|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  | | --- | --- | |  | **MINISTRY OF EDUCATION AND TRAINING** |  |  | | --- | | **FPT UNIVERSITY** | | **Final report** | | FPT UNIVERSITY FLYING OBJECT  Project code: FUFO  D:\Study\Do an FPT\FUFO\Other\03.Logo\logo_final.png | |  | | **Authors:**  Vu Minh Phong - 00885  Hoang Duc Hung - 60115  Vu Duc Thang - 60124  Ly Khoi Nguyen - 60206  Tran Thi Yen - 01269 | | **Supervisor:**  Phan Duy Hung - Instructor/Supervisor |   **Hanoi, August 9th, 2012** |
| Abstract Unmanned Aerial Vehicle [1], also referred as UAV, and is providing an increasing importance role in many aspect of human life today. As in the past few decades, UAV would be classified as a military area only due to the extremely high cost in both development and operation. However, due to the evolution of technology which leads to the reduction in price of technology equipment and parts, small UAVs are being researched and developed among the technology Universities and companies.  A simple low cost UAV solution will provide an attractive alternative to many civilian and even military applications where human, plane or helicopter would traditionally be used. This application could include: traffic/urban surveillance, aiding search and rescue operations, forest fire detection, national border protection, cross-border smuggling control, etc.  The objective of this project is to build and experiment with a low cost prototype UAV. This involves constructing an UAV, a vehicle control system, as well as evaluating various design features and building materials.  This UAV is a Quadrocopter[2] model, which is propelled using four motors and propellers in opposite directions. The detail of this model will be discussed more intensively later in this report.  The basic theory will mainly consist of modeling and control theory. However, there are some extents in filtering and controller constants finding technique which is crucial and valuable for every Quadrocopter related project. These extents will be discussed both theoretically and practically.  It is important to emphasize that existing solution on the market today are result of projects with far greater budgets than the one available here. Hence, the goal is to experiment with a low-cost UAV and to determine what is possible to archive with small resource. |
|  |

Table of Contents

[Abstract 2](#_Toc332995949)

[1 Introduction 5](#_Toc332995950)

[1.1 Objective 5](#_Toc332995951)

[1.2 History of Quadrocopter 5](#_Toc332995952)

[1.3 Similar products and ideas 5](#_Toc332995953)

[1.4 Scope of the this project 6](#_Toc332995954)

[1.5 Definitions and Acronyms 7](#_Toc332995955)

[2 Hardware study 9](#_Toc332995956)

[2.1 Hardware Overview 9](#_Toc332995957)

[2.2 DsPIC30f4012 Microcontroller 9](#_Toc332995958)

[2.2.1 Overview of dsPIC30f4012 9](#_Toc332995959)

[2.2.2 Feasibility of dsPIC30f4012 in FUFO 10](#_Toc332995960)

[2.3 Bluetooth module HC-06 11](#_Toc332995961)

[2.3.1 Overview of HC-06 11](#_Toc332995962)

[2.3.2 Feasibility of HC-06 in FUFO 12](#_Toc332995963)

[2.4 9DOF: BMP085 + L3G4200D + ADXL345 + HMC5883L 12](#_Toc332995964)

[2.4.1 L3G4200 Gyroscope 13](#_Toc332995965)

[2.4.2 ADXL345 Accelerometer 14](#_Toc332995966)

[2.4.3 BMP085 barometer 16](#_Toc332995967)

[2.5 Mystery FireDragon ESC 30A 16](#_Toc332995968)

[2.6 Motor Himodel A2212-10T 1400kv 17](#_Toc332995969)

[2.6.1 Brushless VS Brushed 17](#_Toc332995970)

[2.6.2 Motor Himodel A2212-10T 1400kv 18](#_Toc332995971)

[2.7 Batteries 19](#_Toc332995972)

[3 Software study 20](#_Toc332995973)

[3.1 Video transmission 20](#_Toc332995974)

[3.2 Motor control with PWM 22](#_Toc332995975)

[3.3 JAVA programming 23](#_Toc332995976)

[4 Hardware design 23](#_Toc332995977)

[4.1 Quadrocopter hardware context and Circuit design 23](#_Toc332995978)

[4.2 Quadrocopter mechanical design 26](#_Toc332995979)

[5 Quadrocopter Dynamic 26](#_Toc332995980)

[5.1 Inertial frame and Body frame 27](#_Toc332995981)

[5.2 Quadrocopter assumptions 28](#_Toc332995982)

[5.3 Quadrocopter dynamic 28](#_Toc332995983)

[6 Modeling the Quadrocopter 32](#_Toc332995984)

[6.1 IMU 33](#_Toc332995985)

[6.2 Euler angular movement measurement 33](#_Toc332995986)

[6.3 Mathematical Filter 35](#_Toc332995987)

[6.4 Altitude measurement 40](#_Toc332995988)

[6.5 PID Controller 41](#_Toc332995989)

[6.6 PID Tuning 43](#_Toc332995990)

[6.7 Software and firmware design 49](#_Toc332995991)

[6.7.1 Software system architecture 49](#_Toc332995992)

[6.7.2 Software on Phone and Computer 51](#_Toc332995993)

[6.7.3 Firmware 53](#_Toc332995994)

[7 Experimental results 59](#_Toc332995995)

[7.1 Experimental setup 59](#_Toc332995996)

[7.2 Experiments results and analysis 62](#_Toc332995997)

[8 Conclusion and suggestion for future development 65](#_Toc332995998)

[9 List of References 66](#_Toc332995999)

[Appendix A: Variants of FSM in FUFO system 67](#_Toc332996000)

[Appendix B: Software Detail Design diagrams 71](#_Toc332996001)

[Appendix C: Firmware and Hardware recorded problems 84](#_Toc332996002)

[Appendix D: Development process 85](#_Toc332996003)

[Appendix E: Test case and Test report 86](#_Toc332996004)

# Introduction

## Objective

The project objective is making a low cost prototyped Quadrocopter system with a restricted resource and evaluates its flight capability both indoor and outdoor. To some extent, the project will also include some aspects of Filtering technique and vehicle control method to determine what is the best achievable solution for this type of UAV without increasing the budget for development.

## History of Quadrocopter

Research into the initial development of Quadrocopters began in the early twentieth century. One of the first engineers to attempt to design a Quadrocopter was Etienne Oemichen. Oemichen began his research in 1920 with the completion of the Oemichen No.1. This design consisted of four rotors and a 25 Horsepower motor; however, during tests flights the Oemichen No.1 was unable to obtain flight. Two years later Oemichen completed his second design; the Oemichen No.2. His second design consisted of four rotors and eight propellers along with a 125 Horsepower motor. Five of the propellers were used to achieve stable flight while two were used for propulsion and the final propeller being used to steer the aircraft. In April of 1914, the Oemichen No.2 achieved an FAI distance record for helicopters of 360m, which the Oemichen No.2 broke with a distance of 525m.

While Oemichen had begun working on his early designs in France, Dr. George de Bothezat and Ivan Jerome began their own research in January 1921 for the United States Army Air Corps. They completed their design in mid-1922, and the first test flight took place in October of 1922 in Dayton, Ohio. Bohezat's and Jerome's design weighed around 1700 kg at the time of take off and consisted of four six-bladed rotors along with a 220-HP motor. After many tests, the Quadrocopter was only able to achieve a maximum flight time of 1 minute 42 seconds and maximum height of 1.8 meters.

Following the research of Oemichen, Bothezat and Jerome, other researchers have attempted to create their own successful vertical flying machines. One such was being the Convertawings Model “A” Quadrocopter. The Convertawings Model “A” Quadrocopter was designed and built in the mid 1950‟s with civil and military purposes in mind. This particular Quadrocopter consisted of four rotors, two motors as well as wings. Due to lack of interest, however, the Convertawings Model “A” Quadrocopter was never mass produced.

Currently Bell Helicopter Textron and Boeing Integrated Defense Systems are doing joint researched on the development of the Bell Boeing Quad Tilt Rotor. The initial design consists of four 50-foot rotors powered by V-22 engines. The main role of the Bell Boeing Quad Tilt Rotor will be that of a cargo helicopter with the ability to deliver pallets of supplies or also deploy paratroopers. The first wind tunnel tests were completed in 2006 and the first prototype is expected to be built in 2012.

## Similar products and ideas

While most of the multirotor systems nowadays are just hobby toy which can be controlled manually by a RF controller, there are some commercialized multirotor systems that are capable of stable flying, video processing, GPS tracking and even obstacle determining and avoiding. Their qualities are superb and in some aspect, they are equal to the military UAV. However, their prices are too high to be able to be mass produced and used in real-life situations, thus remain as another hobby toy for rich people.

