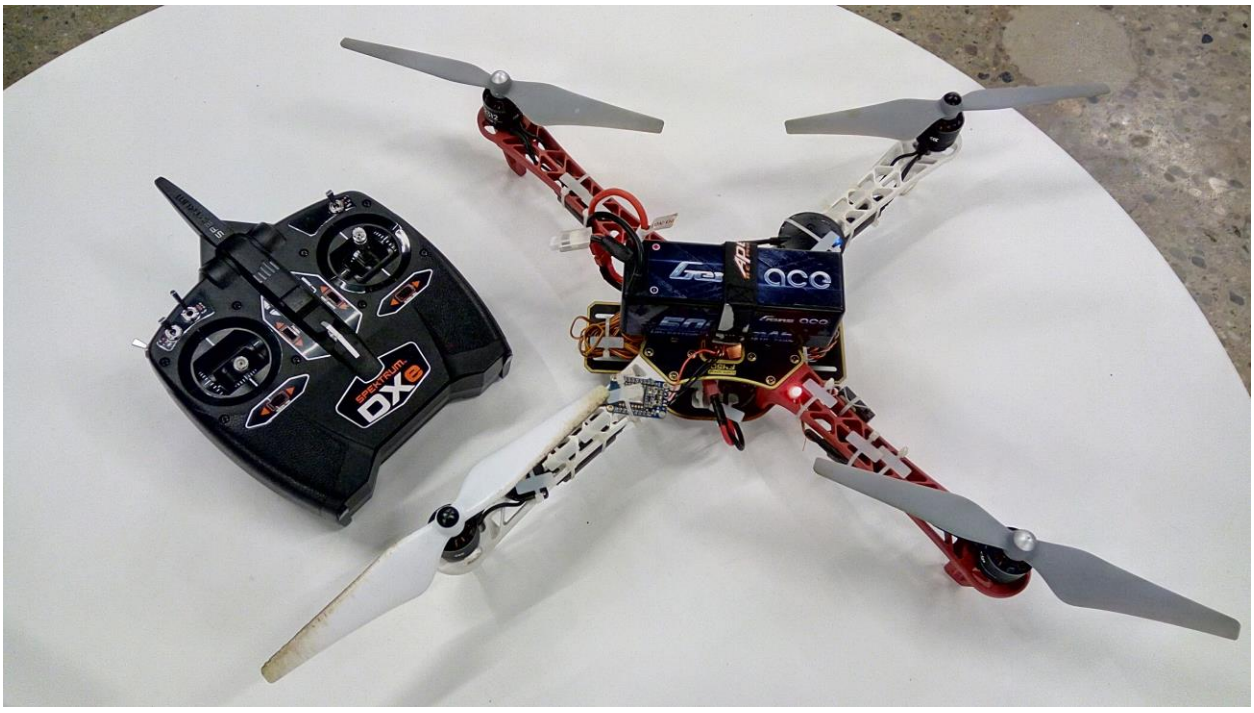
Autonomous Air Traffic Controller

Project Report

University of Pennsylvania
ESE-519: Real Time Embedded Systems
Team Firestorm
Fall-2016



Abstract

Autonomous Air Traffic Control (AATC) project is a simulation and a demonstration platform for a portable ATC setup for UAVs which will assist in an ongoing research focussing on segregated airspace for future high density autonomous traffic and commercial aviation like Amazon UAV delivery.

AATC consists of a base station for the ATC comprising a single board computer for running flight stacks and associated code supported with a Wi-Fi access point setup to broadcast messages to quadrotors. There are two or more quadrotors in the airspace (Wi-Fi signal range) of the base station, which are controllable and execute missions based on the messages received by them. Each Quadrotor has a GPS module for the basic localization and has other fail safe configurations like geofencing, low battery and ground control station (GCS) link.

The system can be expanded easily by adding more quadrotors, which have been tested to be working fine with the existing software, as easy as assigning new network IPs to them. The software running on ATC is derived from code running simulations on Ardupilot SITL (Software in the loop) simulator. These simulations included various missions like single quad take-off and landing, multiple quads sequential and parallel take-off and landing, obstacle avoidance etc.

Introduction

Civilian and military air space has so far been very efficiently controlled by existing Air Traffic Control with the primary motive of completion of a flight safely. All this is done manually with visual and navigation instrument assistance. Also, ATC is highly centralized and has many static and hierarchical points of reference.

Autonomous ATC for UAVs is designed to have a decentralized airspace which is portable with singular point of reference. This ATC must coordinate on a semi ad hoc basis between drones waiting to land. It should be able to manage in lost-link situations and other emergency situations plus setting a schedule of landings.

