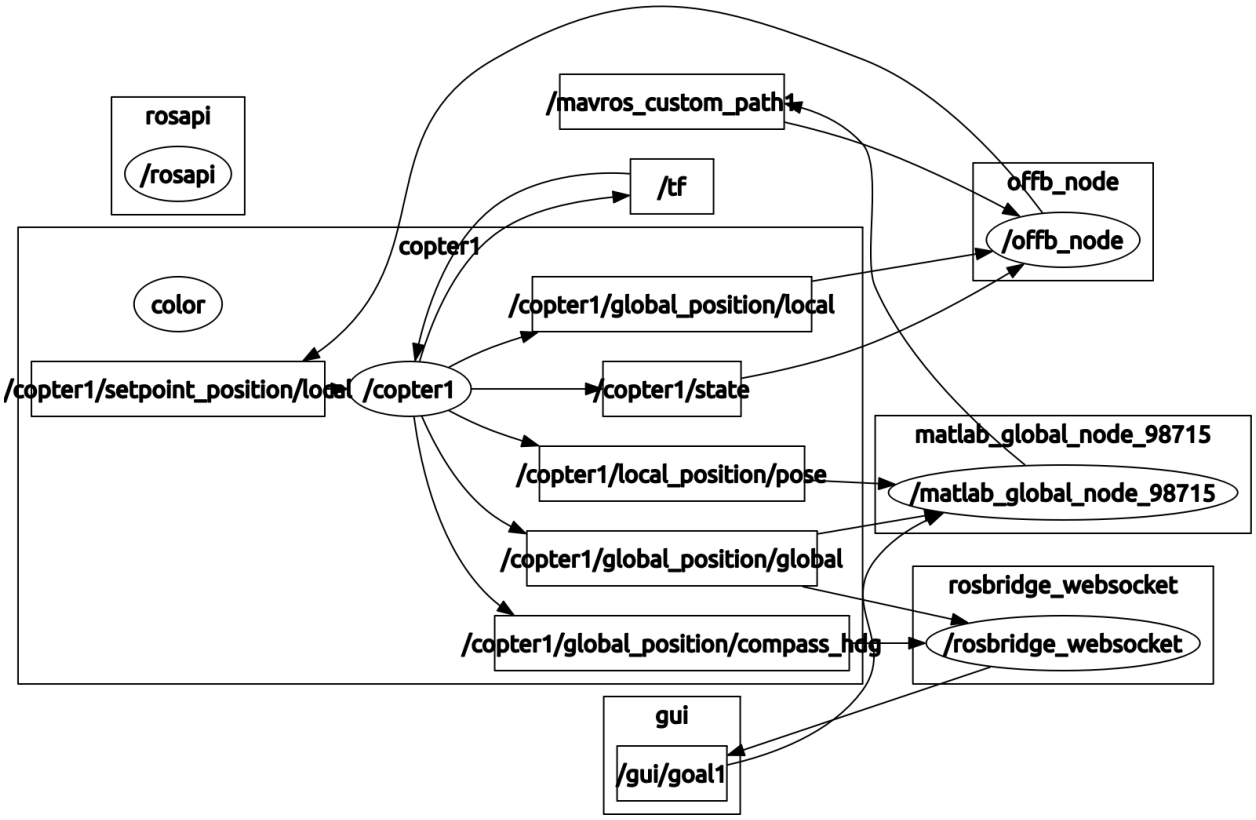
Launch team_mega

```
rosrun teamMega teamMega
```