Such products are:

|  |  |
| --- | --- |
|  | **Airbot X600-BKPP (€ 34,500)**  - Radio control unit (2.4 GHz)  - Live video stream  - GPS waypoint navigation |
| http://www.eurolinksystems.com/images/cyberGray_small.png  PNG PNG | **CyberQuad Maxi ($ 36,000)**  - Radio control unit  - GPS waypoint navigation  - Live video stream |
|  | **Scout Air Reconnaissance System (Unknown)**  - Wi-Fi secured network (up to 3km)  - Small  - Live video stream via secure network  - Quiet flight  - GPS waypoint navigation.  - Military purpose |

## Scope of the this project

The scope of this project is prototype a Quadrocopter system, includes both hardware and software. The final product must satisfy the following specifications:

|  |  |
| --- | --- |
| Specification | Description |
| Autonomous balancing | Automatically maintains its position in 3D space. |
| Autonomous altitude holding | Automatically maintains its altitude level while hovering. |
| Live video stream | Sends the captured video stream back to PC. |
| Using Bluetooth and Wi-Fi as connection methods. | Please refer to Figure 1. |
| Control by PC/ Android Phone | Pilot sends command to the Quadrocopter by a PC or an Android Phone.  On PC software, the altitude level and altitude indicator are visible to pilot. |

Table 1: Scope of FUFO project



Figure 1: Connection method of FUFO system

## Definitions and Acronyms

|  |  |  |
| --- | --- | --- |
| **Acronym** | **Definition** | **Note** |
| ESC | Electronic Speed Controller[3] |  |
| UART | Universal Asynchronous Receiver/Transmitter |  |
| SPI | Serial Peripheral Interface |  |
| PID | Proportional-Integral-Derivative[4] |  |
| INS | Inertial Frame |  |
| FUFO | FPT University Flying Object |  |
| AOC | Application on Computer |  |
| AOP | Application on Phone |  |
| PC | Personal Computer |  |
| IMU | Inertial Measurement Unit |  |
| CMOS | Complementary Metal-Oxide-Semiconductor |  |
| 10DOF | Ten Degree Of Freedom |  |
| FSM | Finite State Machine |  |
| UART | Universal Asynchronous Receiver/Transceiver |  |
| PWM | Pulse Width Modulator |  |
| SPI | Serial Peripheral Interface |  |
| I2C | Inter-Integrated Circuit |  |
| ESC | Electronic Speed Controller |  |
| TCP/IP | Transmission Control Protocol / Internet Protocol |  |
| UDP / TP | User Datagram Protocol / Internet Protocol |  |
| BLDC | Brushless DC |  |

Table 2: Definition and acronyms

# Hardware study

## Hardware Overview

After several months of studying the possibilities of hardware components for this Capstone project, the team has decided to use the below components to for the hardware of the Quadrocopter:

* 1 Microcontroller: dsPIC30f4012.
* 1 Bluetooth module: HC-06
* 1 Sensor module: 9DOF: BMP085 + L3G4200D + ADXL345 + HMC5883L
* 4 ESCs: Mystery FireDragon ESC 30A
* 4 Motors: Himodel A2212-10T 1400kv

Each of these components will be clearly discussed in the next sections.

## DsPIC30f4012 Microcontroller

### Overview of dsPIC30f4012

DSPIC30F4012 is a 28-pin micro processor published by MICROCHIP Technology Inc.

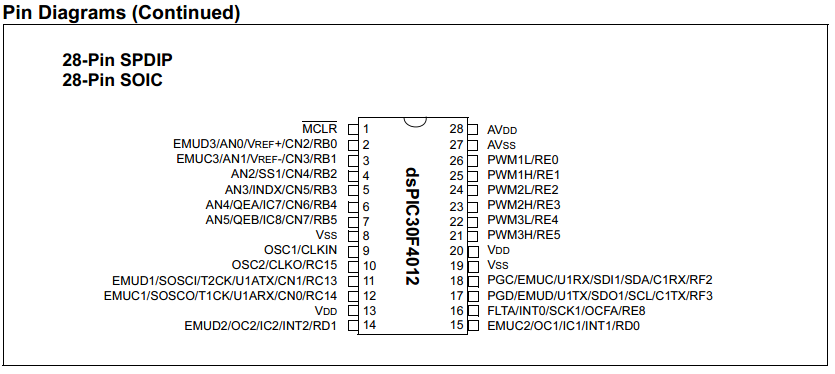


Figure 2: dsPIC30F4012 pin diagram

Features of DSPIC30F4012:

* 84 base instruction
* 24-bit wise instruction, 16-bit wise data path.
* 48kbytes on-chip Flash program space (16k instruction words)
* 2 Kbytes on-chip data RAM
* DC upto 40MHz external clock input
* 16x16 bit working register array.
* 16-bit compare /PWM output function.
* Five 16-bit timers.
* 6 PWM output channels.
* 3 duty cycle generators.
* 10-bit A/D with 6 input channels
* About half a size of PIC16f887.
* 1 UART channel
* 1 SPI channel + 1 I2C channel

Operation voltage: 5V or 3.3V

### Feasibility of dsPIC30f4012 in FUFO

|  |  |  |
| --- | --- | --- |
| **FUFO's circuit needs** | **dsPIC30d4012 provide** | **Status** |
| Have 1 I2C connection for 1 module 9DOF sensor | x1 SPI/I2C connection | OK |
| Have 1 UART connection for Bluetooth module at 9600 baud rate. | * Clock upto 40Mhz (4Mhz is good enough for 9600 baud rate) * x1 UART connection | OK |
| Have 4 PWM outputs for 4 ESCs | * x6 PWM output pin (can work independently) * x5 16 bits timers * 3 duty cycle generators | OK |
| Work at 3.3V | Can either work at 3.3V or 5V | OK |
| As small, light and cheap as possible | * The price of the Microcontroller is 120,000VND on average, which is acceptable. * The size is about a half of PIC16f887 (small and light) | OK |
| Require as little effort to learn as possible | * Similar to other Microchip product in development. We do not have to learn too many new things. * Can use MpLab IDE and for development. | OK |

Table 3: Feasibility of dsPIC30F4012

## Bluetooth module HC-06

### Overview of HC-06

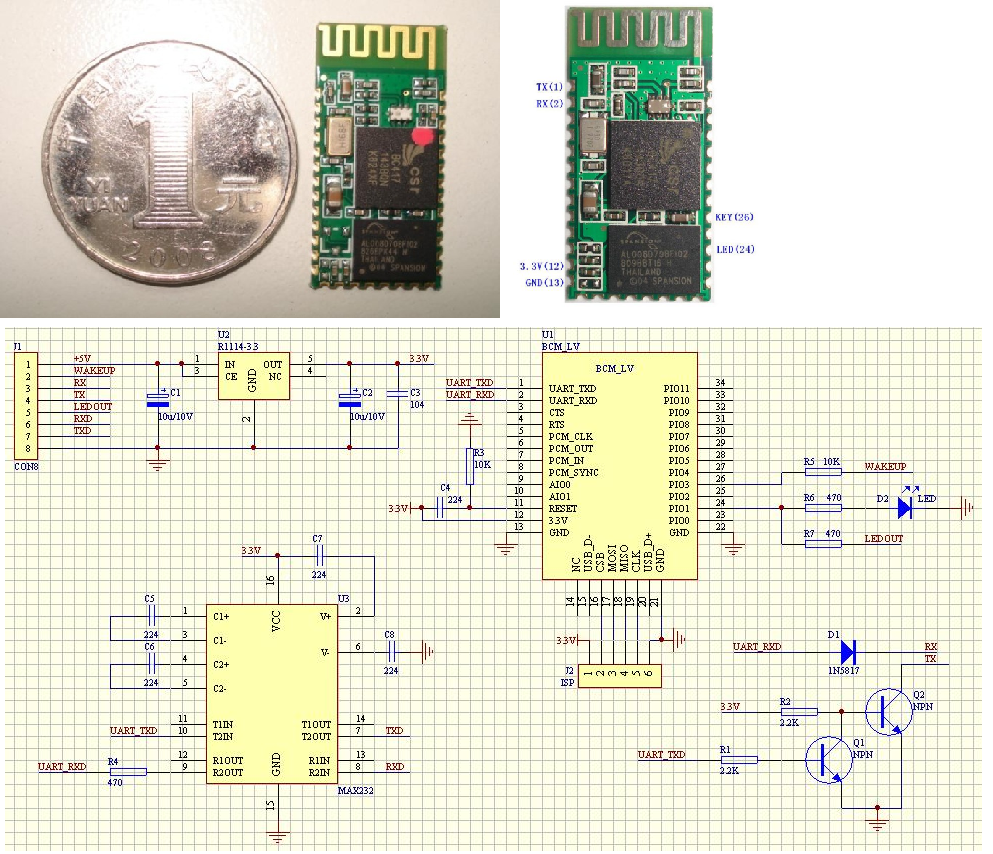


Figure 3: HC-06

Features:

* Use the CSR Bluetooth chip, compatible with the Bluetooth V2.0 protocol
* Slave
* Operating Voltage 3.3V
* Adjustable baud rate : 1200，2400，4800，9600,19200，38400，57600，115200
* Size: 28mm x 15 mm x 2.35mm
* Operating Current 40 mA
* Sleep Current < 1mA
* Program with AT command language
* UART interface
* Range 30ft ~ 9,1m
* Made in China
* Price: 350,000 VND

