Flying Access Points

Johan Westlund
jwestlu@kth.se
0702527856

Haider Farooq
haiderf@kth.se
0735295987

Sebastian Nilsson
senil@kth.se
0707604441

Fredrik Lindblom
flindblo@kth.se
0730884983

Mattias Durovic
durovic@kth.se
0704194519

Abeer Akhtar
abeera@kth.se
070512687

Sharan Yagneswar
sharan@kth.se
0764150869

Mina Tawfik
mtawfik@kth.se
0727242185

José Luis Bismarck Fuentes Morales
jlbfu@kth.se
0728594454

December 21, 2015

Abstract

The purpose of this project was to develop a prototype flying access points for KTH and Ericsson's Integrated Transport Research Laboratory (ITRL), to be furthered into a fleet on future stages and integrated with other projects at ITRL. Our hardware consisted of a jDrones quadcopter with a Pixhawk flight controller and an Xperia M2 mobile phone as our processing component for Wi-Fi connectivity and cloud communication.

A group of nine students have under six weeks designed and build a working prototype to show that the concept is viable.

Contents

1 Introduction

1.1 Purpose

This report describes the final results of the Flying Access Point project carried out by the authors on behalf of Ericsson.

The project goal was to extend a quadcopter drone provided by Ericsson to allow it to carry out two new functions. The first was to allow the drone to act as an access point providing WiFi access to nearby users. The second was to make it possible to remote control the drone (and eventually a fleet of such drones) from a central location. For more detailed requirements see section 2. Note that the project did not cover the legal aspects of operating such a drone since they vary between countries.

The purpose of this flying access point is to provide connectivity in places where normal infrastructure does not reach or provides insufficient capacity. This could for example be the case in a rural area where connectivity is needed only sporadically. Or it could be a means of increasing available capacity during a large peak, say during a sporting event or covering a resort only during vacation season. The system could also be used to rapidly restore connectivity during an accident or disaster where normal infrastructure has been damaged.

The mobile nature of the flying access point could also be taken advantage of to provide dynamic coverage of areas based on usage projections. For example the bus stops in a city could be covered only during those times (rush hour, popular events, and so on) when a large number of commuters are expected to be present. This dynamic allocation could even be used to respond to unexpected usage peaks in real time by using a high speed quadcopter.

We note that while the project developed a system that provides WiFi connectivity using cellular infrastructure it could just as easily provide other forms of connectivity or use other types of infrastructure to connect to the network backbone. All that is required is a sufficiently lightweight transmitter and receiver.

Control of the drone fleet is to be carried out from a central server residing somewhere on the network. Drones are parked in some central location or locations from where they can be flown out in response to requests for coverage. The central cloud server determines which drone to send, where to send it, and how long it should remain there. The drone needs only to fly to the provided coordinates and activate its on-board access point while keeping the cloud server updated on its current status.

This cloud server is to be developed by a separate project group. The requirement for this project is only to ensure that the drone is capable of following the received flight plan, act as an access point when it reaches its destination, and communicate its current status to the cloud server.

Assuming there was time remaining once the above was completed Ericsson

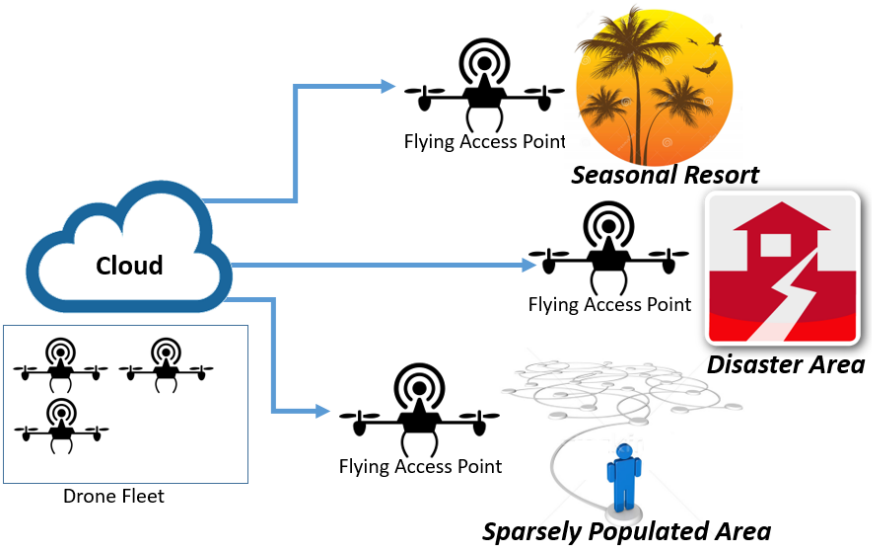

Figure 1: A conceptual sketch of some of the potential uses of the flying access point system. A central server residing in the cloud and controls a fleet of drones parked at central locations.

also wanted the team to explore three possible extensions to this system. The first of these was to explore the possibility of having the drone ride along on commuter buses as a means of extending the drone range. The second was to implement a WiFi captive portal on the drone hardware to allow better access control to the provided network, for example to only allow access to paying users. The third was to survey possible solutions for autonomously recharging the drone batteries without human intervention so as to allow the cloud server to automatically rotate a set of drones and provide connectivity over an area for longer periods of time.

Finally it should be noted that Ericsson wanted the flying access point to be developed using commercially available components. Preferably those available as open source and open hardware.

1.2 Existing solutions

In the bigger picture, since the advent of quadcopters for the general consumer public, not many solutions exist which envision the possibility of using them as Wifi-hotspots. This can also be attributed to the fact that it is a fairly new technology. One such solution exists where an hobbyist assembled a drone with a LTE -Wifi router attached [1], but it was neither stable nor capable of autonomous flight. No complete commercial solution exists however. Examples of other variants of this solution do exist in the form of for example the usage of military drones for providing Wifi connectivity for soldiers [2], Google's Internet balloon [3], and Facebook's Internet Drone [4].

Both Google's and Facebook's solution are aimed at providing Internet con-

nectivity in remote locations, designed to fly continuously and the latter two are still in the conceptual stage [4]. These solutions are ridiculously expensive, would require months of research, and do not cover the specific use case we were requested to fill. What is needed is a quick way of deploying Wifi connectivity on demand.

However this is only the case for the full platform. The various subsystems necessary to assemble the flying access point are common on the market and in the development community. What follows is a breakdown of the current state of the art of the individual subsystems needed to assemble the full flying access point system.

1.2.1 Flight Controller

Quadcopters are mainly fly by wire devices which means they need a computer to control them. A flight controller is a combination of hardware and software which controls the craft. It consists of a microcontroller, sensors, and other peripherals combined into a single circuit board. There are many such controllers on the market, some of the more common ones can be found listed at [5, 6].

- Naze 32
- CC3D
- KK2
- Ardupilot 2.6
- PixHawk

The existing solution provided was an **Ardupilot APM 2.6**, and was already better than the Naze32, CC3D, and KK2 since it had more flash memory, computation power, and hardware capable for autonomous flight [5]. However it was decided to replace this with the latest **Pixhawk** because of its better specifications. It features the following advantages [7].

- Faster and more powerful than APM 2.6
- APM stops receiving new updates because of slower hardware.
- Rugged enclosure with strong connectors.
- Memory card support for data logging.
- An USB device can be plugged into the USB port while the Pixhawk is powered via battery. On the APM this would lead to overcurrent and a short circuit[8] and this particular functionality was required during development.

1.2.2 Firmware

The firmware which runs on the flight controller is usually open source [6]. **Cleanflight** is an example which can be run on the Naze32 controller but lacks autonomous flight functionality. The **APM Arducopter**, compatible with APM 2.6 and Pixhawk, supports autonomous flight functionality and is open source as well. It was chosen for this project.

1.2.3 Frame

The frame supports the entire structure of the drone [6]. The number of rotors and the weight its designed to take are the two main parameters when deciding on the Frame. While there are many available in the market, the existing frame from the JDrone provided by Ericsson was retained because it was deemed suitable for the project.

1.2.4 ESC Units, Motors and Batteries

ESC units stand for Electronic speed controllers. They translate the PWM signal from the flight controller to the actual currents being fed to the motors [5, 6]. The ESC's are rated on the current they deliver to the motors. The motors draw more current for more torque produced and the existing solution provided with the Jdrone had 30 Amp motors and hence 30 Amp ESC's. To provide such current high discharge rate Lithium polymer batteries were provided with the Jdrone. Though these are more powerful then required it was decided to retain them as they came as part of the recommended frame design.

1.2.5 Propellers

The propellers are designed to provide enough lift for the weight of the frame. The three parameters which characterize them are attack angle, width, and material. Various solutions from carbon fiber to plastic exist. The more expensive ones being better balanced which exposes the frame and sensors to less vibrations [5].

1.2.6 Landing Beacon

Since landing with GPS does not always result in high precision there exists automatic landing solutions which are compatible with the Pixhawk firmware. The IR lock [9] is a system comprising of an IR camera mounted on the quadcopter and a set of IR beacons to be placed on the desired landing location. The IR camera then guides the flight controller to home in on the beacon allowing for very precise landings.

1.3 Challenges

The primary challenge envisioned during the planning stages of the project was the need to automatically land precisely in a desired location and specifically one whose coordinates where not exactly known. This would be necessary to allow the drone to ride along on a commuter bus since the there is limited space on top of the bus in which to land and since it is not exactly known where the bus will stop relative to the bus stop.

In reality this turned out to be fairly simple since an existing system known as IR-lock has already been developed to allow for precise landing of quadcopters and other multirotor craft. What actually proved to be the greatest challenge of the project was the communication between the cloud and the drone autopilot.

The reasons for this was that the team needed to use a commercially available autopilot solution. While the team did confirm that the autopilot selected did in fact have a protocol for communicating with a remote computer during flight it failed to notice that the documentation for this protocol was both incomplete and outdated. Successful implementation of the cloud to autopilot communication, a task expected to require the work of two team members for at most two weeks and be completed by early November instead left these two team members working until the final week of allocated project time.

1.4 Design Advantages

To the best of our knowledge this is the only example of a quadcopter capable of receiving navigation commands from a network location during flight. While there are systems that can be pre-programmed to follow a specific flight path they all require that a local operator first program the flight plan into the autopilot and then manually activate the quadcopter and tell it to initiate flight using a direct radio link.

In addition the attachment of a WiFi access point to a drone is also uncommon though also a fairly trivial accomplishment.

2 Requirements

Ericsson provided a set of requirements divided into three distinct phases.

Phase 1 consisted of implementing the base functionality of the flying access point. This included the ability to fly to waypoints according to commands sent from the cloud and the ability to act at a WiFi access point. Phase 2 involved extending the prototype to allow it to ride along on commuter buses (or similar) to conserve battery power and extend range. It also included developing a captive portal for the access point functionality. Phase 3 finally required further extensions to allow automatic drone recharging. It should be noted that Ericsson only required phase 1 functionality in order to consider the project a success.

From these feature descriptions the team developed a set of requirements for each phase. Phase 1 and 2 requirements are listed below along with their completion state at the end of the project. Phase 3 requirements are not listed since no work on phase 3 functionality was carried out due to time constraints.

2.1 Phase 1 Requirements

As stated phase 1 required the base functionality be implemented. The team translated this to three requirements. One relating to the drones ability to fly to the target location. Another relating to the ability to communicate orders to the drone from the cloud. And the final one relating to its ability to act as a WiFi access point.