Our motivation was to have a portable autonomous ATC setup for quadrotors to enable coordination between several quadrotors in a simulated TRACON environment with communication protocols established to give real time ATC directives like holding, new routing etc. and simulate real TRACON contingencies.

Our goals were to have:

a. a pre-programmed Take-off sequence

a. a pre-programmed Land-in sequence
b. ATC directed sequential and parallel take-off and land-in
c. Simulate mayday events or airfield events like an emergency landing of quadrotor or obstacle avoidance.

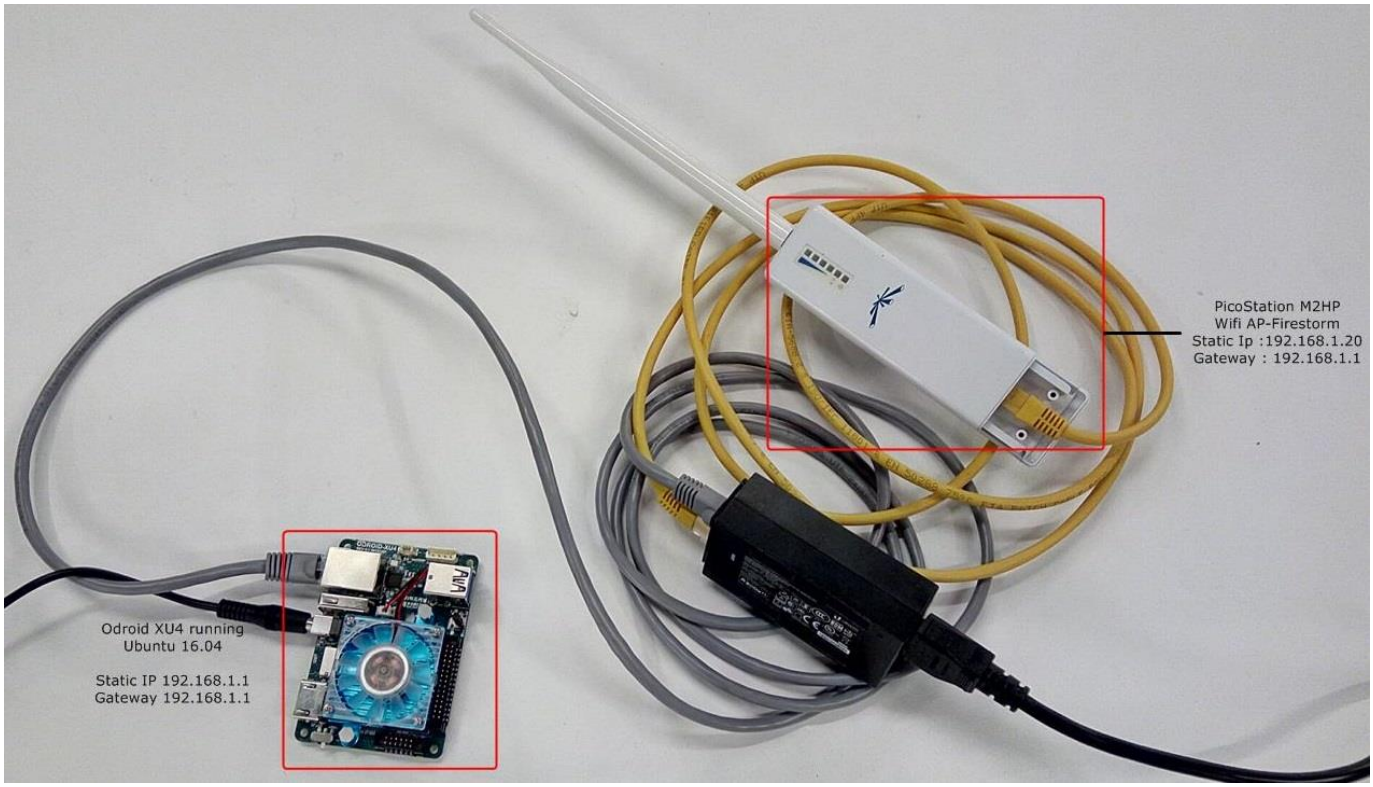
System Design

Our hardware has two broad sections:

a. Base station for ATC
b. Quadrotors

The base station comprised of the following:

a. **Odroid-XU4** - This is a powerful single board computer running Ubuntu 16.04. Mavproxy was used as the command line GCS package. Python scripts based on Dronekit API execute missions from the ATC.
b. **PicoStation M2HP** - This was used for creating a Wireless Access point (**Firestorm**) for broadcasting commands to the quadrotors. Picostation was connected to Odroid over Ethernet and hence Odroid communicated with the different quadrotors over wifi via Picostation.