### Feasibility of HC-06 in FUFO

|  |  |  |
| --- | --- | --- |
| **FUFO's circuit needs** | **dsPIC30d4012 provide** | **Status** |
| Use a Bluetooth module to establish UART connection between Android phone and Microprocessor | Use the CSR Bluetooth chip, compatible with the Bluetooth V2.0 protocol | OK |
| 10m range is enough | Range ~9.1m (30 feet) | OK |
| Baud rate = 9600 | Adjustable baud rate : 1200，2400，4800，9600,19200，38400，57600，115200 | OK |
| Small size | Size: 28mm x 15 mm x 2.35mm | OK |
| UART interface | * Program with AT command language * UART interface | OK |
| Available | The only Bluetooth module can be found in Vietnam market at this time | OK |

Table 4: Feasibility of HC-06

## 9DOF: BMP085 + L3G4200D + ADXL345 + HMC5883L

****

Figure 4: 9DOF: BMP085 + L3G4200D + ADXL345 + HMC5883L

9DOF module (BMP085 + L3G4200D + ADXL345 + HMC5883L) is a Chinese-made sensor module which consists the following sensors:

* + L3G4200: gyroscope
  + ADXL345: accelerometer
  + BMP085: pressure sensor
  + HMC5883L: digital compass

Price of this module is 1,100,000VND

This is the only sensor module our team can find in Vietnam. Each of these sensors will be discussed in the following part.

### L3G4200 Gyroscope

#### Features



Figure 5: L3G4200D Gyroscope

* Three selectable full scales (250/500/2000 dps)
* I2C/SPI digital output interface
* 16 bit-rate value data output
* 8-bit temperature data output
* Two digital output lines (interrupt and data ready)
* Integrated low- and high-pass filters with user-selectable bandwidth
* Ultra-stable over temperature and time
* Wide supply voltage: 2.4 V to 3.6 V
* Low voltage-compatible IOs (1.8 V)
* Embedded power-down and sleep mode
* Embedded temperature sensor
* Embedded FIFO
* High shock survivability
* Extended operating temperature range (-40 °C to +85 °C)
* ECOPACK® RoHS and “Green” compliant

#### Feasibility

Why it is suitable for FUFO project?

* + Ultra-stable three-axis digital output gyroscope
  + I2C/SPI digital output interface
  + 16 bit-rate value data output
  + 8-bit temperature data output
  + Wide supply voltage: 2.4 V to 3.6 V
  + Three selectable full scales (250/500/2000 dps) are capable of measuring rates with a user-selectable bandwidth.
  + The only Accelerometer that we can find in Vietnam.

### ADXL345 Accelerometer

#### Features



Figure 6: ADXL345 accelerometer sensor

* Ultralow power: as low as 23 μA in measurement mode and 0.1 μA in standby mode at VS = 2.5 V (typical)
* Power consumption scales automatically with bandwidth
* User-selectable resolution:
* Fixed 10-bit resolution
* Full resolution, where resolution increases with g range, up to 13-bit resolution at ±16g (maintaining 4 mg/LSB scale factor in all g ranges)
* Patent pending, embedded memory management system with FIFO technology minimizes host processor load
* Single tap/double tap detection
* Activity/inactivity monitoring
* Free-fall detection
* Supply voltage range: 2.0 V to 3.6 V
* I/O voltage range: 1.7 V to VS
* SPI (3- and 4-wire) and I2C digital interfaces
* Flexible interrupt modes map to either interrupt pin
* Measurement ranges selectable via serial command
* Bandwidth selectable via serial command
* Wide temperature range (−40°C to +85°C)
* 10,000 g shock survival
* Pb free/RoHS compliant
* Small and thin: 3 mm × 5 mm × 1 mm LGA package
* 3-Axis, selectable sensitivity: ±2 g/±4 g/±8 g/±16 g Digital Accelerometer

#### Feasibility

Why it is suitable for FUFO project?

* 3-Axis Digital Accelerometer
* Flexibly selectable sensitivity: ±2 g/±4 g/±8 g/±16g
* High shock survival: 10000g
* Supply voltage range: 2.0 V to 3.6 V
* I/O voltage range: 1.7 V to 2.5 V
* SPI (3- and 4-wire) and I2C digital interfaces
* The only Gyroscope sensor that we can find in Vietnam.

### BMP085 barometer

#### Features



Figure 7: BMP085 barometer sensor

* 300 ... 1100 hPa (+9000m ... -500m above sea level)
* Power consumption scales automatically with bandwidth
* Free-fall detection
* Supply voltage range: 1.8 V to 3.6 V
* I/O voltage range: 1.7 V to VS
* Low noise: 0.4 in low-power mode

#### Feasibility

Why it is suitable for FUFO project?

* Low noise.
* Enough for altitude requirement
* SPI (3- and 4-wire) and I2C digital interfaces
* Comes with the module.

## Mystery FireDragon ESC 30A

ESC is Electronic Speed Controller, a simple circuit board that handles the work of controlling speed of Brushless DC motor. After connect all the wire to correct position, the motor's speed can be controlled by the corresponding PWM signal input into ESC.

Here are some features of Mystery FireDragon ESC 30A, which is available on the market:

* Weight: 22g (including wire)
* Max Current: 35A (for ten seconds)
* BEC: 2A at 5V
* Input: 2s - 3s Lithium-ion Polymer.
* Price: 220,000VND

## Motor Himodel A2212-10T 1400kv

### Brushless VS Brushed

Motor is the device that generates power to rotate the propeller. There are two types of motor: brushed motor and brushless motor. The two figures below will show some concepts

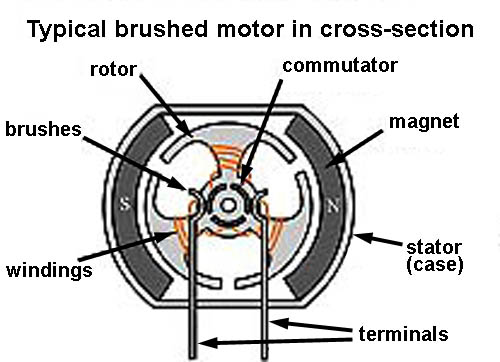
****

Figure 8: Brushed Motor

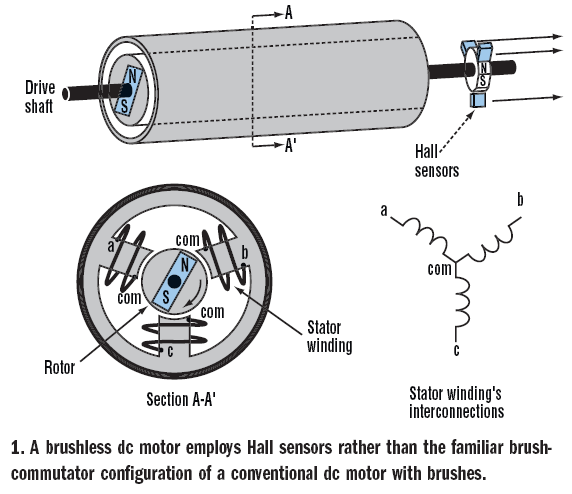


Figure 9: Brushless Motor

**Brushed VS Brushless:**

* BLDC motors are often more efficient at converting electricity into mechanical power than brushed motors.
* The maximum power that can be applied to a BLDC motor is exceptionally high, limited almost exclusively by heat, which can weaken the magnets.
* BLDC motor has more torque per watt (increased efficiency), increased reliability, and reduced noise, longer lifetime (no brush and commutation erosion).

### Motor Himodel A2212-10T 1400kv

Features:

* No. Of cells:2-3 Li-Poly
* Brushless motor.
* Max. Efficiency current: 6-12A (>75%)
* No load current / 10 V:0,7 A
* Dimensions:27.5x30 mm
* Shaft diameter: 3.17 mm
* Weight:47 g
* Recommend prop : 8x4,9x5
* Thrust : 800g +



Figure 10: Motor Himodel A2212-10T 1400kv

Explanation:

* 1400kv or 1400RPM/V (RPM is revolutions per minute). Example:
  + Use power 1V for this: 1400 x 1 = 1400 revolutions/minute.
  + Use power 10V: 1400 x 10 = 14.000 revolutions/minute.
* Max. Efficiency current: As the motor performance, the higher efficiency of energy, the lower of energy loss.
* No load current / 10 V: 0, 7 A mean: if motor doesn’t work, power loss is 10V: 0,7A.
* Recommends Propeller : 8x4, 9x5:



Figure 11: Propeller 8x4 and 9x5

## Batteries

In determining what battery would best suite our needs for this project, the team had to take many factors into account. These included, but were not limited to the total flight time and the total current draw from our four motors. The four motors which are used in this project are the HIMODEL A2212 (as seen in subsequent section) motors. Each motor weighs approximately 47 grams and has a max current draw of twelve amps. Multiplying that by four, we get a total current draw of 48 amps from our four motors. To have a substantial amount of flight time while keeping the total mass of the Quadrocopter to a minimum, the team decided on the Turnigy 3 cell 11.1 volt 3000 mAh battery from Turnigy as shown below. This Quadrocopter will have 2 batteries connect in parallel, which means a battery system of 11.1v 3 cell 6000 mAh To determine the total flight time you must divide the battery's power rating by the current draw of the four motors and multiplying by sixty.