2.1.1 Flight Requirements

The requirement set out by Ericsson was for the drone to take off, fly to the desired location, and land. The team translated this to three main requirements, that the drone be capable of: taking off, generate and follow a flight path, and land successfully on the target destination. All with enough precision to be used on outdoors environments and an area the size of a bus as a potential landing target. Compliance to flight requirements was tested verified as follows:

1. **Lift Off**: We tested the drone's lift off on multiple surfaces with multiple characteristics, including grasslands, concrete and uneven ground, the default features of the flight controller allowed our quadracopter to lift off in most situations, only excluding those when the lift off surface resulted in radically uneven thrust distribution, such as the edges of tables or at very small platforms.

2. **Navigation and path planning**: For this requirement we tested several kinds of routes as missions, we did not include obstacle avoidance nor 3d routes due to our constrained time to deliver, but we were able to complete a satisfactory 2d navigation by using the tools provided by the Pixhawk flight controller.

3. **Landing**: The baseline GPS and inertial navigation compliance was perhaps the most difficult and inconsistent parameter to test, as multiple external factors affect GPS acquisition time and accuracy, including the weather, satellite position, time of the day and other geographical factors. We tested our accuracy by sending the drone to multiple missions to GPS coordinates and measuring the error between the target coordinates and the manually acquired GPS coordinates measured at the place of landing via an external mobile device. Since all the quadcopter's GPS, the measuring phone's GPS and Google Earth's geodetic GPS model have errors and discrepancies the error calculations are merely theoretical approximations, see 3.6 for more information.

2.1.2 Cloud Communication Requirement

The requirement set out by Ericsson was that the drone be controllable from a server residing in the cloud. For this to work the drone would need some way to connect to an internet server and then receive commands from it. In practice the imperfect nature of wireless connectivity also meant that the drone must operate properly even if the cloud server becomes temporarily unreachable at any point.

This requirement was fulfilled and verified. The pixhawk flight controller includes functionality that allows it receive commands trough a serial connection. To meet this requirement the access point hardware platform was connected to the serial port of the pixhawk and the software required to relay commands from the cloud server to the pixhawk was developed. It was verified by sending messages and receiving answers. We also verified the content in the mission list after sending a mission with USB-cable and MissionPlanner. It was verified that data had arrived correctly.

2.1.3 WiFi Access Point Requirement

The requirement set out by Ericsson was that the drone be capable of acting as a WiFi access point providing internet access to any WiFi clients in range. Note that for this phase of the project no user management of access restrictions were required.

This requirement was fulfilled. The required functionality was provided by the Sony Xperia M2 used to provide connectivity. Testing was limited to the range and bandwidth as we determined that the developers of the Xperia had conducted other necessary tests. These tests were conducted through the use of reliable free software.

2.2 Phase 2 Requirements

As noted phase 2 required that the flying access point be extended in two ways. The first and simpler of these was the implementation of a WiFi captive portal. This was translated into a single design requirement. The second was to allow the drone to ride on top of a commuter bus so as to increase range. The team translated this into two different requirements. That the drone be physically capable of landing with sufficient precision to land on a commuter bus roof and that the drone (or cloud) have the necessary logic to make the drone successfully determine when the commuter bus is in position, land on it, and then decide when to take off from it. Note that determining which bus to ride and at which locations to get on and off was not part of the requirements for this project.

2.2.1 WiFi Captive Portal Requirement

The requirement set out by Ericsson was that the flying access point be capable of redirecting new users to a captive portal. By extension this also implies that users who have visited the captive portal no longer be redirected but instead be allowed to access the internet as usual. Note that since Ericsson did not require any actual user management capabilities the team chose not to implement any such logic.

The work with the captive portal was pushed back until the end due to delays in meeting the phase 1 requirements. This led to it not being fully implemented in the end. The captive portal does possess the ability to successfully redirect incoming traffic to a website but we did not manage to create a website where users could login. Much of the logic which would be used by this website to list users that can access the internet without being redirected has been implemented but not properly tested.

2.2.2 Precise Landing Requirement

Testing showed that the landing procedure developed during phase 1 had an imprecision of several meters. This would not allow the drone to land successfully on top a commuter bus. As such a requirement for the commuter bus system was to improve landing accuracy.

To allow for precise landings the IR-lock system was added to the flying access point. This consists of an infrared beacon that is placed at the desired landing location and a infrared camera mounted on the drone. In theory this should allow the drone to land on top of the beacon with very little deviation. Unfortunately time and circumstances did not allow the system to be tested. As such the requirement can not be considered fulfilled since the team does not know if the system works and then if it provides sufficient precision.

2.2.3 Bus Ride Along Requirement

The requirement set out by Ericsson was that the flying access point be capable of riding on top of a commuter bus to conserve battery power and extend its travel range. Since development of the cloud logic responsible for path planning for the flying access point was part of another project the team only needed to produce the necessary logic to determine when the commuter bus was present at the starting point of the ride along and when it had reached the point where the drone was to get off the commuter bus.

This requirement was not met. The required logic was not developed. Since phase 2 goals were not required by Ericsson to consider the project a success the project owner had no objections when notified of this.

3 Control Logic

3.1 Overview and theoretical background

The Multi- rotors drones are simple where they have n number motors and n number of propellers and can fly and move only by changing the motor speed. Drones are controlled by an electronic or computer based system. For a drone to fly smoothly, it's important that the flight controller is fully tuned.There are four basic conditions for a perfect drone flight that needs to be fulfilled [18]:

1. Equilibrium of forces: $\sum_{i=1}^{4} T_i = -mg$ (weight of the quad-rotor) (A)
2. Equilibrium of directions: T_1,T_2,T_3,T_4 and g (T_i: forces generated by the propellers) (B)
3. Equilibrium of moments: $\sum_{i=1}^{4} M_i = 0$ (Moments generated by the forces, M_i=L x T_i where L is the distance from rotor to Center of gravity (COG)) (C)
4. Equilibrium of rotation speeds: (ω_1+ ω_3) - (ω_2+ω_4) = 0 (ω_i: Rotation speed of the propellers) (D)

3.1.1 Reference System

There are two reference systems, which are very important in-order to detect any difference in the above four mentioned conditions [18].

1. The inertial reference systems, i.e. the earth frame (xE, yE, zE)
2. The Quad-rotor reference system, i.e. the body frame (xB, yB, zB)

3.1.2 Euler Angles

There are three angles (ϕ, θ, ψ) define the transformation between the reference systems:

1. Roll,ϕ: Angle of rotation along axis xB ||xE
2. Pitch,θ: Angle of rotation along axis yB ||yE
3. Yaw,ψ: Angle of rotation along axis zB ||zE

3.1.3 Angular Speed

The derivative of (ϕ, θ, ψ) w.r.t time are the angular/rotation speeds of the systems are called Roll rate, Pitch rate and Yaw rate. If the drone met last three conditions B,C and D, which are mentioned above then Roll rate, Pitch rate, and Yaw rate will tend to 'zero'. The first condition A does not needs to be fulfilled, where by increasing or decreasing the rotation speed of all the propellers we can go up and go down as illustrated below:

- Go Up: $\sum_{i=1}^{4} T_i < -mg$

- Go Down: $\sum_{i=1}^{4} T_i > -mg$

Where as Euler angles and rates remain 0.

3.1.4 Rotation

If there is no equilibrium in propellers speed such as:

- $(\omega_1 + \omega_3) - (\omega_2 + \omega_4) \neq 0$ (1) [18]

It will result as angle of rotation along z-axis or yaw rotation. The equation of Rate of yaw will be:

- $\psi = kY((\omega_1 + \omega_3) - (\omega_2 + \omega_4))$ (2) [18]

Since there is difference in the propellers speed, equilibrium of moments also changes and causes rotation along x-axis or Roll Rotation. Unbalanced propeller speed given as:

- $(\omega_1 + \omega_4) - (\omega_2 + \omega_3)) \neq 0$ (3) [18]

Where as the equation of Rate of Roll will be:

- $\phi = kR((\omega_1 + \omega_3) - (\omega_2 + \omega_4))$ (4) [18]

The effect of equation (4), changes the equilibrium of directions 'Ti', as a result 'Ti' will not remain parallel to 'g' and cause in both Roll Rotation and Translated flight which will also result in the rotation along y-axis or the Pitch Rotation. The equation of Rate of Roll will be:

- $\theta = kP((\omega_1 + \omega_2) - (\omega_3 + \omega_4))$ (5) [18]

As a result of equ. (5), Pitch Rotation will decrease the total thrust therefore need more power w.r.t for hovering or yawing.

3.2 Pixhawk Sensors

To control the quad-rotor we use the Pixhawk as a flight controller, which impose a certain rotation rate of axis in the body frame to set the 'ω_i' through P.I.D controller. Due to the uncontrolled variables i.e. air density and wind, it's not possible to have direct control on 'ω_i'. In-order to provide proper tuning Pixhawk use sensor approach as a feedback where it compares the measured value with desired set point and apply correction to the system on the basis of the error. Thus it ensures a smooth flight even in the presence of uncontrollable variables such as air density, wind, etc.

The sensor approach requires calculating rates and angles through the sensors to correct error. The actual angular velocities on the three axis $(\dot{\phi_m}, \theta_m, \dot{\psi_m})$ can be measured by 3-axis gyroscope where as set desired angular velocities on the three axis $(\dot{\phi_T}, \theta_T, \dot{\psi_T})$ are given by the remote control. Since the rate are not angles, e.g. a roll rate of $\phi = 10$ ° but no rotation (rate), i.e. $\phi = 0$ and no input from the controller, result in quad-rotor not being horizontal and tend to perform a translated shift. [18]

3.2.1 Measure Angle by Gyroscope

Angles can be measure from both Gyroscopes and Accelerometers. The angle measured from Gyroscope is affected by approximation errors and offset error, i.e. they give non-zero value when it should be zero, hence estimated angle is not reliable.

3.2.2 Measure Angle by Accelerometer

We can measure tilt angles through Accelerometer on the three axis (a_x, a_y, a_z) reliably only when they are static and not moving. The angles can be calculated using the following formula [18]:

- $A_x = \arctan\left(\frac{x}{\sqrt{x^2+z^2}}\right)$ (6)

- $A_y = \arctan\left(\frac{y}{\sqrt{x^2+z^2}}\right)$ (7)

However these angles are not reliable because accelerometer is sensitive and prone to vibrations during flight.

3.2.3 Sensor Fusion

The angle measurement from Gyroscope and Accelerometer can be combine in order to adjust the error and obtain reliable information. Hence pixhawk does this using Kalman filters through the following equation:

- e(t) = real(t) – estimated(t) (8)

Thus the error signal (differences) are sent to PID controller whose output are the target rates for the feedback rate controller shown in Figure 2.

3.3 P.I.D

Proportional job is to correct any error. If the multi-rotor is supposed to be horizontal and one side is down, then proportional job is to push that back up as quickly as it can. The amount of 'Proportional' (P) to dictate how hard it tries to get back to where it was. On the flight if the P is to high, it will try to hard to come back to the level and result in overshot, then it will try to correct the error again and undershoot. Hence P will leave an offset error and Integral (I) is required to estimate this error. In flight controller I helps to maintain the require attitude. If its trying to fly forward and result nose is rising all the time. It's because, this little sustained change which is letting nose drift up not being taken care by the integral part of the PID as a result 'Derivative' (D) get off. D is the speed at the P and I work out. If P is how hard is try to get back the place it supposed to be, I is there to correct the sustained the differences where its supposed to be and D is how quickly how PID loops try to get back there.