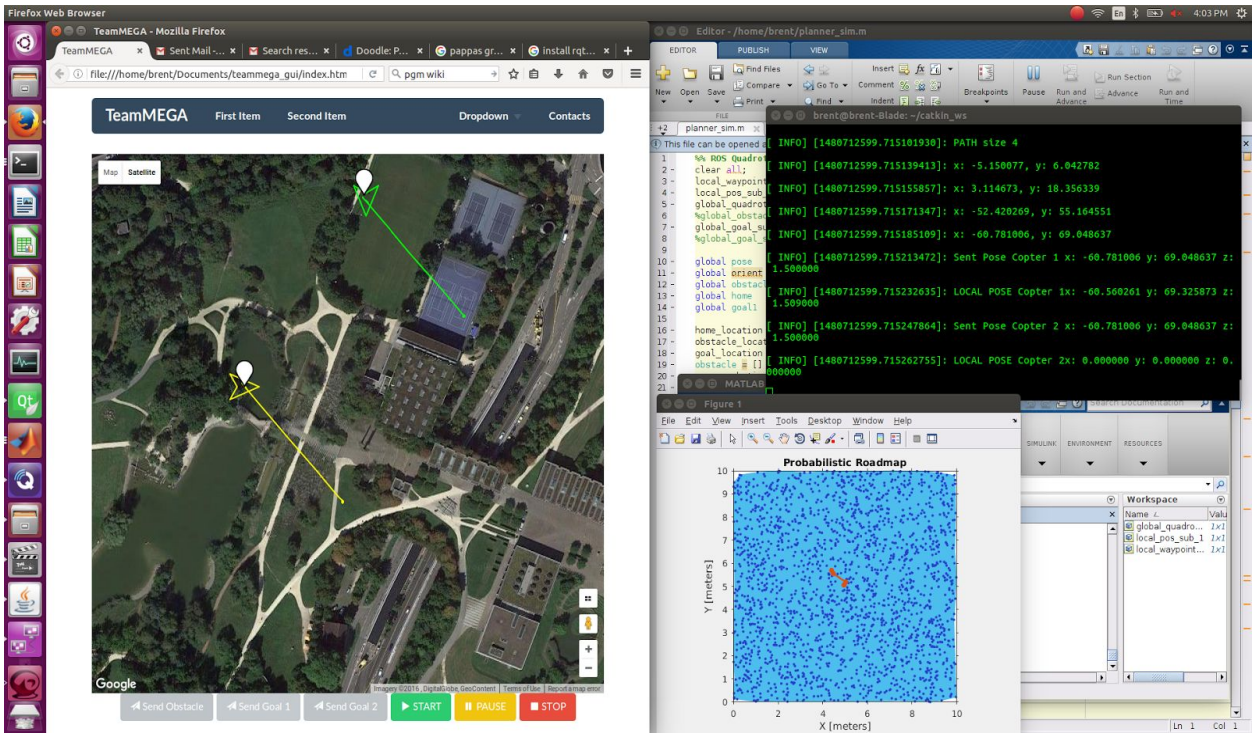
Launch matlab

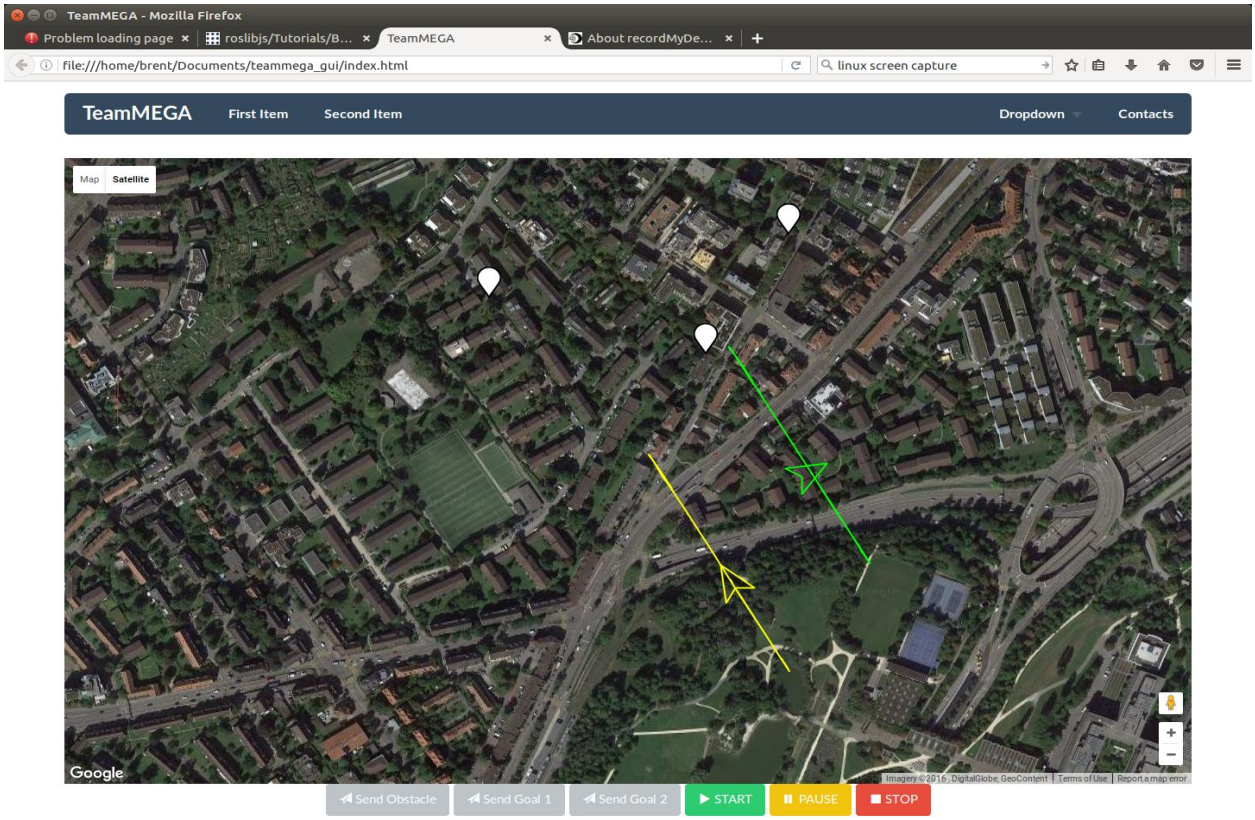
```
matlab planner_sim.m
```

After the above commands have been executed, you will be able to click landing locations and obstacles in the GUI, and observe the planner creating appropriate trajectories for the system, and then successfully flying the quadrotors.



ROS Node architecture (above), and full system in simulation (below).





Multiple Quadrotor trajectories planned simultaneously.

Conclusions and Results

We successfully designed a low-cost and robust system consisting of 2 quadrotors and a base station. We set up MAVROS to communicate with 2 PixHawks. MAVROS is not designed for communicating with multiple vehicles from one node, so we had to run 2 instances of MAVROS nodes with different names. This was achieved by modifying the ROS launch files included in the MAVROS repository. Then, we were able to subscribe to sensor data topics.

Our project goals were refined after the first demonstration as follows:

- First, create a simple GUI that shows current locations of quadrotors and obstacles. The user can click to pick landing positions for the quads or to generate an obstacle. The GUI also makes it easy to view and switch between flight modes.
- Mission 1: 1-4 quadrotors take off, fly to targets, and return to takeoff location

- Mission 2: Mission 1 + quadrotors avoid a 3D obstacle can move based on user controls
- Mission 3: Mission 1 + quadrotors avoid a sphere around a manually-controlled quadrotor