Flight\_time = \*60 minutes

= \*60 = 7.5 minutes

One more important parameter that was taken into account is what is known as the current rating “C”. The battery we have chosen has a continuous current rating of 20 C. This current rating allows us to determine the maximum discharge rate of the battery. To determine this you multiply the current rating by the total power rating of the battery.

Discharge rate (mAh) = current rating \* battery power rating (mAh)

The maximum discharge rate of the battery is 12000 mAh.

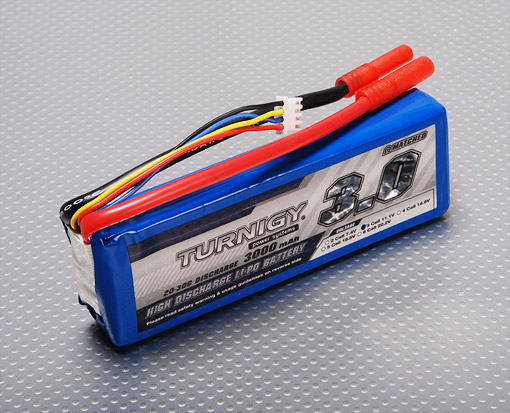


Figure 12: Battery Turnigy 3000mAh 11.1v 3 cells

# Software study

## Video transmission

In this project, a real-time video transmission from Quadrocopter to pilot computer is vital for pilot to control the craft in a long distance. At first, FUFO team intended to integrate a Wi-Fi module and a CMOS Camera module to the Quadrocopter to send picture. However, that may lead to many risks upon the complexity of the main circuit and the short time period system development of this project. Finally, FUFO team decided to use an Android 2.3 Phone instead to solve that issue because an Android 2.3 phone is cheap, and already have integrated camera module and Wi-Fi and 3G module. The Android phone can also communicate with the Quadrocopter via its Bluetooth module. This should not be a problem since the Android SDK has already supported it. That is how the FUFO system looked like the **figure 1**.

The real challenge in this section of FUFO system is the real-time video transmission. The limited bandwidth characteristic of Vietnam Internet infrastructure is the cause of that It is quite impossible to send a 30 FPS high definition video from the Android Phone to the pilot's PC under the Internet Bandwidth. Fortunately, after studying the "*UAV Imagery Frame Rate and Resolution Requirements Study* "**[21]** created by Advance Technology department of Vitro Corporation, the team has figured out that the FPS of an UAV's camera video that is being streamed to pilot can be at 4 FPS in minimum, and JPEG compression of the pictures is not a problem because the pilot can still be able to complete his/her tasks.

With that information, FUFO team has made a prototype of sending picture sequence from an Android 2.2 HTC HD2 phone's 320x240 camera to the PC at approximately 10FPS under JPEG compression. The result is quite positive:

|  |  |
| --- | --- |
| **Description** | **Values** |
| Picture resolution | 320x240 |
| Compression method | JPEG - 50% quality |
| Means of time of taking a picture and compressing it | 124 ms (+5 ms or -5ms) |
| File size before compression | 112.5 KB |
| Means of file size after compression | 8KB (+2KB or -2KB) |
| Means of delay time when send a picture with Viettel 3G | 1s |
| Means of delay time when send a picture with 10/100mbs Wi-Fi | 40ms |
| UDP Packet size | 10KB |
| Loss rate of packet transmitted by UDP via 10/100mbs Wi-Fi | 0/1000 packet = 0% |
| Loss rate of packet transmitted by UDP via Viettel 3G | 12/1071 packets ~ 1% |
| Mean of time needed to draw a picture to screen by JAVA SWING | 5ms |
| Total delay time using Viettel 3G | ~ 1s |
| Total time for a picture to go from phone's camera to PC's application screen using 10/100mbs Wi-Fi | 169ms |
| Possible FPS | 8 FPS |
| Bandwidth needed | ~78KB/s |

Table 5: Prototype's result of real-time video transmission

The results shown by the prototype have clearly proven that this is a good solution for the real-time video transmission section of FUFO project.

## Motor control with PWM

The way to control AC motors is based on principle control a servomotors. The basic signal has a width 1 ms of a logic one with the period 20 ms. After turning on the Quadrocopter must have the PWM pulse a width 1 ms, which means a motor is stopped (Fig. 4 up). With a pulse width greater than 1 ms an ESC won’t let wind off a motor for the safety reasons. For starting up, motors need the pulse width 1 ms. Continuously increasing the pulse width from 1.2 ms up, motors begin to spin. The upper limit of the pulse width it’s 2 ms (Figure 13), when a motor has a maximal performance.

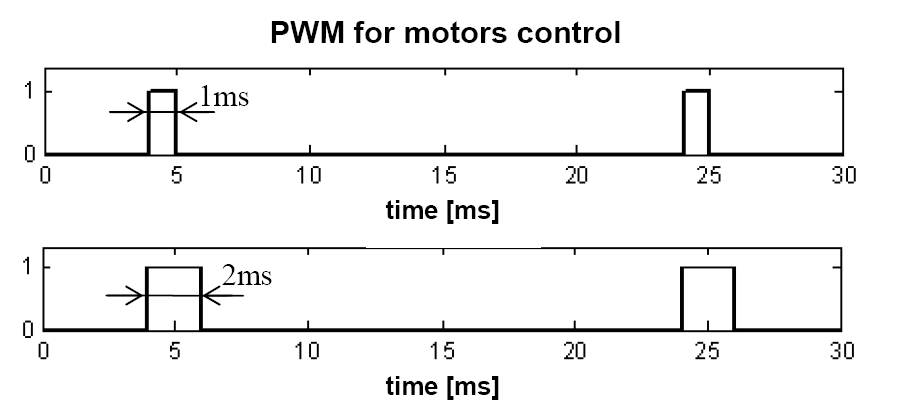


Figure 13: PWM for motors control

## JAVA programming

The software on PC will be programmed by JAVA language. The reason behind this choice is that this language is widely supported by many OS. Therefore it will likely be compatible with any development environment. Moreover, Android programming also uses JAVA as its language. This means that the developer will not have to spend time on training two programming languages for Software on PC and Application on Android Phone.

# Hardware design

## Quadrocopter hardware context and Circuit design



Figure 14: : Hardware context

Figure 14 shows the hardware context of the Quadrocopter. From this context the following circuit schematic design and PCB design has been derived:

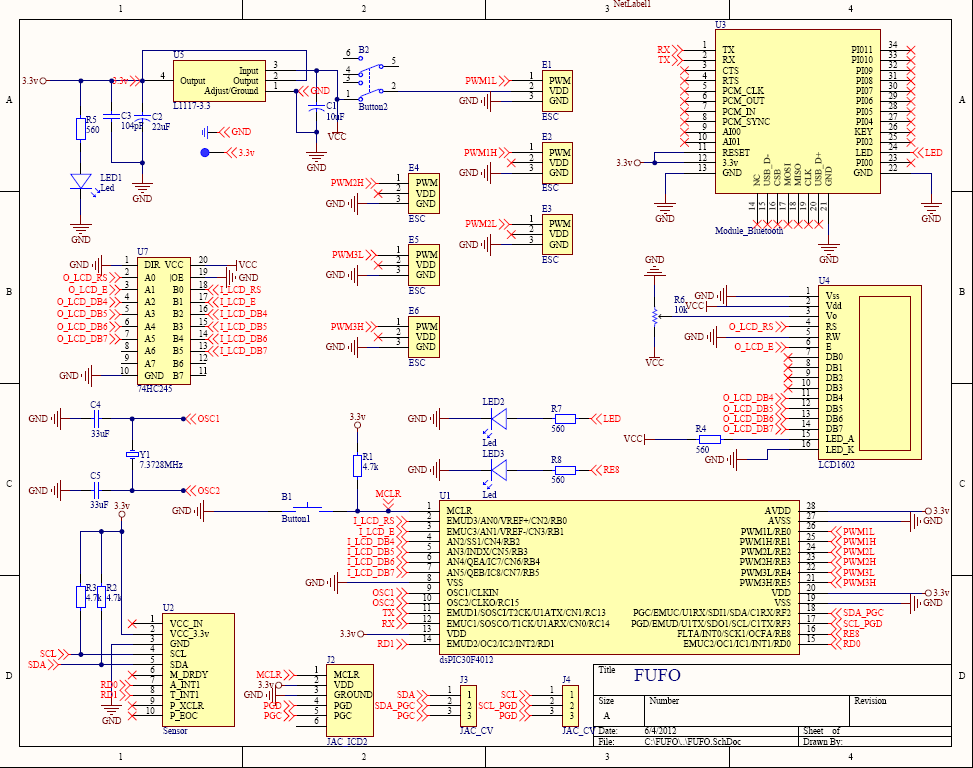


Figure 15: Hardware schematic design

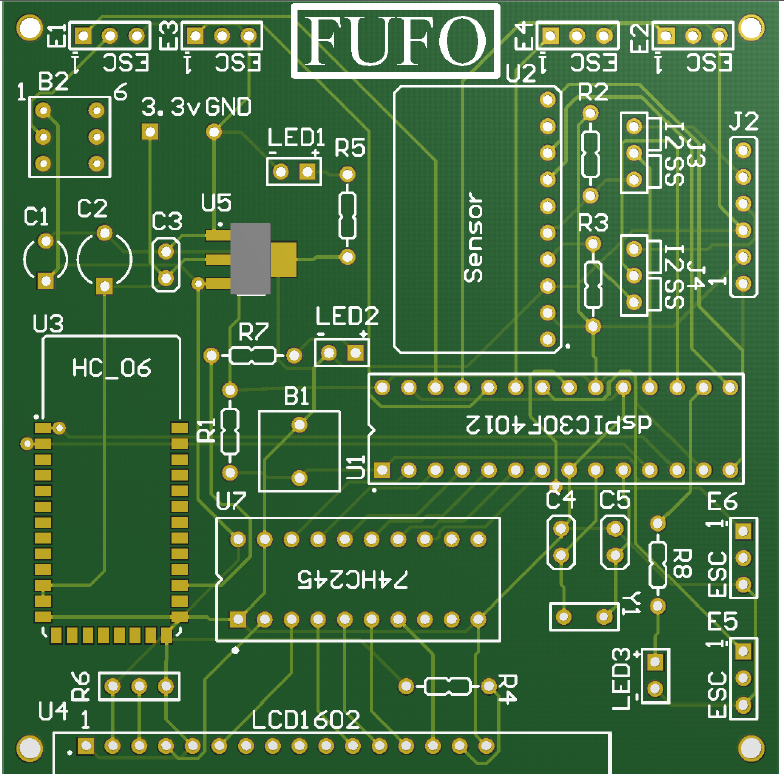


Figure 16: PCB design

This PCB design is evaluated by our supervisors before printing to make sure that it will work. Any fatal error cause PCB design can cause a substantial damage to this project because it will take a lot of time to investigate a hardware problem.

## Quadrocopter mechanical design

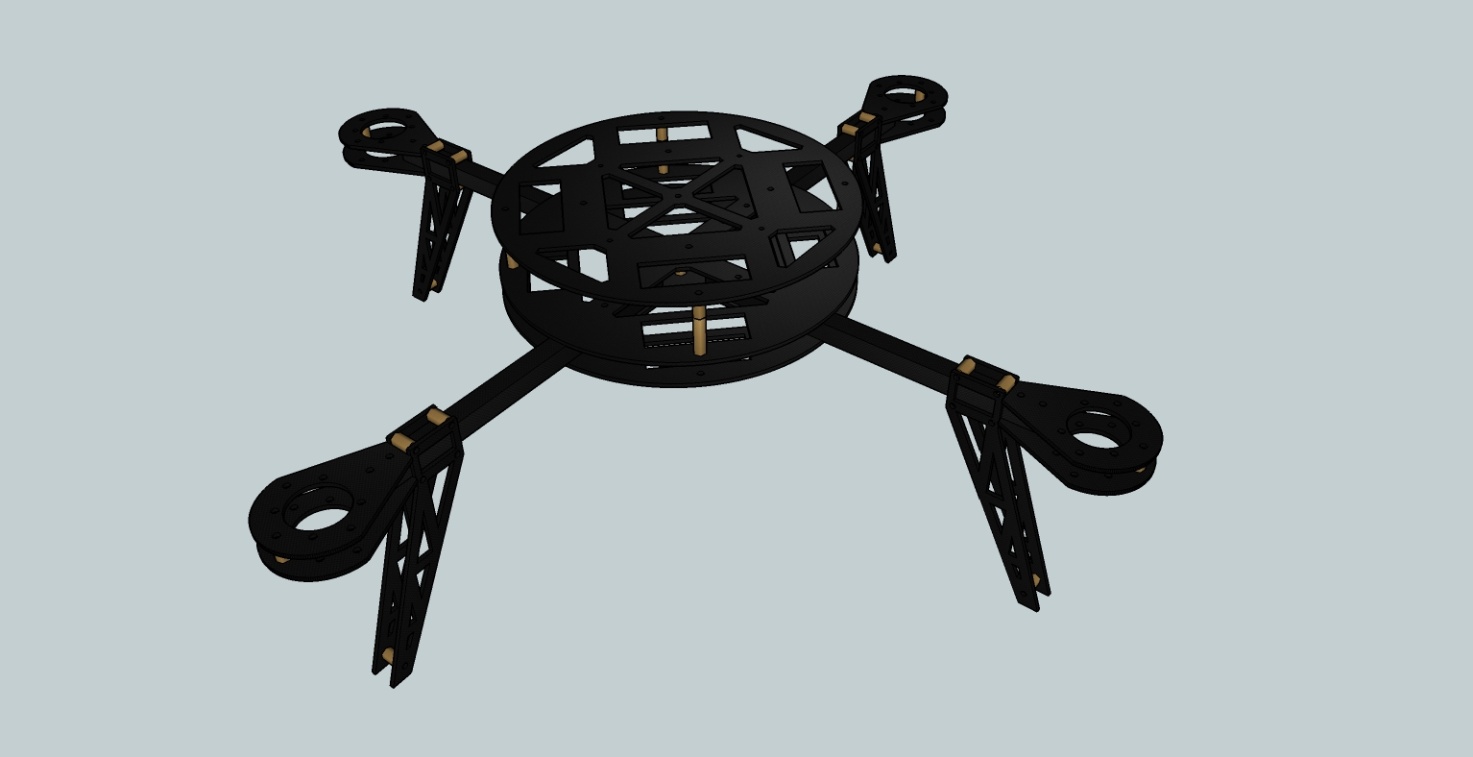


Figure 17: Quadrocopter mechanical design

A 3D mechanical design has been sketched as shown in **Figure 17** (detail mechanical design with exact dimension is included in the CD). The main materials are being used in this design are 2mm Glass Fiber plates and 10mm Carbon Fiber tubes. The latter can be also replaced with Glass Fiber tubes in case of scarcity of Carbon Fiber tubes in Vietnam. The exact dimensions of each component in this design can be found in the Mechanical Design document in AutoCAD format. This design will be manufactured by a CNC machine.

A realized version of this design may not have to be exactly the same in every aspect of the sketched design because of the poor capability in mechanic of software student. Due to the dynamic nature of the Quadrocopter, the hovering capability will not be affected if the moment arms are remained the same, which will be proven by equation 21 in Quadrocopter Dynamic part of this report.

# Quadrocopter Dynamic

In order to understand how to control the Quadrocopter, it is important to understand how it behaves first. This report assumes that the readers do not have much knowledge in aerospace area, therefore, explanation on some aspect of this area is provided as clear as possible with as less math as possible. It is possible for the reader to be confused, so it is recommended to bypass this part if it is not necessary.

For readers who are interested in this subject, please take "*Accurate modeling and robust hovering control for a Quad-rotor VTOL aircraft"[*5] by Jinhyun Kim, Min-Sung Kang and Sandeok Park as a reference.

## Inertial frame and Body frame

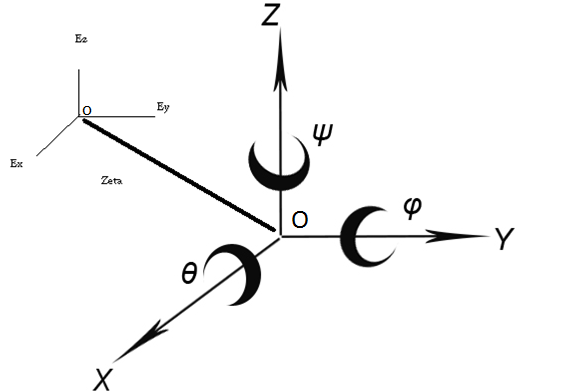


Figure 18: Inertial frame OExEyEz and Body frame OXYZ

Inertial Frame of Reference [6], sometime called Reference Frame or Navigation Frame, denoted as INS, is a 3-axis frame for referencing purpose of a spectacular object in space.

Body Frame is a 3-axis frame that represents the object in space. It is always located with regard to an Inertial Frame.

The body of a Quadrocopter is assumed as a rigid body, the Origin of its Body Frame is located at the mass center.

The inertial frame of a Quadrocopter is earth. How is this possible? The answer is simple: Because the earth is giant compares to the size of a Quadrocopter, therefore it can be assumed as flat. Oz vector of Inertial frame points toward the center of gravity.

In a complex Quadrocopter navigation system, there must be at least two INSs. The first one has the same Origin with the Body frame and the second one has an Origin located on earth surface. This separation is necessary because a Quadrocopter has two general behaviors:

- Maintains its Euler angular positions.

- Maintains its position in space.