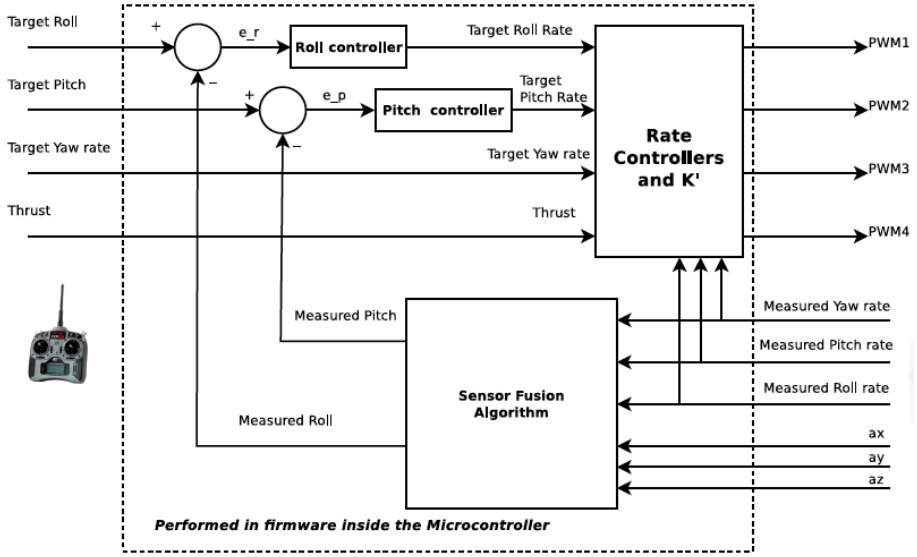

Figure 2: *The feedback control take the Estimated Targets Rate Roll, Pitch and Yaw from the Gyroscope and Measured Rate Roll, Pitch and Yaw from the Accelerometer respective. Then errors are corrected by the Sensor fusion using Kalman filter and Through P.I.D gains are added to output of all four rotor. i.e PMW1, PMW2, PWM3 and PM4. [18]*

3.4 Software in the Loop (SIL)

Once uploading theoretically derived control parameters to the flight controller, prior to testing them on the physical hardware we tested them on jMAVSim, the Pixhawk/PX4 software in the loop simulator. Given the physical constraints of our aircraft, jMAVSim runs a physics engine and performs a 3D simulation of our quadcopter as seen in Figure 4. Using the actual Pixhawk flight controller, connected via USB, it controls a virtual drone, which will simulate all hardware interactions. This proved to be immensely useful for tuning purposes and for figuring out how many of the Pixhawk's were intended to work.

3.5 Tuning

3.5.1 Basic Tuning

After the first flight it was observed that the quad copter does not respond to the control inputs very effectively and there was a bit of trouble maintaining control. Some basic tuning was implemented for the effective control of quad copter.The copter was attached to the ground computer and in APM planner (Config/Tuning) Basic Tuning was selected to start the basic tuning of the copter. This tuning is limited to very general and subjective heuristic and had to be improved upon on later stages. The Roll/Sensitivity bar was used to change the response of the copter from sluggish to twitchy. The process was repeated until a stable response was achieved. The bottom two sliders could be used for throttle control.

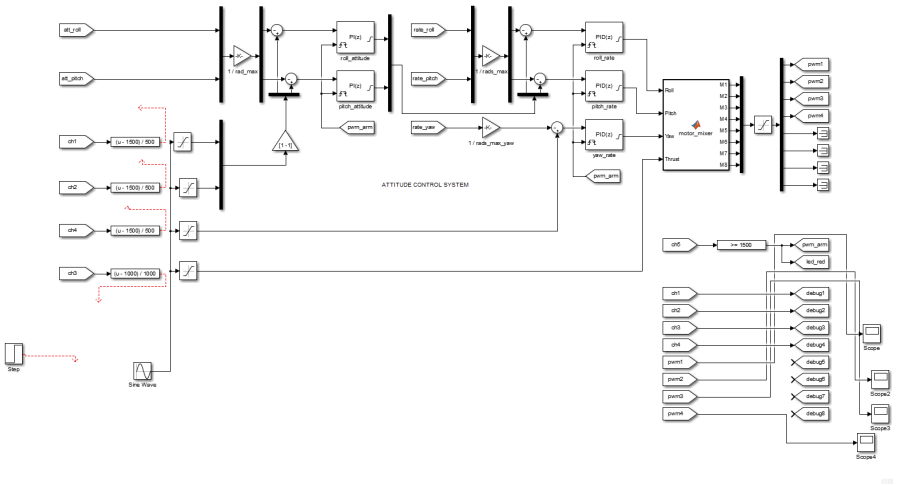

Figure 3: Screen capture of the Simulink Pixhawk block model

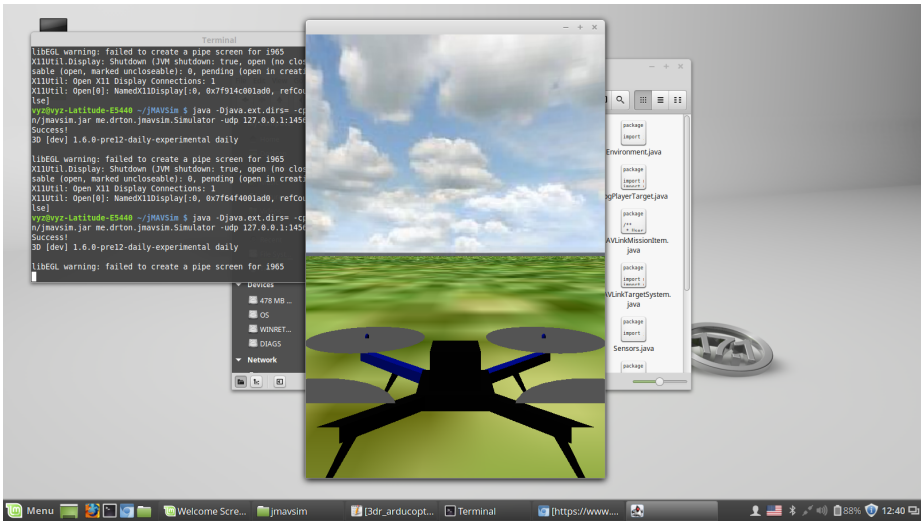

Figure 4: Screen capture of the software in the loop simulated 3D model of our quadcopter executing a mission

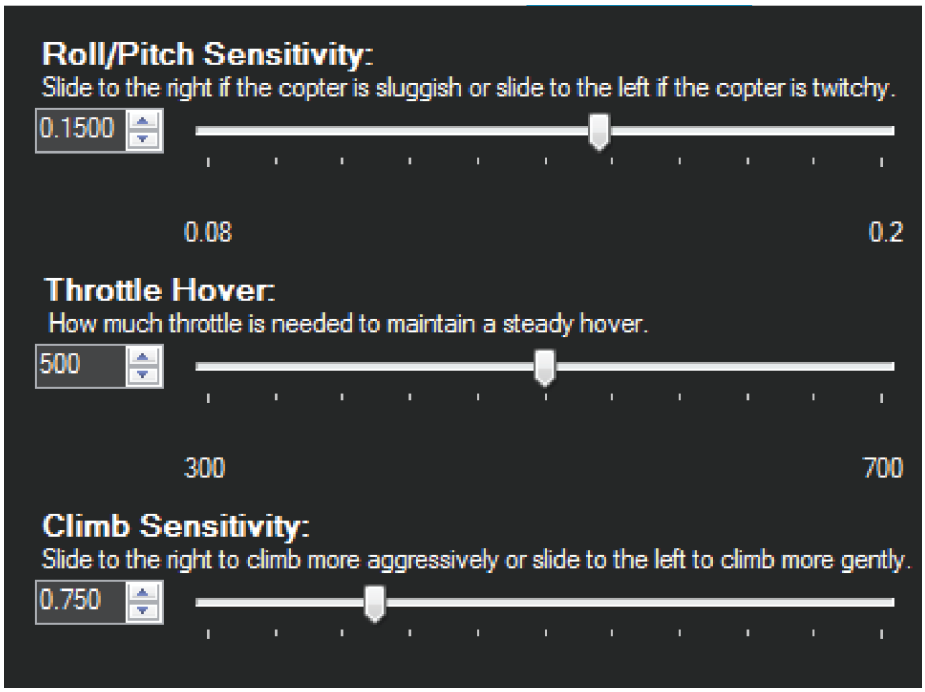

Figure 5: Screen showing the basic tuning parameters to be performed with live telemetry tests

3.5.2 Pixhawk Autotune

With the default PID setting most of the radio controlled vehicles can safely fly using Pixhawk but in order to meet the precision required by our application we required custom tuning parameters, which we attained using the Pixhawk's auto-tune protocol.
The goal is to provide highest response without significant overshoots. The copter is needed to be flyable in Altitude hold mode before using the auto tune mode as the copter need to be twitched in roll and pitch axis. The process takes each rate controller through the following actuator tests:

1. Set Rate I and D to zero.
2. Request a 90 deg/sec roll and depending on the results it chooses an estimate Stab P value.
3. Reduce P rate until no overshoot is visible.
4. Increase Rate D until a bounce-back of 10 percent is detected in the actual rotation rate.
5. Increase Rate P until it starts to detect an overshoot in the actual rate.
6. Commands a 20 deg angle and increases Stab P until it sees a 10 percent overshoot in the actual angle.
7. Finally Stab P is reduced by 20 percent for a more conservative result.

Table 1: Parameters obtained via Pixhawk auto-tune

Stabilize Roll P	4.5	Stabilize Pitch P	4.5
Stabilize Yaw P	4.5		
Rate Roll		Rate Pitch	
P	0.1500	P	0.1500
I	0.1000	I	0.1000
D	0.0040	D	0.0040
I max	500.0	I max	500.0
Rate Yaw			
P	0.2000	I	0.0200
D	0.0000	I max	0.0000

Table 2: GPS Relative Landing Error Measurements

Rel. Error (m)
3.2
4
3
2.2
3.5
8
7
2
3
0.4
3.5

3.6 GPS Accuracy

We knew from the pre-study that GPS and inertial navigation would be insufficient to meet our precision requirements, as both the geodetic model used to issue target coordinates and the GPS system itself have a margin of error which varies depending on location, ionospheric weather and a multitude of geographical factors from the place where the GPS signal is sensed, hence we had planned to integrate the IR-Lock Beacon guided landing system. Nevertheless, in case we could not make it to the stage of the project when we integrated the beacon, we made a study to determine our system's estimated landing accuracy compared to Google Earth's model using only GPS and inertial navigation. To this end we performed a series of test missions, where we would command the drone to fly to a target GPS coordinate, and once it landed we would measure the GPS coordinates of its landing spot using a cellphone's GPS and we would compare its euclidean distance to that of the target given to the drone as $error = \sqrt{(\lambda_d - \lambda_p)^2 + (\phi_d - \phi_p)^2 + (z_d - z_p)^2}$, where λ , ϕ and z denote longitude, latitude and relative altitude respectively. As seen in Figure 6 , we estimate the average accuracy to be around 3 and a half meters, albeit with a relatively high standard deviation.

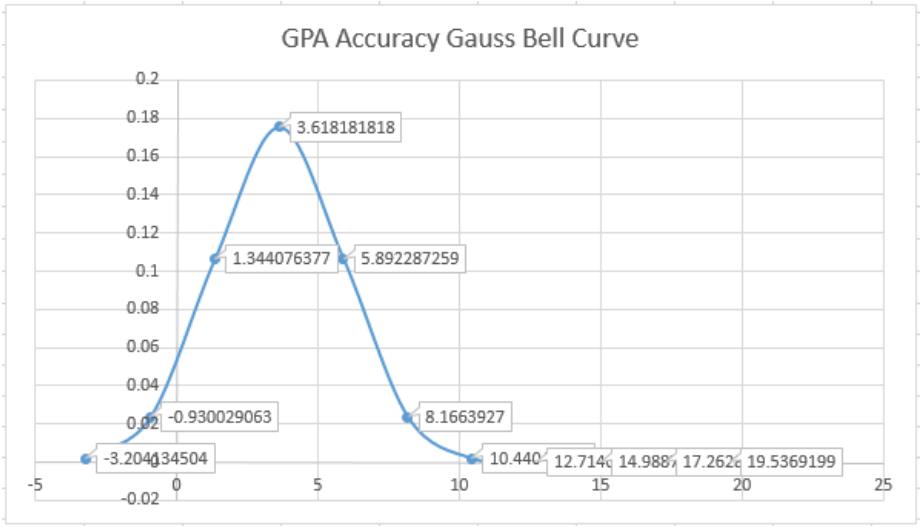

Figure 6: Gauss bell plot of our measured GPS landing error.