We worked on developing the GUI immediately. We based our approach on this project, called Rendezvous and added support for multiple vehicles. We also implemented the feature that allows us to click on the map and publish waypoints to ROS, rather than just subscribing to data. The main building blocks of the system are the HTML and JS code running in the browser, and also ROSBRIDGE that allows JSON queries to send and receive ROS messages. We tested the GUI with 2 simulated quadrotors and 2 actual quadrotors.

Some difficulties were encountered while implementing the clickable GPS landing locations that were used for creating missions. Our planner relied on having cartesian coordinates for planning, e.g. in the X-Y plane. Thus, it was essential to use a mapping from GPS to some sort of flat world space. To solve this problem, we used coordinates generated from the Universal Transverse Mercator (UTM) system, which is a projection of the globe onto a flat map. Over short distances, such as where our quadrotors would fly, this proved effective in planning accurate trajectories.

We worked on the planning and control portion of the code. We used the Matlab ROS libraries to incorporate the PRM path planning method .

Finally, we were able to send take-off and circling commands to the quadrotors in outdoor experiments, but we were unable to get precise tracking of paths given by the GUI. Incorrect RTL settings, caused damage to the quadrotor with too little time left to redo the experimental portion.

Ultimately, we are very proud of the software stack we have created. It is a great tool for further algorithm testing and development, both in simulation and experimentally.