To maintain its Euler angular positions, a Body frame need an INS that has the very same Origin with it. That is how Euler Angles are measured. This INS is denoted as INS1. However, the body frame will eventually need to know its position in a space, or to be more detailed, its Origin position in space. In order to archive this, an INS with a fixed Origin on the earth surface is defined. This INS is denoted as INS2.

As the two INS are existed together, a body frame now will have to vectors of position in space:

𝜂1 = [x,y,z]; (1)

𝜂2 = [𝜃,𝜑,𝜓]; (2)

(1) Is the position of Body Frame's Origin with regard to INS2. x, y ,z are also called the linear translational unit vectors.

(2) Is the Euler Angular [7] position of Body Frame with regard to INS1. 𝜃,𝜑,𝜓 is also called the rotational unit vectors.

NOTE: all the Euler Angles must follow a general direction rule which could be either Right-Hand rule or Left-Hand rule [8].

## Quadrocopter assumptions

The following assumptions have to be made before further discussing on Quadrocopter dynamic:

- The Earth is flat, stationary and therefore an approximate Inertial Frame.

- The atmosphere is at rest relative to the ground (zero wind).

- The motors respond rate is fast enough to neglect.

- The flapping angles of the rotors are negligible.

- Force is symmetric in flight and act at center of mass.

- Change in relative air velocity is negligible.

## Quadrocopter dynamic

With regard to (1) and (2), the followings state vectors of the Quadrocopter can be defined:

𝜂 = [𝜂1, 𝜂2]; (3)

= [ 1,  2]; (4)

1 = [ x, y, z]; (5)

1 = [ωx,ωy,ωz]; (6)

Where (3) are described relative to the Inertial Reference Frames while (5) and (6) represent the velocity vector of linear translational movement and rotational movement of Quadrocopter Body Frame, respectively.

The inertial and body-fixed velocity relation can be represented with a Quadrocopter Jacobian matrix[9]:

=  <=> 𝜂' = J(𝜂)(7)

Where

J1(𝜂2) = (8)

J2(𝜂2) = (9)

In above equations, c and s and t denote sin, cos and tan, respectively.

For the moving base system which is not fixed in an inertial frame, it is not convenient to derive the dynamic formulation using the Lagrangian in terms of the velocities expressed in a body-fixed frame. In this Quadrocopter system, the strap-down sensor approach is used, so it is more natural to write up the dynamics using body-fixed velocities. To supplement this, the Quasi-Lagrange approach is used[10]. The quasi-Lagrange method can give the equations of motion in terms of the body-fixed velocities.

The Quasi-Lagrange in general form is defined as:

L = T - V (10)

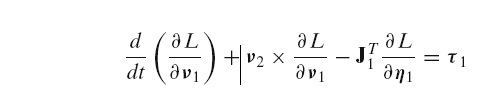
T = vTMv = mv1Tv1 + mv2T I v1 (11)

V = - mgz (12)

Where m and I are mass and system inertia, respectively. Normally, a Quadrocopter body is designed axisymmetric[11], so the inertia is defined as the following equation, and especially, Ixx=Iyy.

I = (12)

With the extended Hamiltonian principle[12], equation 10 gives the following differential equations:

(13)

(14)

Then,

**Mv' + Cv + g = τ**

Finally, the 6 independent equations of motions are obtained as following:

m[vx' - vyωz + vzωy - g sinϴ] = 0 (15)

m[vy' - vzωx + vxωz - g sinϴ sinϕ] = 0 (16)

m[vz' - vxωy + vyωx - g sinϴ cosϕ] = u1 (17)

Ixxὠx + (Izz - Iyy­) ωyωz = u­2 (18)

Iyyὠy + (Ixx - Izz­) ωzωx = u­3 (19)

Izzὠz = u­4 (20)

Where

τ = = (21)

is the distance between motor and the center of mass, or it is usually called moment arm. Equation (21) can be rewritten as the form of matrix like below:

= = T (22)

F1, F2, F3, F4 are the force on each related motor. τ is the torque vector of the Quadrocopter.

The above equations now can finally be explained in physical aspect as below:

- The total force created by four motors is the force that lifts the Quadrocopter, as shown in equation (17)

- Force created by two opposite motors is capable of rotating in left/right or front/back directions the Quadrocopter, as shown in equation (18) and (19).

- Force created by four motors but two opposite one will rotate in a particular direction and two others will rotate in the reverse direction is capable of rotating the body around itself.

Figure 19 below shows the motor alignment scheme of a Quadrocopter and its motion which is induced by the combination of 4 motors.

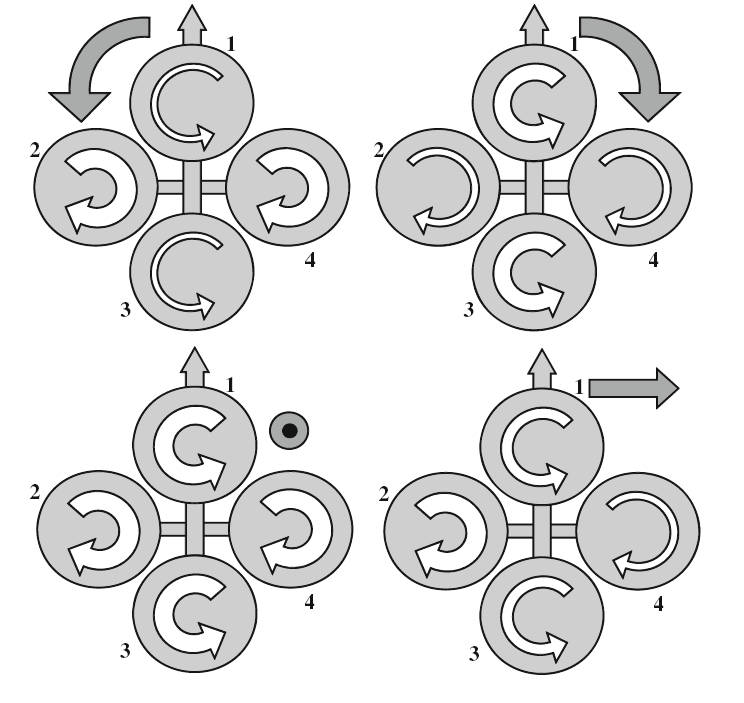


Figure 19: Motors alignment scheme

Until now, only the dynamic equations of motion related to the body fixed INS1 are derived,. However, for the control purpose, it is more convenient to use the dynamic equations derived in earth-fixed coordinate frame like below:

Mη(η) η'' + Cη(ν, η) η '+ gη(η) = τ η(η) (23)

To express the dynamic equations in earth-fixed coordinate frame like Equation 23, the following relationship is needed:

η' = J(η)ν ⇐⇒ ν = J−1(η) η'

η' = J(η) ν' + J'(η)ν ⇐⇒ ν' = J−1(η)[η'' − J'(η)ν] (24)

Then, the system matrices are defined as below:

Mη(η) = J−T(η)MJ−1(η)

Cη(ν, η) = M'η(η)

gη(η) = J−T(η)g(η)

τ η(η) = J−T(η)τ

(25)

Finally, the equations of motion in earth-fixed coordinate frame can be derived.

mx'' = (sψsφ + cψcφsθ)u1 (26)

my'' = (−cψsφ + sθsψcφ)u1 (27)

m(z'' + g) = cθcφu1 (28)

M η2 η''2 + M' η2 η'2 =  (29)

where

M η2 = (30)

As shown in previous subsections, the linear equations of motion of a Quadrocopter are simple in earth-fixed reference frame, while the angular equations are advantageous to express in Body-fixed INS. According to the above analysis, finally, the following equations are derived.

mx'' = (sψsφ + cψcφsθ)u1,

my'' = (−cψsφ + sθsψcφ)u1,

m( z'' + g) = cθcφu1,

Ixxω'x + (Izz − Iyy)ωyωz = u2,

Iyyω'y+ (Ixx − Izz)ωzωx = u3,

Izzω'z = u4.

(31)

As shown in the equation 31, for the linear motions, all the states are subordinated to the control parameter *u*1, hence only one state is controllable and the others are subjected to the controlled linear motion and angular motions. In this paper, for the hovering control, we only consider and control the *z*−directional linear motions. Especially, the hovering control with *φ* ≈ 0 and *θ* ≈ 0 can make the dynamics much simpler form like equation 32, and it is easy to design the controller.

m(z''+ g) = u1,

Ixxφ'' = u2 − (Izz − Iyy)θ'ψ' ,

Iyyθ''= u3 − (Ixx − Izz)ψ'φ',

Izzψ'' = u4

(32)

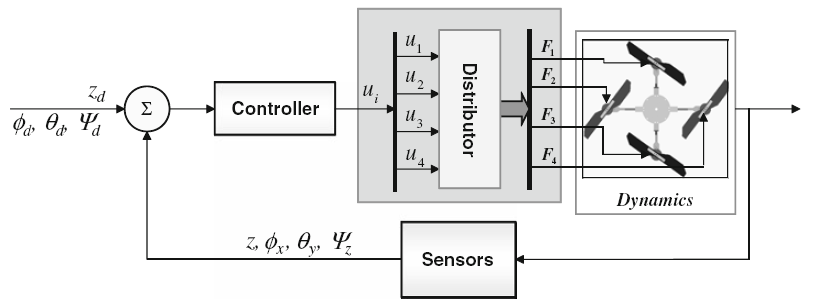


Figure 20: Basic structure of the controller for the UAV

To satisfy the conditions where equation 32 is valid, *φ* and *θ* have to stay near the neutral positions, which are zeros.