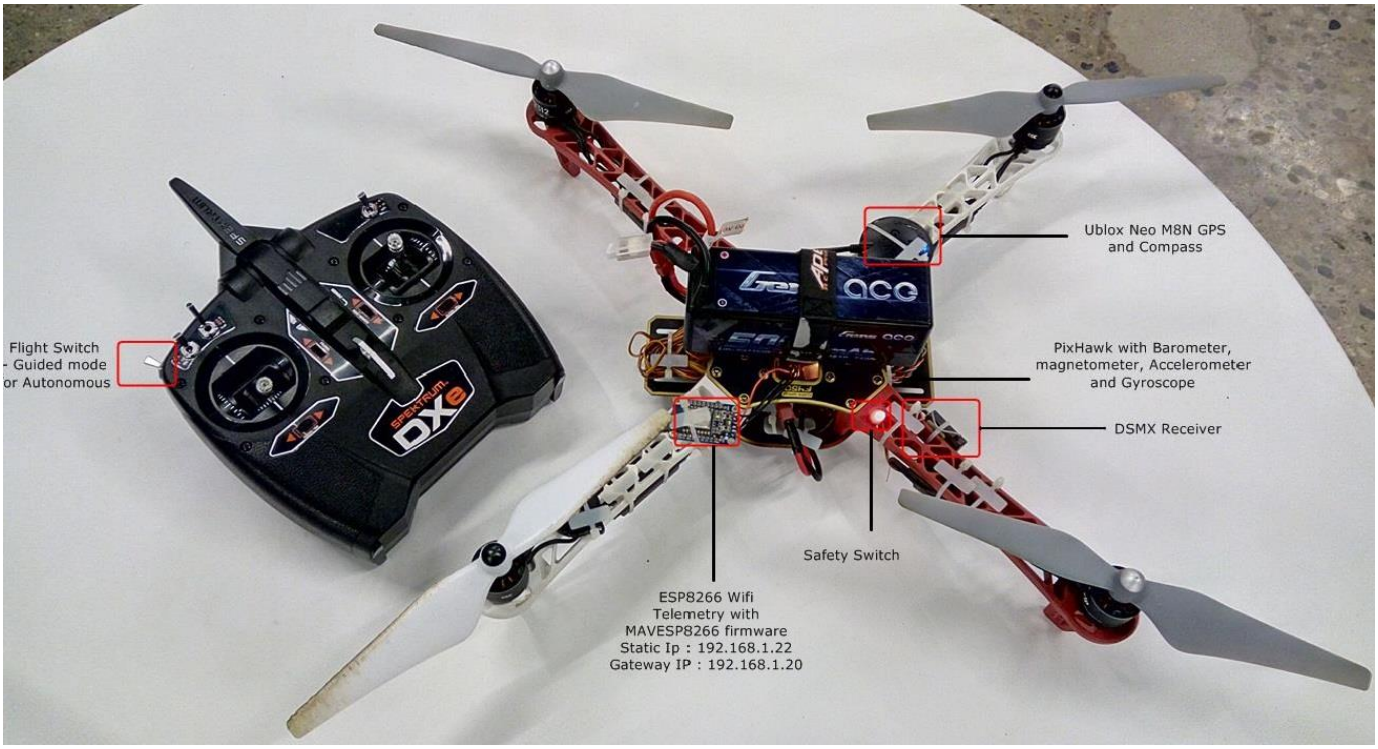


Each Quadrotor comprised of the following:

a. **DJI Flamewheel F450**: This was the 'X' type Quadrotor chassis used in the project. This consisted of four motors, four ESCs (Electronic Speed controllers) and propellers too. Each ESC connected to the Pixhawk on its PWM ports.
b. **PixHawk** : This was the Flight management unit running Ardupilot (APM). Pixhawk additionally came with a Power management unit (PMU), a buzzer and a LED cum safety switch. PMU connected to the battery and provided voltages to the ESCs and Pixhawk(converted). Pixhawk has internal sensors - Magnetometer, Accelerometer, Gyroscope and GPS.
c. **3DR uBlox GPS with compass kit** : This was an external GPS unit based on Ublox New M8N used for basic localization of the Quadrotor. This unit was connected to Pixhawk over a Serial(GPS) and I2C(compass) port.
d. **ESP8266** : These wifi modules were used for wifi telemetry with the Pixhawk over serial port. Each quadrotor was assigned a different static IP through which it connected to Firestorm accesspoint. ESP8266 was flashed with Mavesp8266 firmware to make it compatible with APM on Pixhawk.
e. **DXe Transmitter** : This was an RC used for manually controlling the quadrotors during initial testing and for contingencies. There was only one used for one ATC setup. More can be used depending on availability and other factors.
f. **DSMX Remote Receiver** : This was used on the quadrotor for establishing communication with the RC transmitter. This was connected to PixHawk over DSM port.
g. **Gens LiPo Battery** : 5000mAh, 4S Lithium Polymer battery was used for powering up the quads. These could last half an hour of flights in normal wind conditions.
h. Other items like extra propellers, LiPo batteries and battery charging units were also needed.