3.7 Flight Controller Logs

An objective view of the overall Roll and Pitch performance can be seen in Figure 7, Figure 8, and Figure 9 for Roll Rate output vs. Desired Roll Rate, Pitch Rate output vs. Desired Pitch Rate, Yaw Rate output vs. Desired Yaw Rate respectively. Due to the proper auto tuning these graphs shows precise rates during different stages of the drone flight, such as Landing, Stabilize, Loiter and Alt. hold. The output rates and desired rates of both Roll and Pitch are closer to zero at Loiter and Alt. hold but shows little over-shot at landing and stabilize mode. This overshot could be due to external factor such as air density, wind, etc.

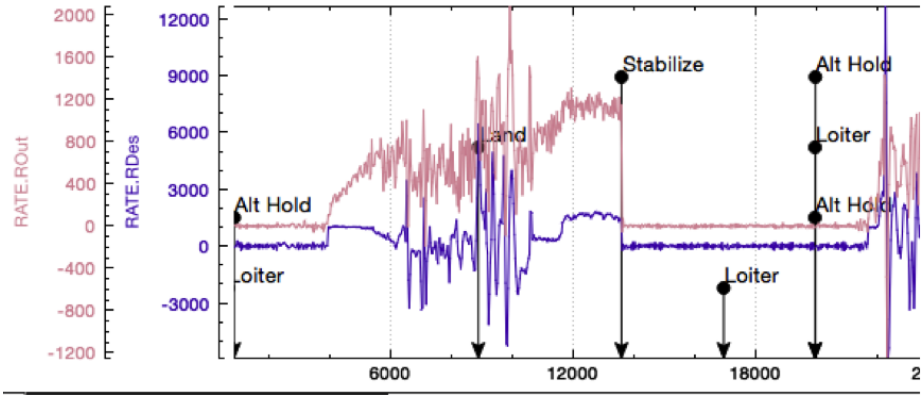

Figure 7: *Roll Rates*

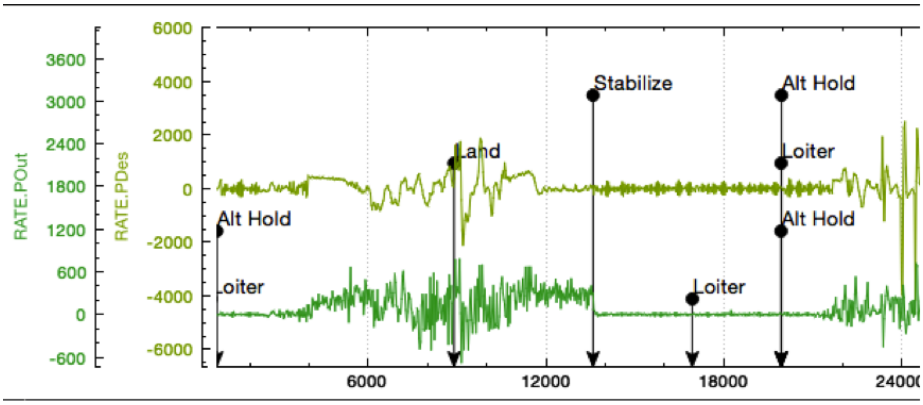

Figure 8: *Pitch Rates*

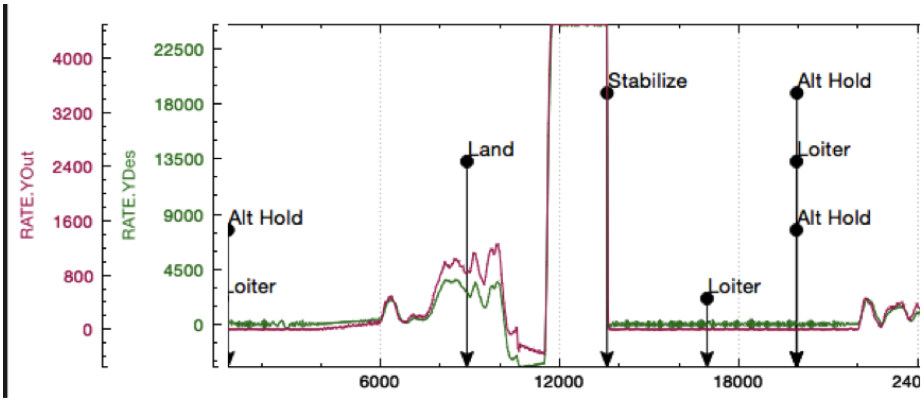

Figure 9: *Yaw Rates*

4 Embedded Systems Architecture

The quadcopter is essentially a fly by wire device, and needs complex computation and calculations to keep it stable and steady. This is achieved with a variety of sensors, actuators and converters for these actuators. The firmware with RTOS runs on top of the underlying hardware and manages all the control and telemetry.

4.1 Hardware Architecture

The hardware entities which make up the drone are:

4.1.1 Flight Controller

The Flight controller is a PCB which integrates all the main sensors and the processor into a single package. It is essentially responsible for all the calculations and actuation signals. In this project it was decided that the Pixhawk PX4 flight controller from 3DR will be used.

4.1.2 MicroController

The computational unit is comprised of a 32-bit STM32F427 Cortex M4 core with a floating point unit, 256 KB RAM and 2 MB Flash. This ARM core has the NUTTX RTOS running in it. This communicates to the external world by with a FTDI serial to USB converter chip.

4.1.3 GyroScope

L3GD20 A 3 axis MEMS Digital Gyroscope from STM is integrated on the board. This gives the roll, pitch and yaw rate to the MicroController. These 16 bit length values are stored in the a FIFO buffer and is sent via the SPI interface to the Microcontroller. It also has a **Data Ready** line connected to the microcontroller for notification when the data is available.

4.1.4 Accelerometer and Magnetometer

LSM303D, a 3-axis 14-bit accelerometer and magnetometer from STM is also present. This gives the acceleration in all the three axes in terms of "G". The integrated magnetometer gives a measurement of the magnetic field around it. These communicates to the microcontroller by SPI So the accelerometer gives an orientation of the drone by measuring the downward acceleration in different positions and the magnetometer can measure the acquire the "yaw position" of the drone.

4.1.5 GyroScope and Accelerometer Combo

The Invensense MPU 6000 3-axis accelerometer and gyroscope is also present to add more integrity and redundancy to the final rate readings. This also communicates to the micro controller via SPI.

4.1.6 Barometer

MEAS MS5611, a high resolution barometer which can detect small changes in pressure is also integrated on the board. This provides 24 bit pressure values and can detect changes as low as 10cm in altitude. This also uses SPI to transfer data.

4.1.7 External Connectors

The pixhawk package comes with cables of DF13 standard, a common connector type used in avionics which prevent unintended removal of cables.

4.1.8 GPS and Compass

The GPS is connected externally with a distance to the Pixhawk to reduce interference. The GPS used is the u-blox NEO-7, which has a high update frequency of 5Hz and communicates via UART interface via the DF13 connectors. The compass communicates to the Pixhawk via the I2C interface.

4.1.9 Electronic Speed Controllers

ESCs are used to interface the microcontroller to the motor. They accept PWM input from the microcontroller and drive brush-less DC motors, whilst maintaining the speed of the motor. The refresh rate of the ESC determines how fast the ESC can be updated by the microcontroller to make changes into effect. There are four ESCs connected for the four different motors.

4.1.10 Radio Control Transmitter and receiver

A general 5 channel RC is used for this project. The receiver generates PPM signals which is then connected to the inbuilt PPM decoder.

4.1.11 IR Lock

The IR Lock is a essentially an IR camera which is pre loaded with firmware to detect objects. This IR camera can send these identifications over I2C interface to the micro controller over 50 times a second.

4.1.12 Ultra Sonic Distance sensor

An ultrasonic distance sensor is used to complement the barometer in altitude measurement. It is connected to the ADC ports of the Pixhawk.

4.1.13 Radio Control Transmitter and receiver

A general 5 channel RC is used for this project. The receiver generates PPM signals which is then connected to the inbuilt PPM decoder.

4.1.14 USB to Serial

Since the Pixhawk has 3.3V serial ports, a USB-Serial converter cable is also included in the design. Though the PixHawk has a USB port, it is recommended to use the TELEM ports of the PixHawk for communication. Hence an USB to Serial port by FTDI is used.

The entire architecture is shown in Figure 10. The Pixhawk PX4 acts as the central node where all the processing and communication takes place.

4.2 Pixhawk Software Architecture

The Pixhawk firmware consists of a base RTOS called NUTTX, which manages the low level hardware calls, schedules the different tasks, provides message passing constructs and houses all the device drivers such as actuators, sensors, CAN etc. On top of the kernel there are various libraries which the source code uses for processing. Most important one is Mavlink library and the math library, which provides translational functions and various numerical functions respectively.

The Control Framework uses these libraries to implement the control law and send the corresponding values to the actuators. They read values from the RC receiver or an application, calculates the desired yaw, pitch and roll angles and sends the corresponding PWM output to the motor ESCs. THe Application

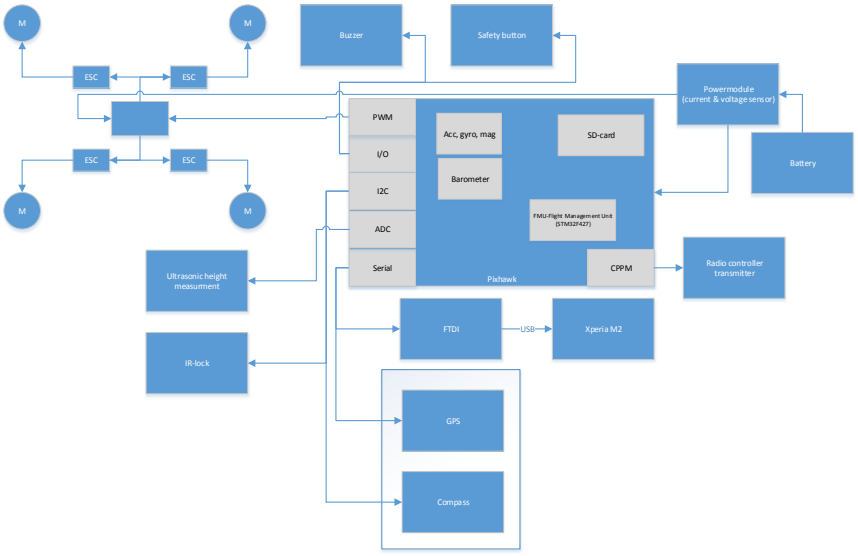

Figure 10: Hardware architecture

layer is the top most layer where all the mission data is stored. One example is a collection of waypoints, which then is fed into the control layer for trajectory control, thus making the drone fly automatically along waypoints.

The Pixhawk firmware uses a message handling system for inter process communication called "publish subscribe", which does not require a locking mechanism. The item which is being subscribed is called **topic**. The process sending the message is called publisher and publishes a topic. The subscriber can read this topic by a callback to the function, call back to the class or a handle.