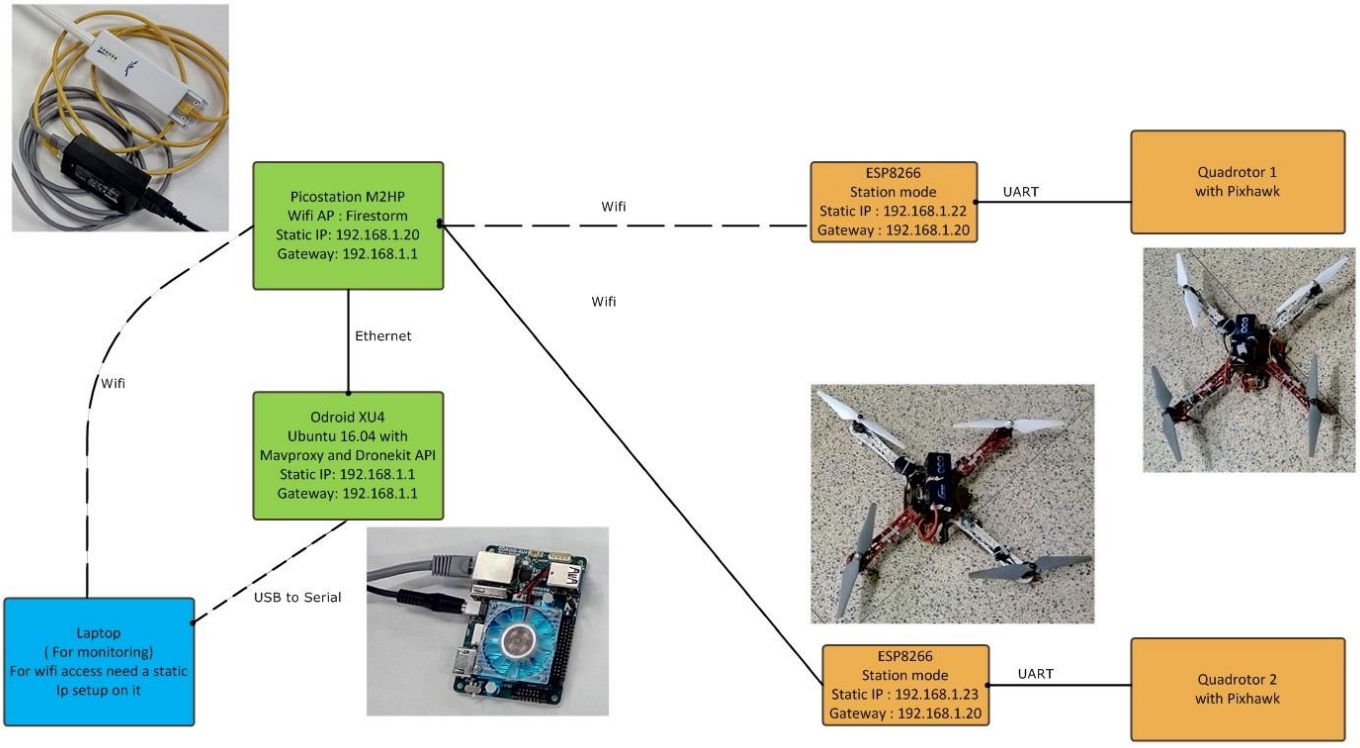
Software comprised of following sections :

a. Ubuntu 16.04 with **Mavproxy** and **Dronekit** API on Odroid-XU4
b. **Arudpilot (APM**) on PixHawk
c. **Mavesp8266** firmware on ESP8266 compatible with APM
d. Simulation environment was setup for testing different missions on Ubuntu 16.04 using **Ardupilot SITL simulator**
e. **Mission Planner**(Windows) was used for calibrating and configuring Pixhawks during initial setup of Quadrotors.



System Architecture

Following image briefly explains our System Architecture on a high level:



Each section of the system is discussed in detail in the development stages.

Setting up Simulation environment

Our initial aim was to get a complete understanding of the functioning of quadrotors and the basics of controlling and navigating them. To accomplish this task we set up a simulation environment using Ardupilot SITL(Software in the Loop). The communication between the virtual vehicle and simulated code takes place over MAVlink. We used Dronekit API to get access to the objects of Ardupilot.

The main challenge in this stage was to understand how MAVlink works and establish a successful connection between virtual rotor and simulation environment. We simulated several scenarios of mission planning by breaking it into stages :

a. Single quadrotor takeoff , navigate to a waypoint and then come back and land.
b. Parallel takeoff of quadrotors (4 nos) and landing.
c. Sequential takeoff of quadrotors (4 nos) and landing
d. Obstacle avoidance for a single quadrotor

After doing this our next task was to port this code to Odroid in Mavproxy environment to accomplish our intended goals. Python codes were written based on Dronekit API , which are available on our git repository.
The demo videos of the simulations on SITL are available on our Youtube channel.

The environment was setup on Ubuntu 16.04 considering the ease of scripting for mission planning in the simulation environment and also the support of Mavproxy and Dronekit API.
Detailed steps to setup SITL on Ubuntu are here.

Setting up Network

For this part of the project, we had to first setup Picostation as Wifi Access point and test its range. This had to be done since it will determine the range of operation of the ATC Basestation.
We test this by establishing wifi communication with an Adafruit Huzzah board(based on ESP8266) . The range outdoors was approximately 100m.

Picostation is setup in bridged access point mode with a static IP for the WLAN network :

Static IP : 192.168.1.20
Subnet IP : 255.255.255.0

Gateway IP : 192.168.1.1

Config file for setting up Picostation in this particular mode is available in Git.

In the next part, we complete the communication interface from Odroid to a wifi device (in our case ESP8266). So we set Odroid as the gateway(static IP 192.168.1.1) to which Picostation is connected as a wifi access point over Ethernet. Odroid is shipped with a Debian image, on which you can configure the static ip as :

a. Either connect the micro-sd card or emmc card to a Linux system or ssh into Odroid over ethernet if its powered on. User id is odroid and password is odroid.
b. Open interfaces file at this path : /etc/network/interfaces
c. Edit and add following lines to it to assign static ip :

```
auto eth0
iface eth0 inet static
address 192.168.1.1
netmask 255.255.255.0
gateway 192.168.1.1
```

d. Exit and save the file
e . Reboot the system with the new configuration. Odroid is setup to a static Ip of 192.168.1.1 and can seamlessly communicate over wifi via Picostation which was connected to its ethernet port.

Wifi Access point broadcasted by the Picostation was 'firestorm'. For any device to connect to this wifi , a static IP configuration had to be done.

Lastly, ESP8266 was configured to connect to this network. Static IP was 192.168.1.xx and Gateway IP was 192.168.1.20 (same as static IP of Picostation).

After all this setup, we were able to seamlessly communicate from the Odroid to any of the devices connected over wifi. Additionally, we could connect with our Laptop over 'firestorm' to ssh and monitor communication on Odroid.
We can also access Odroid over USB to serial interface when it is not connected to Picostation.