4.3 Cloud Architecture

Figure 12 show the class diagram of the cloud and Figure 11 shows the connection between the cloud and the android app in the phone.

4.4 Android App Architecture

5 Software Development

The firmware flashed on the Pixhawk we have interacted with from the phone/cloud:
Custom Firmware supporting IRLock for precise landing:

GIT: https://github.com/ThomasSFL/ardupilot.git
Commit: d377f6e6f2a4a4d09eb425bc15539a631a9ef05
Submodules:
Original PixHawk Firmware

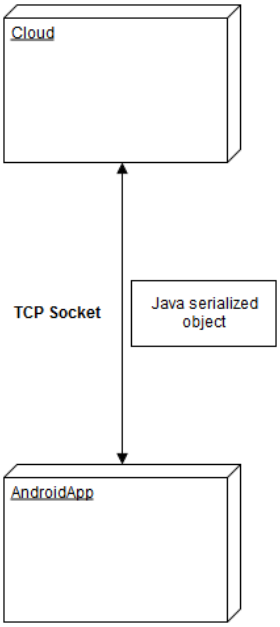

Figure 11: Cloud to phone

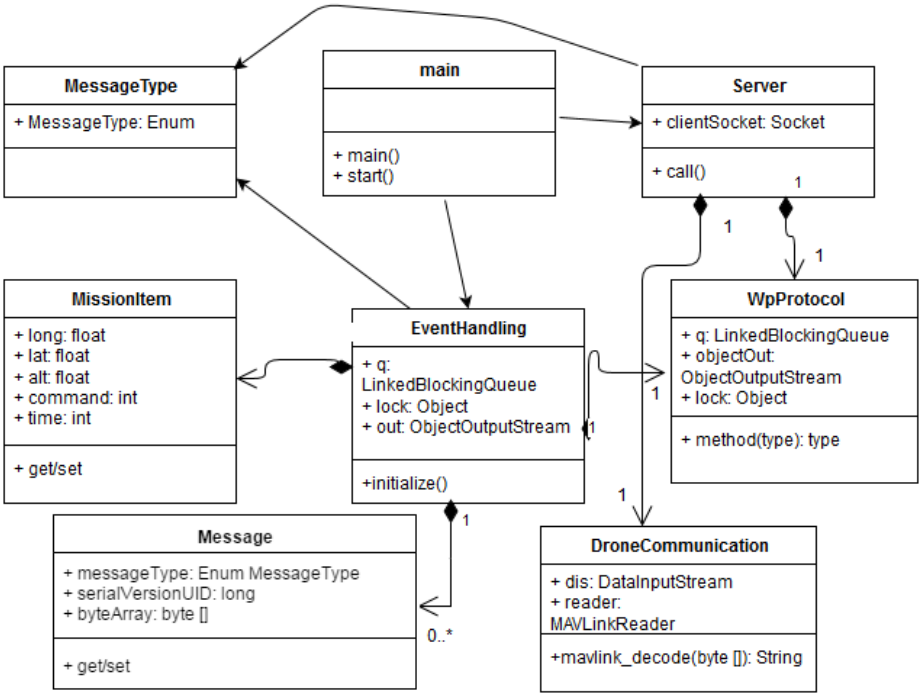

Figure 12: Cloud class diagram

GIT: git://github.com/diydrones/PX4Firmware.git
Commit: 546f20648ad92d6a3acc03cb4b4a7d0cf09dde21

modules/PX4Firmware
NuttX Real Time Operating System
GIT: git://github.com/diydrones/PX4NuttX.git
Commit: d48fa3072b2396c489966c4ab10183ac7cf3dea9
modules/PX4NuttX

5.1 MAVLINK Protocol

Mavlink exists in different versions and the one we are using are 1.0. In the Mavlink protocol, every system, like a drone or a ground control station, is assigned a unique ID called System ID. Within the system parts are assigned unique IDs called Component ID. When a message is sent, it includes the ID's of the sender and if directed to a certain system andor component, the ID's for those as recipient. Messages should only be handled by the systems and component it is meant for, but those can also choose to filter out messages from unauthorized senders.

In the version of pixhawk firmware used, the possibility exists to filter and it is currently configured to only accept messages from System ID 255.

To let systems that share communication channels know the others are present, a Heartbeat message is transmitted with a certain frequency typically 1HZ. If a system in contact with another stops receiving their heartbeat messages, it enters failsafe mode if enabled and the user can choose the actions taken depending on what system it lost contact with. For instance, landing immediately or go back to home position are possible actions of failsafe mode. A heartbeat message contains information about the system transmitting it, like system type such as Ground Control Station, the System ID and Component ID. It can be used by other systems intending to send messages to that specific system.

The pixhawk mounted on the drone has the possibility to use the attached high power sound emitter to sound an alarm when contact is lost with the controlling system, but it has been disabled since it was not convenient to the testing environment.

We use 1 target system and component ids however, the mavlink protocol allows to retrieve those via MAVLINK Parameter request MAV SYS_ID.

5.2 MAVLINK commands

Current firmware handles the following mavlink messages:

MAVLINK_MSG_ID_COMMAND_LONG
This command acts a container for commands from the MAV_CMD enum. It has Command ID as well as parameters defined by the Command ID.

1. MAV_CMD_NAV_TAKEOFF
   This command tells the drone to go to the next mission item that is a takeoff.

2. MAV_CMD_NAV_LAND
   This command tells the drone to go to the next mission item that is a land.

3. MAV_CMD_COMPONENT_ARM_DISARM
   Either arms or disarms the drone.

MAVLINK_MSG_ID_SET_MODE
Allows the user to change the mode. See section 5.3 for available modes.

MAVLINK_MSG_ID_RC_CHANNELS_OVERRIDE
The user can set values for each individual RC-channel. We never managed to get it working though.

MAVLINK_MSG_ID_HEARTBEAT
The heartbeat message previously explained.

MAVLINK_MSG_ID_REQUEST_DATA_STREAM
Tells the Pixhawk to send data from the specified category with specified frequency. This is an old command that is not supposed to be used anymore according to the Mavlink documentation, but it is the only one supported in the current firmware.

MAVLINK_MSG_ID_MISSION_COUNT
MAVLINK_MSG_ID_MISSION_ITEM
MAVLINK_MSG_ID_MISSION_CLEAR_ALL
MAVLINK_MSG_ID_MISSION_REQUEST
MAVLINK_MSG_ID_MISSION_SET_CURRENT
See section 5.6.3 for mission planning.

For parameter specifics see the code.

5.3 Flight modes

The pixhawk supports multiple flight modes as follows:

1. RTL
2. POSHOLD
3. **LAND**
4. **OF_LOITER**
5. **STABILIZE**
6. **AUTO**
7. **GUIDED**
8. DRIFT
9. FLIP

10. AUTOTUNE
11. ALT_HOLD
12. **LOITER**
13. POSITION
14. CIRCLE
15. SPORT
16. ACRO

The ones highlighted in bold is the only that has been tested and are used in the cloud application.
Some of the modes changes the flying characteristics of the drone and others are used for specific tasks.
The pixhawk firmware has its own modes defined that requires the use of the custom mode field when setting mode in the SET_MODE mavlink message.
The main modes used by our software is Guided mode and Stabilize mode. Stabilize mode is the basic mode in which the drone can be armed skipping GPS and other sensors check and that is its only use from our software. To fly missions Guided mode is used. It is the same as Auto mode when flying from the controller, except that no manual action is required. If there is a mission loaded in the flight controller, it can only run in Guided mode and can be activated by setting the current mission to the sequence number of the wanted mission item. A mission sequence always starts from sequence number 0 with its Home location and then the actual mission items start from sequence number 1.

5.4 Communication

The communication between the phone and the cloud is done using a java serialized object, see Figure 13 over TCP sockets. The object consist if an enum that defines the possible message types. They are currently MAVLINK, WIFION and WIFIOFF. If mavlink, it is meant for the Pixhawk and the mavlink message is in the encoded byte array, otherwise it is a command for the phone and the byte array is empty.

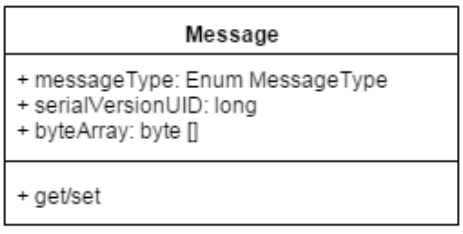

Figure 13: Java serialized object

5.5 Android application

The android application partially acts as a ground control station for the flight controller transmitting heartbeat and acts as a bridge between the flight controller and the cloud.

5.5.1 Debugging UI

The user interface consists of large text area automatically scrolled which acts as logger for the whole application enabling all threads to log their states and transmission and reception messages. Also, it consists of start communication button to start all the communication tasks and WIFI toggle button to enable and disable the hotspot and finally a scroll toggle button to pause the scrolling in order to be able to read the log. A screenshot of the debugger UI is present in Figure 14.

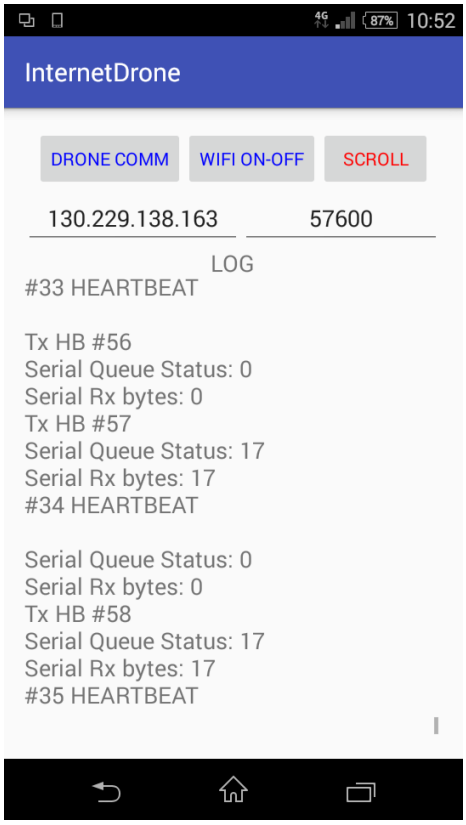

Figure 14: The debugger UI GUI

5.5.2 Tasks

CloudCommunicationTask

- The task is responsible of establishing the connection with the cloud application.