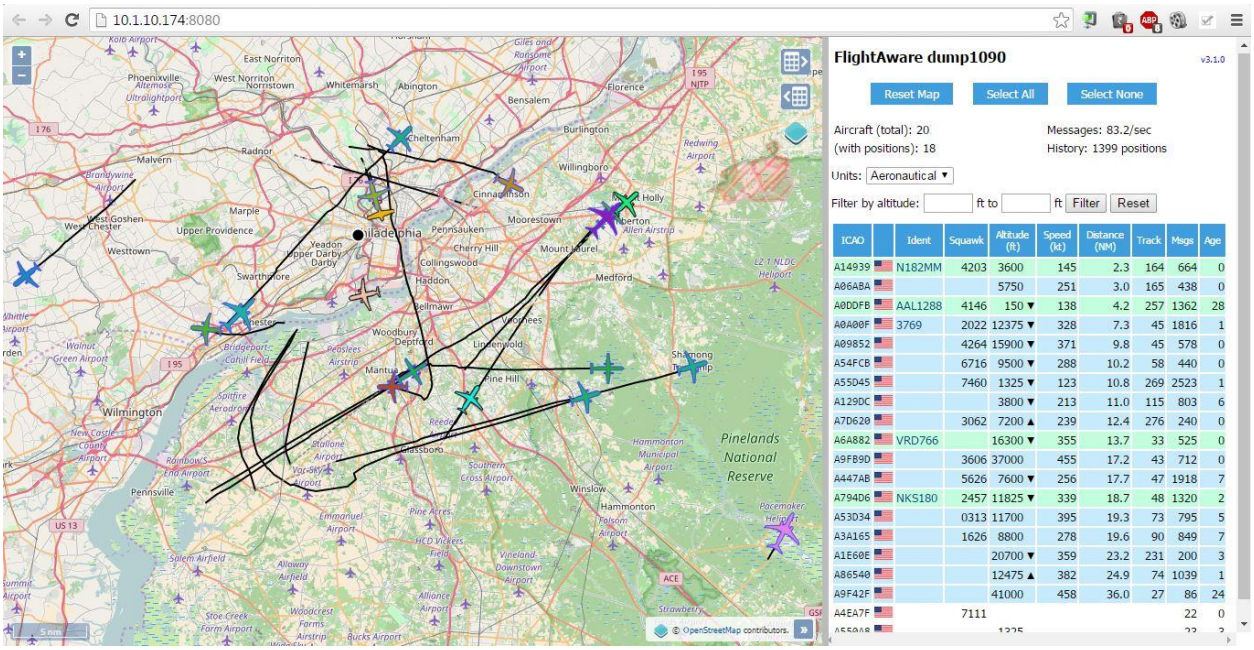
Flightaware system

Before we actually started with the quadrotors, we also got ourselves familiar with the behaviour of airlines in a airspace around a typical ATC, in our case Philadelphia. We chose to proceed with Flightaware system for that.

For setting up the flightaware following items were needed :

a. A raspberry pi board with a micro sd card having Piaware OS.
b. Flightaware SDR dongle
c. 1050MHz bandpass filter
d. 1050MHz magnetic mount antenna

Once the Piaware system was setup , it had to be connected to internet and then an account created to see the corresponding data logged by Flightaware for that particular setup.



Assembly and Calibration

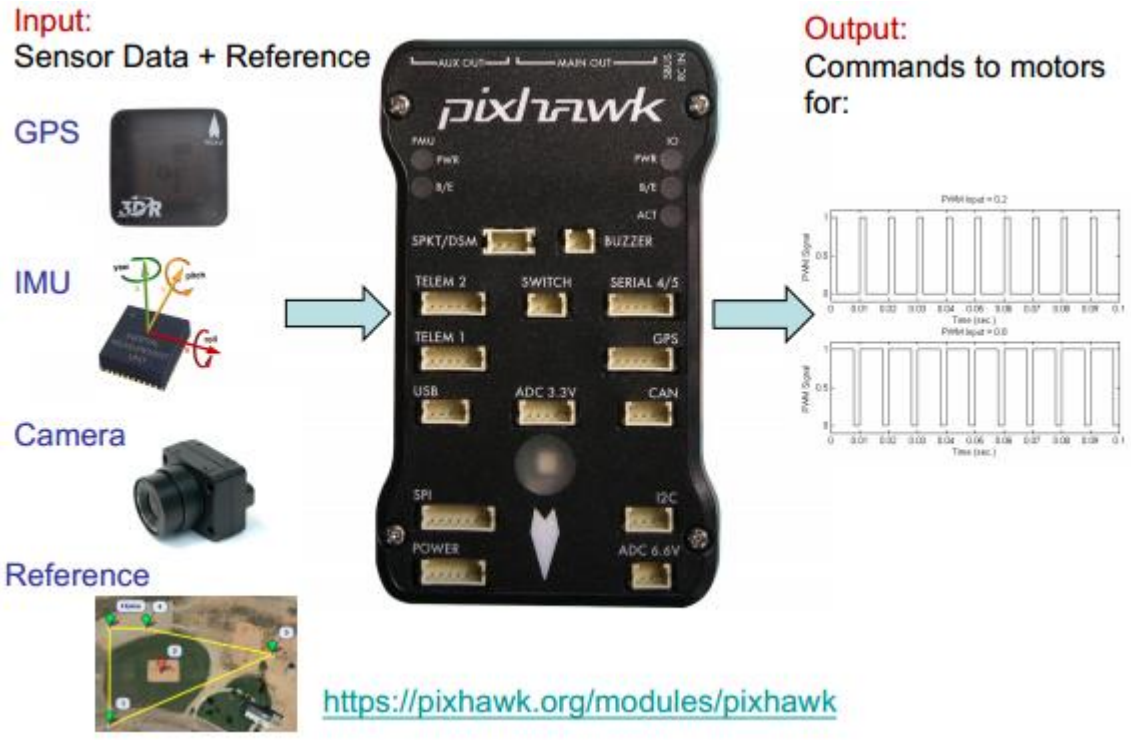
This section involves several stages through which we will cover all the steps required to prepare quadrotors for the first flight.