- It starts in connecting state and keep retrying to connect every 3 seconds.
- Once the connection is established, it creates the **CloudTxTask** and **CloudTxTask** and it moves from connecting state to waiting for tasks error notification. 2 ways synchronization is used here for Cloud connectivity resilience. If an error occurred in any of the tx or rx tasks this task get notified and notify the tx and rx tasks to gracefully exit even if they are blocked waiting for new messages and moves to state waiting for tasks shutdown. Once the tasks gracefully exited this task moves back to connecting state.
- priority: normal

CloudTxTask

- The task is responsible of reading from the mavlink encoded message sent via mailbox from the DroneRxTask and wraps it into the java serialized object and transmits it to the cloud application.
- Duration: blocking on the mailbox.
- priority: normal

CloudRxTask

- The task is responsible of reading message from the cloud socket connection discover whether it is a mavlink encoded message or other control messages such as wifi on and off and take the proper action for each. For instance, if it is mavlink encoded message then it transmits it via the FTDI driver to the flight controller.
- Duration: blocking on the socket.
- priority: normal

DroneTxTask

- The task is responsible of sending heart beat messages to the flight controller.
- The task is resilient to usb connection and disconnection.
- Duration: 900 ms
- priority: high

DroneRxTask

- The task is responsible of receiving mavlink encoded stream from FTDI driver pass it through mavlink decoder , print the messages for debugging and finally forward it to the cloud application for processing.
- The task is resilient to usb connection and disconnection.
- Duration: 200 ms
- priority: high

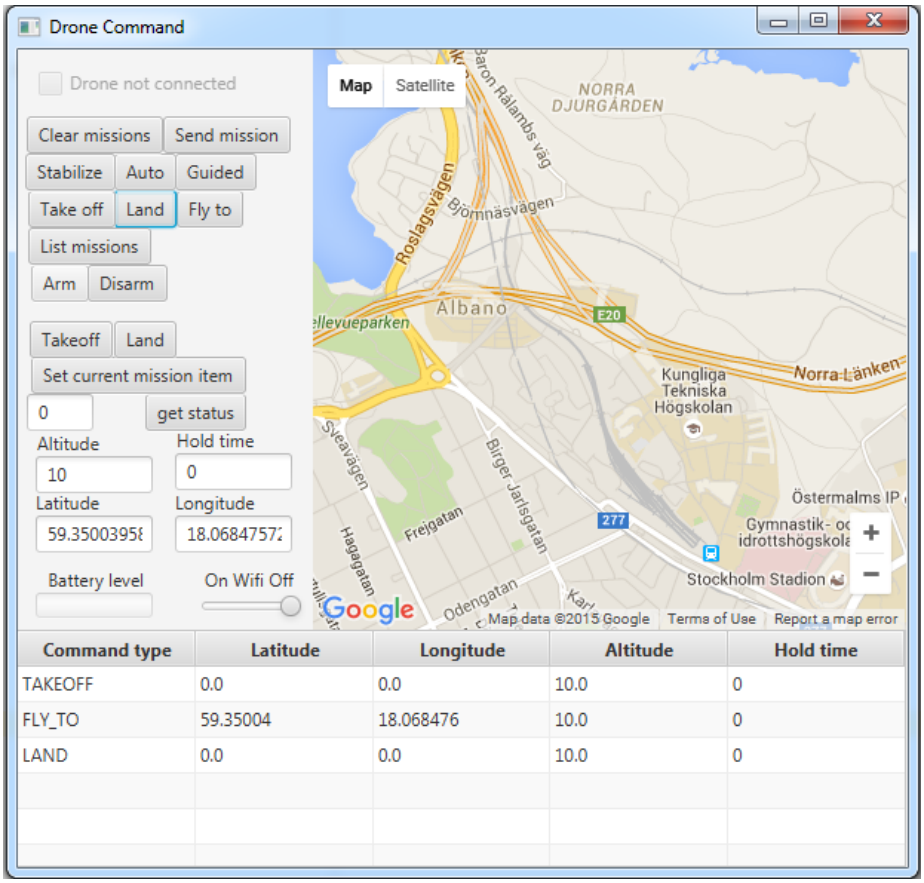

Figure 15: Screenshot of GUI

5.6 Cloud software

The software consists of three tasks continuously running, one additional task while accepting connections and a few other classes. The GUI can be seen in Figure 15.

5.6.1 GUI task

The gui is loaded in the Main class from a fxml-file and EventHandling.java is used to handle the actions from the user. In this class, the server socket task is created when no client is connected. It avoids freezing the GUI while waiting for connections and gives the possibility to accept multiple clients, each spawning tasks for reception. This is not implemented though, but the design choice was made with this in mind.

5.6.2 Reception task

This task receives from the socket and prints the information from the received messages in the console. It should be extended with the functionality to display important message information in the GUI instead like battery level. When a message is received, DroneCommunication.java is used to create a mavlink object from that message's byte array.

5.6.3 Waypoint protocol task

To write a new message, the sequence in Figure 16 is performed. This is according to QGroundControl and what is used in our software. MissionPlanner however sends back a final MISSION_ACK to the Pixhawk.

This task was added to perform the necessary steps for the waypoint protocol. It is the only class that has not been improved since its original creation. It could be integrated in the reception task, but was not done from the beginning because it was easier not to and the need for it was urgent. It is basically a state machine that changes states for each received message in the protocol and output is sent. It also includes some code for reading the current mission list from the Pixhawk, but that code is not working correctly.

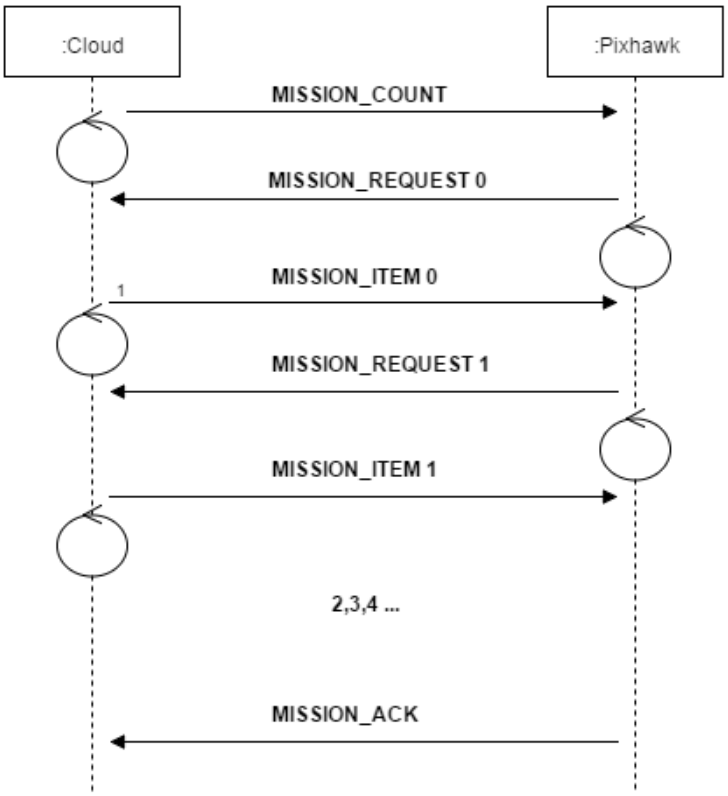

Figure 16: Sequence to write mission

5.7 Captive Portal

5.7.1 Introduction to Captive Portals

It is very common for WiFi networks to have some sort of identification of users. A normal way of doing this is by creating a captive portal. A captive portal is when users connecting to the network are redirected to a login screen, which normally is a website hosted by the WiFi provider. This allows the provider to restrict the use of the network to authorized users and to gather information about which users use the network and when.

5.7.2 The work process

We decided to implement a captive portal through the use of iptables. Iptables is a flexible firewall utility that uses policy chains to allow or block traffic [11]. It allows you to create a list of rules which all incoming traffic will be checked against. Should a rule be applicable it will be applied. If no rule is found the default action will be taken[11].

We began by defining the requirements needed to create the captive portal. From that we arrived at an initial set of requirements which were later expanded upon as our knowledge of iptables increased which allowed us to better understand what was needed to achieve the desired results. Based on these requirements we designed the following rules to be implemented:

1. iptables -N internet -t mangle;
2. iptables -t mangle -A PREROUTING -j internet;
3. iptables -t mangle -A internet -m mac –mac-source a2:ee:d5:f3:9e:14 -j RETURN;
4. iptables -t mangle -A internet -j MARK –set-mark 99;
5. iptables -t nat -A PREROUTING -m mark –mark 99 -p tcp –dport 80 -j DNAT –to 192.168.43.1:8080;
6. iptables -t filter -A FORWARD -m mark –mark 99 -j DROP;
7. iptables -t filter -A INPUT -d 192.168.43.1 -p tcp –dport 80 -j ACCEPT;
8. iptables -t filter -A INPUT -p tcp –dport 80 -j ACCEPT;
9. iptables -t filter -A INPUT -p udp –dport 53 -j ACCEPT;
10. iptables -t filter -A INPUT -m mark –mark 99 -j DROP;
11. iptables -A FORWARD -i p2p0 -o wlan0 -m state –state ESTABLISHED,RELATED -j ACCEPT;
12. iptables -A FORWARD -i wlan0 -o p2p0 -j ACCEPT;
13. iptables -t nat -A POSTROUTING -o p2p0 -j MASQUERADE;

We then went on to create a small app which runs these rules on the Sony Xperia we were working on. This was the step that took the most time as we had several issues during the process.

Finally we attempted to create the captive portal itself through the use of a local website on the Xperia through the use of nanohttpd[12]. This is where we ran out of time and as such we did not have the ability to create a proper login screen. Instead we decided to focus our efforts on proving that the redirect was functioning properly and as such made a website which only displays a small amount of text to confirm that we have been properly redirected.

5.7.3 Issues

During our work we ran into several obstacles that halted our progress. The greatest of these was testing our code. This was a very time-consuming task which led to slow progress. The reason why this took unexpectedly much time was because it was hard to debug errors due to many Linux libraries being used which caused errors with unclear reasons and also the rules themselves could not be tested until the program itself was functional. The error we ran into the most was an exception caused by a "broken pipe" which was a lost connection. This happened while running the initialization process where all rules were created. Solving this took several days. We eventually found a way to avoid this issue by having the rules stored in an external file saved to the SD-card of the phone instead of having them defined in the code.

Another issue which caused great delays was building an understanding of iptables and how to use them. This took far more time than expected. In the end the cumulative delays prevented us from creating the logic logic for the captive portal.

5.7.4 Results

Due to large delays in our progress we were unable to completely finish the captive portal but we have managed to create most of it. We have the ability to redirect incoming traffic to a website located on the Sony Xperia. This website however only contain a few lines of text to confirm that the redirect worked as intended.

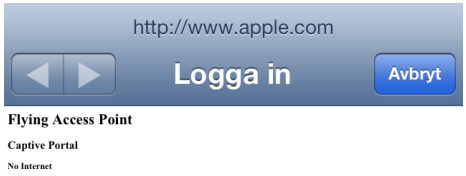

Figure 17: User Screenshot

We have implemented some of the methods needed to implement the login logic but have been unable to properly test them. We have also come to change

one of our design choices over the course of our work. Due to this we are not using all of the rules. We have removed the following rules:

1. iptables -t mangle -A internet -m mac –mac-source a2:ee:d5:f3:9e:14 -j RETURN;

2. iptables -t filter -A INPUT -m mark –mark 99 -j DROP;

The first rule was removed due to us not having been able to finish the captive portal. It is the rule which, based on a list of mac addresses, decides whether or not a connecting user shall be allowed to access the internet directly. The second rule was intended to block traffic that was not directed at any of the ports we were using but we removed it as we found it to be better not to block all other traffic.

Now we will explain the remaining rules:

1. iptables -N internet -t mangle;
   Creates a new chain named "internet" in the manging table (iptable). The purpose of this table is to manipulate packets before the routing decision is made.

2. iptables -t mangle -A PREROUTING -j internet;
   The "-A" modifier signifies that a pre-routing rule is appended to the table mangle and to our chain "internet". The "-t" specifies the table, which is "mangle"

3. iptables -t mangle -A internet -j MARK –set-mark 99;
   Append a rule to our chain to mark all incoming packets with a 99 mark

4. iptables -t nat -A PREROUTING -m mark –mark 99 -p tcp –dport 80 -j DNAT –to 192.168.43.1:8080;
   Append a rule to the PREROUTING chain in the NATing table, that any packet that has been marked with 99 and has the destination tcp port 80, will go through a static nating redirecting it to given ip and port, that ip and port in our case here is the phone ip where our nano webserver is running on port 8080

5. iptables -t filter -A FORWARD -m mark –mark 99 -j DROP;
   Append to the forward chain (which is packets being forwarded) a rule to drop all packets marked with mark 99. This means that all packets we just redirected in the past rule will be stopped by this rule so that they don't do any further.

6. iptables -t filter -A INPUT -d 192.168.43.1 -p tcp –dport 80 -j ACCEPT;
   Append to the input chain (all packets coming in to the phone ip) a rule to accept all packets coming to port 80 (accessing www) which have not been marked by the 99 mark

7. iptables -t filter -A INPUT -p tcp –dport 80 -j ACCEPT;
   Append to the input chain a rule to accept all packets coming to port 80 (accessing wwww) which are not marked.

8. iptables -t filter -A INPUT -p udp –dport 53 -j ACCEPT;
   Does the same as rule 7, only for port 53 (which is DNS).

9. iptables -A FORWARD -i p2p0 -o wlan0 -m state –state ESTABLISHED,RELATED -j ACCEPT;
   Accept forwarding from input interface p2p0 (the android 3g/4g interface) to the output interface WiFi packets with tcp state established. That state means the connection that has been already established (meaning the packets succeeded to reach the internet)

10. iptables -A FORWARD -i wlan0 -o p2p0 -j ACCEPT;
    Accepts forwarding in the other direction, complimenting rule 9.

11. iptables -t nat -A POSTROUTING -o p2p0 -j MASQUERADE;
    This rule does the dynamic nating on all packets going out to the internet (p2p0) in the postrouting chain.

The "-A" marker found in the rules above stands for "Append" and is used to append new rules to the chain. When disabling the captive portal the same rules as above are executed with the exception of exchanging all "-A" for "-D" which instructs the app to delete the rules instead.

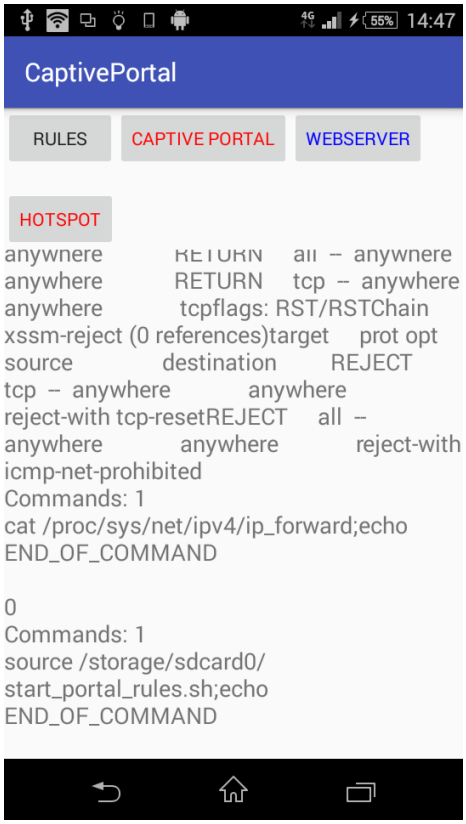

Figure 18: Captive Portal App Screen-shot

- Overview of the buttons seen in the app screenshot in Figure 18

Rules This button in the app displays the rules currently in use on the screen.

Captive Portal Enables/Disables the captive portal.

Webserver Starts/Stops the nanohttpd webserver which the captive portal redirects to. If it is disabled the users will be able to access the internet directly.

Hotspot Enables/Disables the WiFi hotspot.

6 Hardware

As the parts are of-the-shelf parts the important part of the hardware development was to mount the parts on the provided frame.

6.1 Assembly

The frame from jDrone was already assembled from Ericsson with the motors and motor controllers attached. What we have done is to attach the sensors, cables, pixhawk, phone and other hardware to the frame. A document of the assembly can be found on our GitHub [15]. We tried to separate the high power parts to the bottom and the sensors to the top to minimize the interference.

6.2 Fastener

6.2.1 Screws and nuts

The frame manufacturer had used poly carbonate screws and nuts for holding parts together. They prove a lightweight and yet strong fastening in comparison to a metal screw. They was proven to be strong for soft impacts and for hard impacts they will break in order to save other parts from breaking like the motors.

They are a bit brittle to enable this break at hard impact but it also made the screw sensitive to the torque applied by the screwdriver with made it a bit hard to screw in with a hand operated screwdriver.

But as the screws broke after several crashes, new screws was needed. Poly carbonate screws was hard to find and because of previous experience with nylon 6,6 we chose to buy more screws of Nylon 6,6 type to replace the broken screws. The nylon 6,6 is not as brittle as the poly carbonate and this made it easier to screw them in and still provided a good stiffness. The nylon 6,6 proved to be a good replacement for the poly carbonate and no issue have been perceived yet.

6.2.2 Cable ties

For this prototype a quick and easy way of attachment of parts was needed as the time to manufacture parts would have taken to much time. They have been really useful for cables as that what they are made for but also they proved to be useful for attaching the pixhawk and other parts. Because they work similar to ropes, testing if the parts fall of or not was essential. Testing was conducted by rotating the assembled drone to see if the parts still stays on. A bit of shaking close to what could be when flying to see if still is robustly connected.

6.3 Landing

For the first test only the poly carbonate legs was used as landing legs. They proved to be very sensitive to aggressive landing in a bad angle as they break easily when bending. To solve this we went for a Styrofoam base to make the landing more soft for aggressive landings. The base also provided a good structure for mounting the landing sensors and also to keep the battery in the center of the drone. Cutouts was made for the legs that also provided a good support against rotation and a rope was used to hold the Styrofoam to the frame. We got the inspiration for this solution from an earlier project at KTH [14], but made some additional improvements to save more weight and to use it for packaging.

6.4 Cabling

Cables was provided by the jDrone and the pixhawk package and the fastening was done with the cable ties to the frame.

6.5 Frame

A frame from jDrone[16] was provided by Ericsson. A first task was to consider if the frame is good enough or if we need to buy a new frame. We chose to keep the drone based on that it's modular, easy to assemble and easy to repair and to get a new will cost money and time.

6.6 Phone

We first placed the phone on top of the frame, but after some test it seams like it was interfering with the GPS and it was moved down to the Styrofoam base. It also made it easier to navigate the phone when mounted.

6.7 Testing WiFi Signal Strength and Bandwidth of the Phone

Tests were carried out to determine the signal strength and the bandwidth of the WiFi hotspot provided by the drone. When we designed a test for this we decided to take into consideration the possibility that the orientation of the drone would affect the WiFi signal. We decided to repeat our signal strength tests at different orientations, separated by 45 degrees, to ensure we would detect any deviations based on orientation. We decided to measure the signal strength and bandwidth from five different distances where the furthest away distance would be the limit of connecting to the hotspot. The signal strength was measured through the use of the Xirrus WiFi Inspector software and the drone was rotated to test the signal strength at different orientations. The bandwidth was tested through connecting to `www.speedtest.net/sv/` and performing a test.

6.7.1 Results

Signals are recorded in dBm. Angels are counted with the top of the phone as 0 degrees, increasing clockwise around. Speed is tested with an angle of 0 degrees

and are averages of five separate tests.

	Zero Distance	3.3 meters	6.6 meters	10 meters	13.3 meters
Signal Strength	5 bars	3 bars	3 bars	2 bars	1 bar
0 degrees	-45 dBm	-71 dBm	-76 dBm	-79 dBm	-81 dBm
45 degrees	-43 dBm	-70 dBm	-72 dBm	-74 dBm	-81 dBm
90 degrees	-44 dBm	-65 dBm	-72 dBm	-74 dBm	-77 dBm
135 degrees	-47 dBm	-62 dBm	-70 dBm	-70 dBm	-76 dBm
180 degrees	-46 dBm	-65 dBm	-76 dBm	-73 dBm	-75 dBm
225 degrees	-43 dBm	-68 dBm	-71 dBm	-70 dBm	-74 dBm
270 degrees	-45 dBm	-70 dBm	-73 dBm	-77 dBm	-79 dBm
315 degrees	-50 dBm	-71 dBm	-80 dBm	-78 dBm	-80 dBm
Speed Test					
Up	15.37 Mbps	7.39 Mbps	1.83 Mbps	0.55 Mpbs	FtC
Down	25.91 Mbps	2.43 Mbps	0.11 Mbps	0.08 Mbps	FtC
Ping	67 ms	70 ms	78 ms	71ms	89 ms

FtC: Failed To Connect - could not run test (also failed to connect to google.se). Most likely cause: Time Out

6.7.2 dBm

dBm stands for "decibel milliwatts" (in this case) and is a way to measure power. Formula: P(dBm) = 10 · log10(P(W) / 1mW) where P(dBm) = Power expressed in dBm P(W) = the absolute power measured in Watts mW = milliWatts log10 = log to base 10 If a dBm value is negative then the "higher" number is less powerful (-80 dBm is less powerful than for example -60 dBm).

6.7.3 Interpreting the Results

The results indicate the signal strength tends to be stronger from the sides of the phone, rather than from one of the ends. This could be caused by the placement of the internal WiFi unit (in case it's in the middle of the phone for example, which would make it easier for the signals to propagate from the sides which have less materials and electronics to disturb the signal). The effective range of the WiFi is approximately 10 meters. Beyond 10 meters the connection becomes unreliable due to disconnects and time-outs. This is definitely of interest during future work as it would be very beneficial to increase the WiFi range to make the drone able to cover a greater area. This could be done by utilizing an antenna rather than the built in unit of the phone. When considering this a primary concern would be battery life as an antenna easily could affect the battery life notably.

6.8 Testing and flying

How we did the testing to make sure that we didn't take to big risk and to have a system for how we test.

6.8.1 Pre-flight calibration

Calibration is important for the drone to be stable when flying.

- ESC calibration, only needed the first time or when you see that the propellers are not rotating at the same speed when they are suppose to.
- Accelerometer calibration that calibrates the accelerometer by placing the drone on a flat surface in different directions.
- Compass calibration that calibrates the compass by rotating the drone around all axis.
- Radio controller calibration to calibrate signal output from the controller to the stick position.
- Battery calibration for accurate state-of-charge readings.
- Range finder calibration for accurate height measurement reading.

6.8.2 Pre-flight check/start-up

Make sure that the drone is ready to fly by checking for lose parts and that everything is placed in the right position. This list assumes that you stop and analyze the problem if some of the points isn't successful.

- Connect a fully charge battery.
- Wait for the buzzer to make all the sound.
- If the safety button is activated, indicated by flashing led. Hold it in until it stops flashing.
- Try to start in any mode to see if all motors are working.
- Switch to Loiter mode. This mode requires a good GPS called 3D lock. This is indicated by a green led on the pixhawk when the GPS lock is good enough.
- Try to fly a bit to see if the drone seams stable.
- Run you mission.

Problems and solutions:

Drone flies around the target in bigger and bigger circles. Called the "Toilet boil problem" and it's cased by compass-motor interference and can be solved by calibrating the compass while the motors are spinning. This can appear more when the battery has a low charge because the voltage is lower and therefore the current needs to be bigger for the same amount of power.

Doesn't get GPS lock. This have been a problem though the whole project. Different solutions are to power cycle the drone, increase the HDOP value(more then 2 is not recommended), move the sensor away from the high power parts that creates magnetic interference or wait for a better day in case of bad weather.

6.9 Spare parts

As parts breaks new one needs to replace them. As we see it there is three options.

- Repair the broken part. We used quick glue for repairing the plastic parts and it creates a strong bond enough to hold for most of the forces. But the bond is weaker then a complete unbroken part as we experienced when the plastic broke after a lot of use with quick maneuvers.
- Manufacture the part. A drawing for water cutting could be found [17]. I could almost directly be used to cut out parts from plastic sheets or it could be 3D printed and we prepared for doing that but as the time was not enough and spare parts available was enough so we only prepared a CAD of the part so that they where ready for printing.
- Buy the spare part. We bought for example extra batteries that we can't make. And if there isn't any facility for manufacturing then the part might be bought as well. More instructions on how to buy parts where found in [14].