First stage was to assemble the quadrotor by following detailed tutorial videos available for the quadrotor chassis being used. Some broad steps common to all are:

a. Solder the ESCs on the baseplate
b. Mount the motors as desired by the chassis on the arms. Two motors are clockwise and two anticlockwise and they will support different propellers too.
c. Once motors have been fixed on arms, Arms and baseplate can be screwed together.

d. Mount Pixhawk in the exact centre of the baseplate with the "front" arrow on it pointing in intended front side of the quadrotor (decided wrt to order of motors).
e. Now connect other parts like the buzzer, safety switch, dsm receiver, wifi module, gps and compass modules and the power management unit to the Pixhawk. Wifi needs additional setup before connecting to Pixhawk (explained later)
f. Ensuring everything is connected properly and all parts firmly fixed, finish the rest of the frame and mount a charged battery securely with a velcro strap.
g. Remove propellers until the first flight.

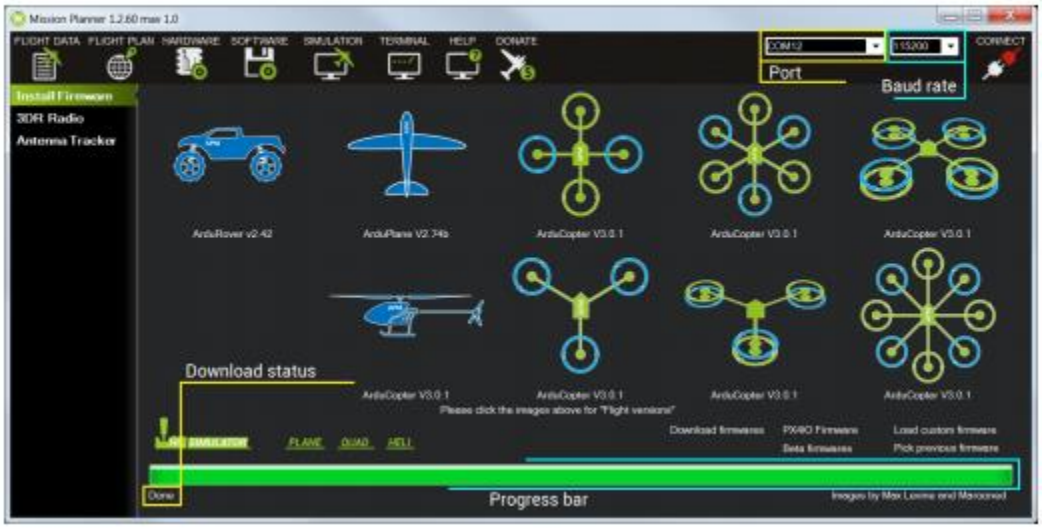
Next stage involved setting up the Pixhawk Flight management unit (FMU) with the firmware, GPS and Compass calibration, Accelerometer calibration and Motor control parameters. These settings were done in stages so as to ensure pixhawk was setup properly for the first flight.
Accelerometer, GPS and Compass calibrations had to be done several times later as their readings got disturbed during testing. All this setting up can be done using Mission planner on Windows. Detailed steps can be found here.



Ardupilot firmware on Pixhawk was configured with several failsafe mechanisms like it should land when battery reaches 10 percent. Geofencing was done wrt Launch

position so that it did not go too far. GCS failsafe was done so that quadrotor lands as soon as it loses contact with GCS. You can disable GCS failsafe later on.

Robot Selection and Sensor calibration



Source: http://ardupilot.org

DXe transmitter was configured correspondingly to accomodate for these failsafe settings and flight modes. The Transmitter was used for manual control for initial stages and as a backup in case autonomous control failed. The flight modes on the quadrotor were set to Loiter, Stabilize and Land using Mission Planner. All the ATC communication takes place in Stabilize mode.

ESP8266 was used for wifi telemetry with Pixhawk (TELEM2). The interface between esp8266 and pixhawk was UART configured at baud rate of 921000. After setting this up, Pixhawk could be connected to Laptop(Mission Planner/code) via both wifi and USB interface. Wifi interface was established over UDP port 14550.
ESP8266 was flashed with Mavesp8266 firmware for the above configuration. To flash esp8266, it has to be put in flash mode which can be done by connecting GPIO_ 0 to GND and then flashing the firmware over its serial port.
Detailed steps can be found here. Wifi module has to be set to station mode with static ip address once it is flashed with mavesp8266.

Each wifi module on the quadrotors is set with a different static ip and a different UDP client port and UDP host port. This enables Odroid to send specific commands for each quadrotor. For example, Quadrotor 1 has IP 192.168.1.22, UDP_CPORT 14555 and UDP_HPORT 14550 then Quadrotor 2 will have IP 192.168.1.23, UDP_CPORT 14565 and UDP_HPORT 14560 and so on.