7 System Integration

The software on the Pixhawk is the NuttX RTOS. The way the scheduler works is that it reads from a list of tasks called Task Table. The scheduler then divides up the main thread to fit in these tasks to be executed the required amount of times. The frequencies at which certain tasks run per second are

- 400 Hz
- 200 Hz
- 100 Hz
- 50 Hz
- 20 Hz
- 10 Hz
- 3 Hz
- 1 Hz

Each task is also given a maximum execution time at which it can run. The table below shows the functionality, frequency and maximum permissible execution time. This gives a brief overview of the main threads running in the Pixhawk code.

Frequency	Functionality Description	Execution Time (us)
400	Control loop which updates motors and attitude readings	250
400	Check ground station connection	180
400	Data Logging	100
100	Read Radio Switches	130
100	Updating average throttle	90
100	Accumulate compass values and calculate	100
50	Get throttle reading and update	75
50	Update GPS values	200
50	Get barometer reading and update	90
50	Ground control station data stream	500
10	Update altitude	140

Table 3: Task Table and their properties

8 Prototype Description

As was already mentioned in section 2 the flying access point (pictured in Figure 19) fulfills all phase 1 and some phase 2 requirements put forth by Ericsson.

To summarize it is capable of taking off, flying to a specified location, and landing without any need for user action beyond giving the command and location where the command is to be carried out. It is capable of doing all these things on command from a cloud server connected via the cellular network with commands being accepted both during operation and as queues of commands to be carried out later.

It is capable of providing internet connectivity to nearby clients connecting to the flying access point over WiFi. It can also potentially redirect these clients to a captive portal for the purpose of handling user login or just to display a user agreement.

As described in subsection 3.6 it can reliably land within a 3.5 meter radius of the selected landing point.

9 Individual Contributions

9.1 Abeer Akhtar

In the first half of the project I was involved with a group to find feasibility about Raspberry pi and some peripheral like GPS module and WiFi dongle. I also searched on setting up the platform for Raspberry Pi, libraries we would need and also API. Advanced my knowledge in setting up the environment for drone flying installing firmware in flight controller, calibration procedure and repairing the drone.

In the final stage of the project, I mainly worked with control part of the project and in particular on Control theory behind of the Drone Flight and control parameter of the flight controller i.e. Pixhawk. Developed knowledge of

Figure 19: a photograph of the completed flying access point prototype.

model based approach of drone for controlled behavior, understood the Model of drone flight and its crucial aspects such as Roll, Pitch, Yaw, thrust, moments, and force. Last but not the least together with other group members, worked in agile manner, regularly remained in contact with other members and kept myself updated at all times.

9.2 Fredrik Lindblom

During the prestudy I was part of the group that investigated different choices of drones. When the development started I was part of deciding the communication protocol between the cloud and the phone. I then implemented the protocol in the cloud software and made a GUI to control the drone. I also studied how different existing ground control stations used the Mavlink protocol due to the documentation being insufficiently detailed and how some of the commands was handled by the Pixhawk. The knowledge from the study resulted in a working set of commands from the cloud that could fulfill our needs in controlling the drone.

9.3 Haider Farooq

My contributions were analysis during the initial calibrations, analysis of the auto tuning of the controller, participating in the drone flight tests and repairing of the drones broken parts.

9.4 Johan Westlund

My contribution have been as the test-pilot of the drone under testing. Planing the layout of the assembly and doing the assembling. Make or buy the extra/spare parts needed of the drone like the Styrofoam base and repairing has been a big part to make the drone ready for flights. This also include calibration

pretesting and test planning to make sure that other members could conduct their tests.

9.5 José Luis Bismarck Fuentes Morales

My main contributions to the project were the control characterization and base SIL model checking of the APM and and later of the Pixhawk controllers. In later stages I was part of the team that performed the drone's test flights, my inputs here include running, simulating and validating the drone's auto-tuning and verifying compliance to our requirements from our multiple subsystems, including: GPS, drone fail-safe mechanisms, sensors and actuators calibration. I achieved this via a conjunction of physical test flights, software simulation with jMAVSim and Simulink, and making a critical analysis of the data collected from our flights by the Pixhawk's flight logs. Additionally I did the viability and implications pre-study for using an Android phone as a processing component, as well as early prototyping of the Android app; helped diagnosing and repairing the drone during the most damaging of its crashes and upgraded the shielding Styrofoam base to lessen its aerodynamic drawbacks.

9.6 Mattias Durovic

My contributions have mainly been in managing the project and ensuring that all necessary documentation gets written. My work has primarily consisted of planning the project work, ensuring that all team members are working on the right things, keeping track of the project progress, and resolving any issues and conflicts within the team. I have also spent quite a bit of time doing drone repairs and have acted as a back up pilot for test flights. During the prestudy stage of the project I studied the possibility of using a Raspberry Pi for access point and internet connectivity.

9.7 Mina Tawfik

Briefly, first, I assisted in design decisions evaluating android vs raspberry pi and original android vs rooted one vs cyanogenmod. Second, I got the opportunity to influence the cloud android communication protocol format and to be responsible on the android application capable of communicating with the cloud application via java serialized object over TCP socket and the flight controller via FTDI over USB to serial. A consumer producer design pattern has been used to resolve the internal communication between the transmission and reception tasks. Third, I handled the heartbeat and WiFi requirements. Fourth, I handled the application resilience. Moreover, I assisted in MAVLINK troubleshooting via reading some pieces of the pixhawk source code, via USB message sniffing and I assited in troubleshooting the captive portal. Finally, I contributed when convenient with my documentation pieces to the documentation work such as this report and I assisted the a graphic designer to design our poster.

9.8 Sebastian Nilsson

During the early stages of the project I participated in investigating options for the project. Primarily I was looking into options for WiFi units. I also

participated in writing the various kinds of project documentation. My main tasks during the project were the WiFi testing and the Captive Portal. Of those the captive portal is the one which took the most time and the one I spent much of the later part of the project with. I also designed and constructed a "bus" which we intended to use to test the drone's ability to land on the limited area of a bus roof and travel with the bus to a target destination before taking off and staying around that area to provide WiFi, however we did not get to the point where we could use the IR lock properly and therefore the bus was never used.

9.9 Sharan Yagneswar

My contributions were aiding in the main architecture of the quadcopter, providing assistance with knowledge of hardware, physical construction and calibration of the quadcopter and firmware development. '

10 Conclusions

This report has given a technical overview of the drone the team modified to act as a flying access point for Ericsson.

While the team did not manage to implement all the features it had hoped to include at the outset of the project, and while some of those that were included were not tested to the degree that they should have been, the team is still overall happy with the prototype that was ultimately delivered to the project owner at Ericsson. It includes all the core functionality that the project owner requested as a bare minimum (flight, waypoint navigation, access point functionality). On top of this the team also feels that it has managed to solve the design aspect of the WiFi captive portal and precise landing problems though further work would be needed to properly finalize these features.

That being said the current design should not in any way be seen as a finalized product since the request by Ericsson was for a proof of concept prototype showing that the underlying idea is reasonable. The team has therefore focused less on rigorous testing and design and more on exploring the feasibility and possible features that could be included in such a system.

In this the team feels it has succeeded well. The prototype delivered clearly shows that you can add access point functionality to a drone, that you can modify it to fly missions based on remote commands from a cloud server, and that this can be done using open software and open hardware without the need for custom made components.

A large amount of work would of course still be needed to turn the flying access point from concept prototype into a deployable product. Some of which is summarized below.

In conclusion then, both the team and the project owner feel that they have done a good job given the time and resources made available to them. And

that the documentation delivered to the project owner will be a valuable asset in continuing the development of the flying access point system.

11 Future Work

11.1 RCOVERRIDE

Being able to resolve the rc_channels_override message problem, we can attach a joystick for manual control to the cloud application.

11.2 Live Stream

Extending the android application making use of the cameras to live stream the drone view to the cloud application.

11.3 IR-lock

As this project didn't have the time to test the landing beacon this should be more research up on and tested. This will make the drone drone able to land more accurate and also to land on dynamic objects that are not fixed to a precise position each time. For example a bus will never stop in the exact same position and the beacon can guide the drone.

11.4 Range finder/ultrasonic height measurement

Haven't been tested yet, but the barometer didn't provided good enough measurements to leave the drone to fly long distance. Also in terrain where the height varies a lot the height measurement would be a good addition.

11.5 Enclosed hardware

A problem have been with the outdoor flying when the weather always is an issue. A better enclosed package would make it possible to fly more.

11.6 Obstacle avoidance

To have a true autonomous drone the drone it self needs to be aware of it's surrounding to not fly in to anything.

11.7 Indoor navigation

For demo purpose and it would make some testing not depend on the weather. Obstacle avoidance would be a big part of it.

11.8 Autonomous charging

To minimize human intervention the drone need to be able to take car of it self and automatically charge when the energy is low.

References

[1] (2013, September 2). $100 Home-Made QuadRotor Beams Wi-Fi. http://www.uasvision.com/2013/09/02/100-home-made-quadrotor-beams-wi-fi/. Accessed: 2015-09-27.

[2] (2014, April 14) Pentagon to use drones to create remote wi-fi hotspots. http://www.bbc.com/news/technology-27019389. Accessed: 2015-09-27.

[3] Project loon - how it works. https://www.google.com/loon/how/. Accessed: 2015-09-27.

[4] Connectivity lab. https://info.internet.org/en/story/connectivity-lab/. Accessed: 2015-09-27.

[5] Smith, K. (2014, July 14). Best flight controllers and why. http://myfirstdrone.com/tutorials/buying-guides/best-flight-controllers/. Accessed: 2015-09-27.

[6] Montgomery, C. (2014, June 3) Multi-rotors, first-person view, and the hardware you need. http://www.tomshardware.com/reviews/multi-rotor-quadcopter-fpv,3828-3.html. Accessed: 2015-09-27.

[7] 3DR store pixhawk. https://store.3drobotics.com/products/3dr-pixhawk/. Accessed: 2015-09-27.

[8] APM 2.6 USB has died http://www.dronetrest.com/t/apm-2-6-usb-has-died-solved/236. Accessed: 2015-09-27.

[9] Ir lock. http://irlock.com/products/ir-lock-sensor-precision-landing-kit. Accessed: 2015-09-27.

[10] jMAVSim https://pixhawk.org/dev/hil/jmavsim. Accessed: 2015-09-27.

[11] Brown, K. (2014, June 2). The Beginner's Guide to iptables, the Linux Firewall. http://www.howtogeek.com/177621/the-beginners-guide-to-iptables-the-linux-firewall/. Accessed: 2015-12-20.

[12] Tiny, easily embeddable HTTP server in Java, https://github.com/NanoHttpd/nanohttpd. Accessed: 2015-12-20.

[13] APM Copter Tuning http://copter.ardupilot.com/wiki/tuning/. Accessed: 2015-09-27.

[14] Vanin, M. (2014, January 29). Quad Manual for the Smart Mobility Lab. https://www.kth.se/polopoly_fs/1.451019!/Menu/general/column-content/attachment/quad_manual.pdf. Accessed: 2015-12-20.

[15] Copter assembly documentation.docx. https://github.com/EricssonResearch/flyingaps/blob/master/Documentation/jdrones/Copter%20assembly%20documentation.docx. Accessed: 2015-12-20.

[16] jDrone http://www.jnmechanics.com/Welcome.html. Accessed: 2015-12-20.

[17] Water-jetting Hexacopter frames out of Lexan http://diydrones.com/profiles/blogs/water-jetting-hexacopter-frames-out-of-lexan. Accessed: 2015-12-21.

[18] Drone Control Theory http://www.slideshare.net/corradosantoro/quadcopter-31045379. Accessed: 2015-09-25.