First Flight

After all the configurations were done and motors tested with the RC transmitter. We first had to ensure that it is able to arm the quadrotor before its first flight. With the help of the RC transmitter (DXe transmitter) Quadrotor had its first flight in 'Loiter' flight mode. In this we checked for the stability and drift of the rotor. We had to make some minor changes to our component placement to remove the drifting issue.

Then we programmed waypoints on the quadrotor using Mission planner GUI and tested whether it is able to navigate through those waypoints as desired. We observed major issues with GPS location update and Altitude update. Under windy conditions, these got even more difficult.

Command line control of the quadrotor

Next, we connected the rotor to laptop using WiFi and accessed it using mavproxy on laptop.

We were able to have the rotor takeoff and land using commands send from the laptop. The main challenge in this stage was to get the onboard ESP8266 to be compatible to ardupilot. For this we needed to flash MAVESP8266 firmware on ESP8266 and after struggling and trying out some different methods we were able to have a connection established between rotor and laptop.

After this we tried one of our baseline goals to give waypoints to the quadrotor from the commandline and then takeoff and see it follow those commands. We were able to successfully do this after several tryouts with GPS and altitude fixes done properly.

Controlling Quadrotor via ATC

The next stage was to have our ATC setup communication with the quadrotor.

After setting up seamless communication between Odroid and the quadrotor, we ported our software configurations from the Laptop to Odroid and executed Takeoff and landing procedures.
ATC configuration and setup is discussed in detail under 'setting up network'.

Controlling multiple Quadrotors via ATC

Once we were able to reliably test our ATC communication and control over a single quadrotor we extended it to the second quadrotor for our demo.
Keep in mind about the different UDP port configurations assigned for each of the wifi modules on the quadrotors and the IPs associated with them.

GUI for ATC missions

A simple GUI with some missions was created which can be then used to execute certain missions from the ATC.

Challenges we ran into

a. Major challenge for us was to understand the basics of quadrotor and to control it in a limited time frame.
b. Next we had to get the simulation environment up and communicate with the SITL which in turned required understanding of Ardupilot and MAVLink protocol.
c. To have complete understanding of various python objects used to control the rotor and be able to write code for the rotor.
d. On the hardware front, we faced several issues with GPS and compass calibrations failing while we tested and quadrotor landed imperfectly. This took

some figuring out and also manual control using RC had to be kept as a must for all our testing.
e. During our initial phases, it was challenging to setup a seamless communication between pixhawk and the odroid (basestation) , at the same time monitoring odroid.
f. Lastly, power was also a concern. Once batteries reached a certain level, quadrotors have to be landed or asked to return to launch position

Accomplishments that we're proud of

a. We have a complete simulation environment setup to test our codes and also try out different trajectories and strategy which turned out to be quite helpful because direct testing on actual quads can damage the quads and prove expensive.
b. We have a working quadrotor with full calibration that can be controlled by RC Receiver, laptop and Odroid(remotely).
c. We have testing videos of us controlling quadrotor through laptop and Odroid by giving it commands to takeoff, land and navigate through waypoints.

What we learned

a. Setting up network interfaces for different configurations
b. Flying a Quadrotor involving various stages of assembly and configuration
c. MQTT server setup for setting up messaging between quadrotors and basestation but was not required later as it turned out incompatible with Mavlink
d. Mavlink protocol communication
e. Simulating codes written for actual testing environment using Software in the loop simulator, which saved time and money

What's next for Autonomous Air Traffic Controller

a. Proper implementation of trajectories to be followed by quadrotors.
b. Autonomously controlling four quadrotors in a controlled environment to simulate several flight scenarios with an ATC.

Conclusion

This project taught us a great deal about Quadrotors and how they function. Our system involved mostly any calibrated and functional Quadrotor interacting with our

ATC communication setup around MAVproxy which uses MAVlink protocol to communicate with the Flight management units onboard Quadrotors.
We were able to simulate quadrotors for various missions demonstrating real time scenarios like obstacle avoidance and parallel takeoffs and landing. We then established a reliable communication channel between a standalone computer and quadrotors over Wi-fi.

In the end we were able to realize a portion of autonomous Air traffic control intended for UAVs, partially completing our reach goals.

From here, this project can be taken in many different directions with use of even more efficient radio telemetry and addition of splines for path planning of the quadrotors. While not all of our reach goals were met, we view it as a success as we provide a platform with scope for further development.

Important Links

Blog : https://devpost.com/software/autonomous-air-traffic-controller-pg62u5
Code : https://github.com/modi-karan/Autonomous-Air-Traffic-Controller
Demo Video : https://goo.gl/BBe7dL, https://goo.gl/5FxJXI