# Modeling the Quadrocopter

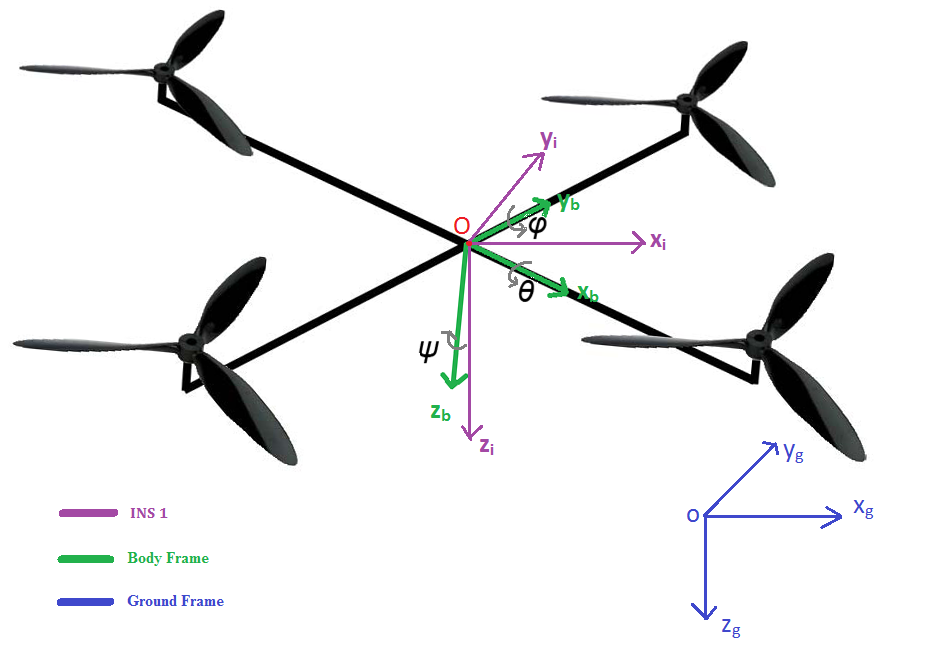


Figure 21: Strap-down Inertial Measurement system.

This part will mainly discuss about the aforementioned dynamic model in embedded system term.

As mentioned in (2), there are three variables that need to be considered in INS1 system: 𝜓 (yaw), 𝜃 (roll), 𝜑(pitch). Hence, measuring the change in those angle and control the speed of each motor are important.

There are many approaches to archive this, and the most commonly used method is using a strap-down Inertial Measurement system, where body frame are axisymmetric with the Quadrocopter body as in figure 21.

## IMU

A good and cheap choice for small UAV system nowadays is the Inertial Measurement Unit (IMU)[13]. It is a packet comes with an accelerometer, a gyroscope, a magnetometer and a barometer. For the prototyped version of Quadrocopter which is developed to evaluate rotational movement and altitude movement aspect, all of the sensors above are used which an exception of the magnetometer because of time issue.

Furthermore, the data that can be mined from an IMU are:

- Yaw angle.

- Pitch angle.

- Roll angle.

- Altitude in meter.

The question on how to get these data will be answered in the next sections.

## Euler angular movement measurement

Using a direction cosine transformation matrix with a little modification in "*Tilt sensing using linear accelerometer*"[14] by Laura Salhuana, the body acceleration movements which is recorded by an Accelerometer sensor can be converted to Euler angular movements by the following equations:

𝜑 = (180/π) \* Arctan[ -Rx / SQRT(Ry2 + Rz2) ] (33)

𝜃 = (180/π) \* Arctan[ Ry/SQRT(Rz2 + µRx2) ] (34)

where Rx, Ry and Rz are acceleration movement on x, y, z vector, respectively. µ is a fraction of Rx2 that needed to be bigger than zero to prevent the denominator in equation (34) ever being zero.

Unfortunately, the Yaw angle cannot be able to calculate using accelerometer data.

Another approach to calculate Euler angles is using Gyroscope's data, which is just angular velocity. This approach is fairly simple: just integrate the instant angular changes over time.

𝜓 **=** (35)

𝜃 **=**  (36)

𝜑 **=**  (37)

The tested results of equations (33) to (37) are shown in the figures below:

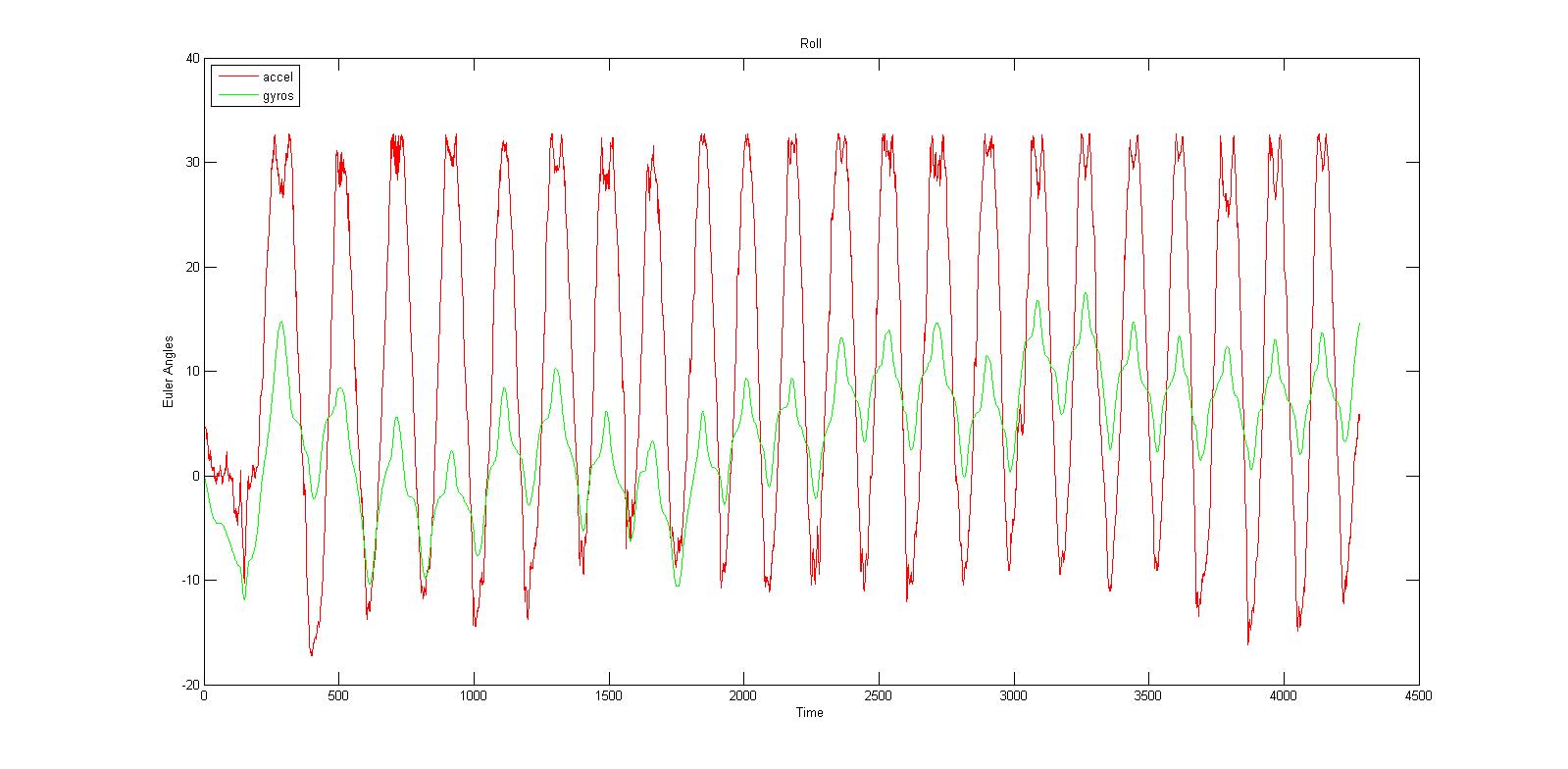


Figure 22: Roll Angle

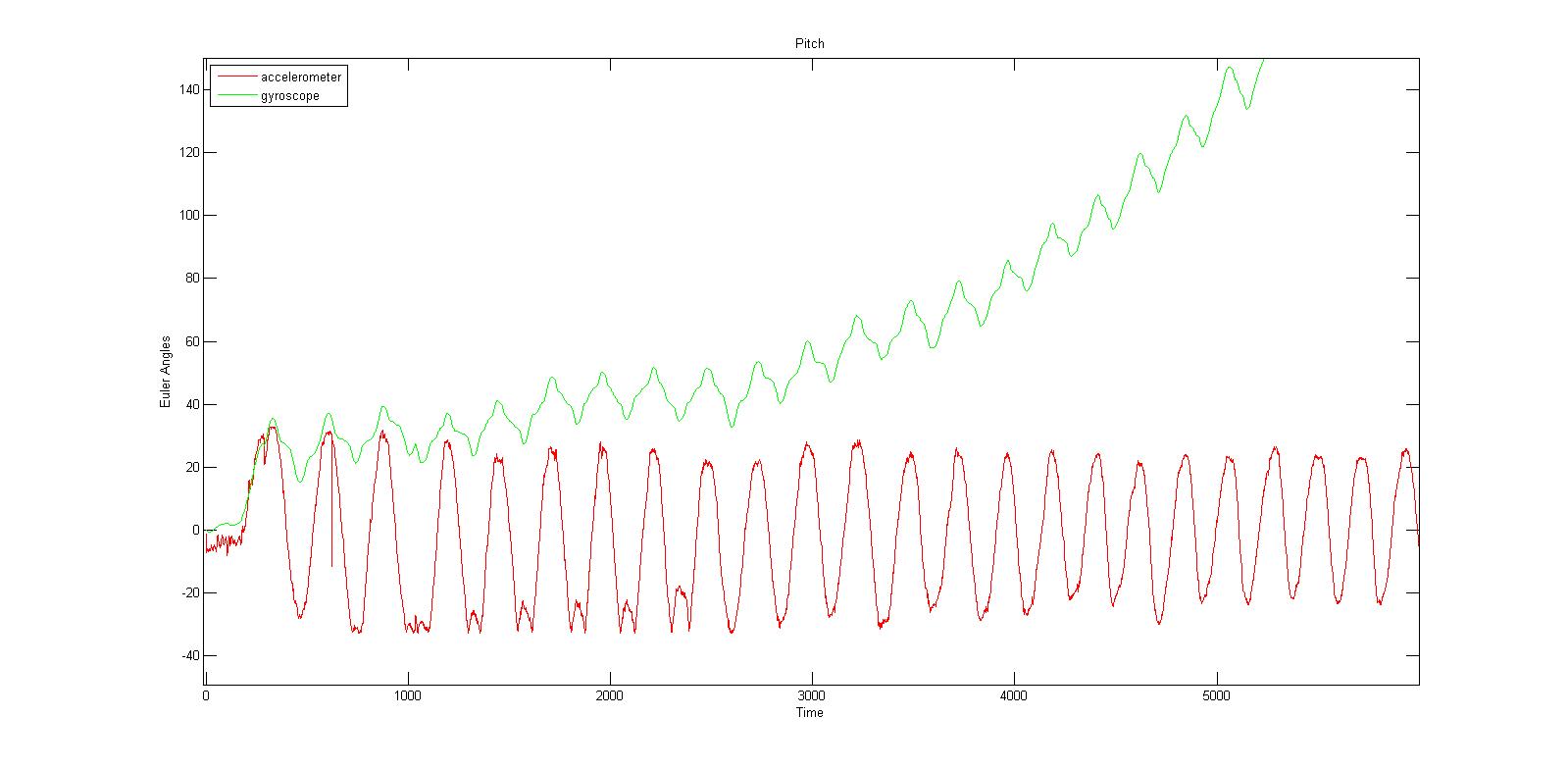


Figure 23: Pitch Angle

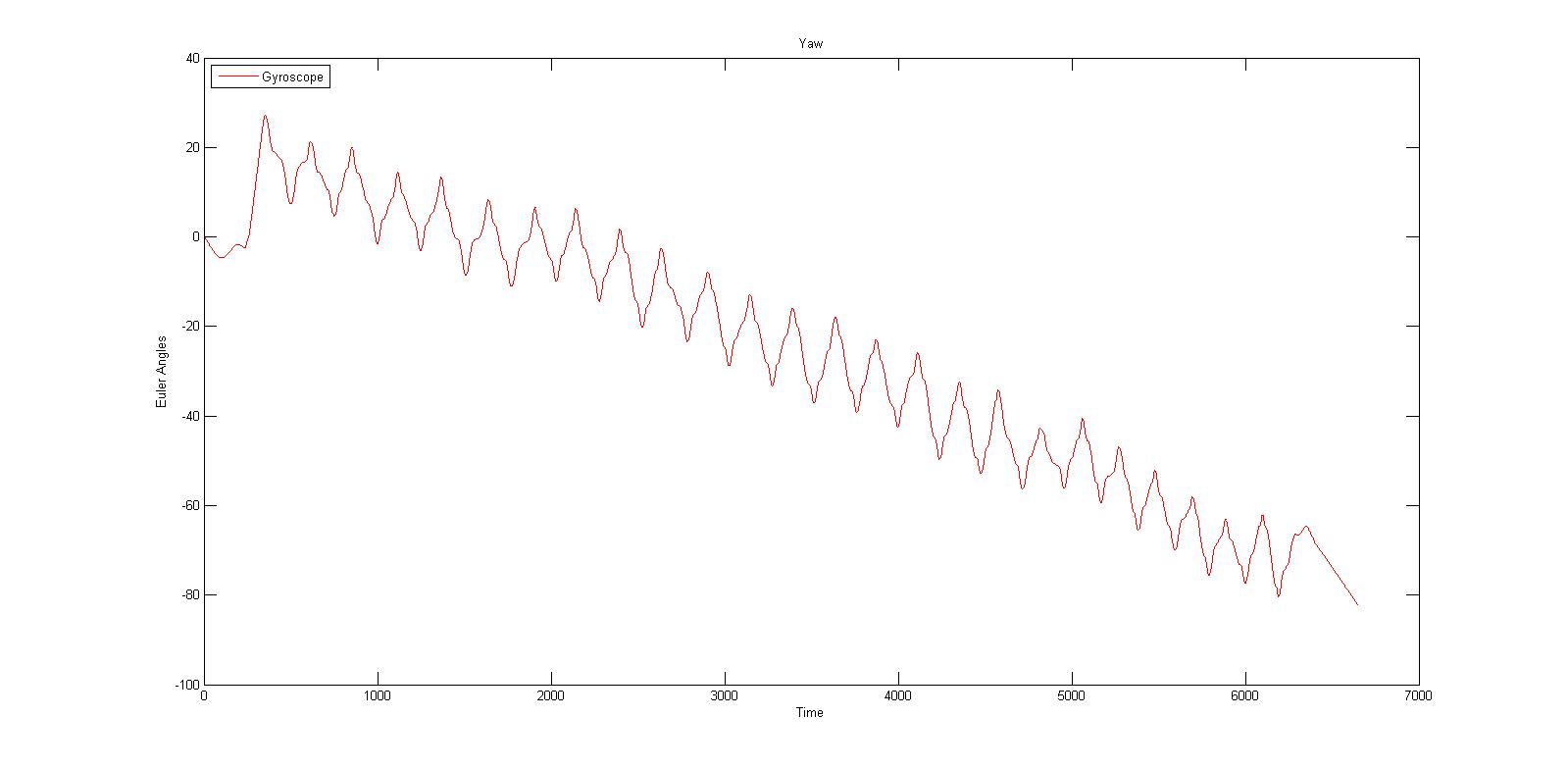


Figure 24: Yaw Angle

All of the testing setups are the same: keeping the body of the IMU oscillates around a point. However, the results do not seem to be good enough. The accelerometer calculated angles are too noisy and the Gyroscope calculated angles are drifted too much.

After a throughout investigation on the problem, the truth has been revealed:

- Due to the extremely sensitive characteristic of the gyroscope sensor, the noise in angular velocity data is generally unavoidable and when the gyro data is integrated, the noise will also be integrated together. Furthermore, the gyroscope has its limitation where the output is not a constant offset when it is in static condition. In fact, this value will keep changing especially when there is temperature change. This condition is called **drift**. Although the drifting is very small, when we are dealing with integration, even the smallest offset will cause the integrated data to grow to infinity. To cut the long story short, the angular movement calculated by Gyroscope data is nice at a moment but extremely inaccurate in a long run.

- In contrast, accelerometer data is quite good in a long run but fluctuating too much in a short period. The reason behind this is lying in the assumption of equation (23) and equation (24)which assumes the only force affect Accelerometer is Gravity Force. However, in operating environment, there are many other random forces that may be included in the calculated data such as wind, vibration... Therefore the angular movements measured by accelerometer have quite a lot of random noise.

A solution for this problem is proposed: mathematical Filters.

## Mathematical Filter

As mentioned above, angular data measured from Gyroscope is acceptable at a moment but become extremely inaccurate after being integrated. Naturally, a high-pass filter [15] can be applied to regulate the long-run data with reasonable values.

Nevertheless, the noisy characteristic of accelerometer's angular data can be adjusted with a low-pass filter[16], in which the signals that is much longer than the time constant pass through unaltered while signals shorter than the time constant a filtered out.

In the following figures, the test data in part 4.2 are recalculated using Low-pass and High-pass filter:

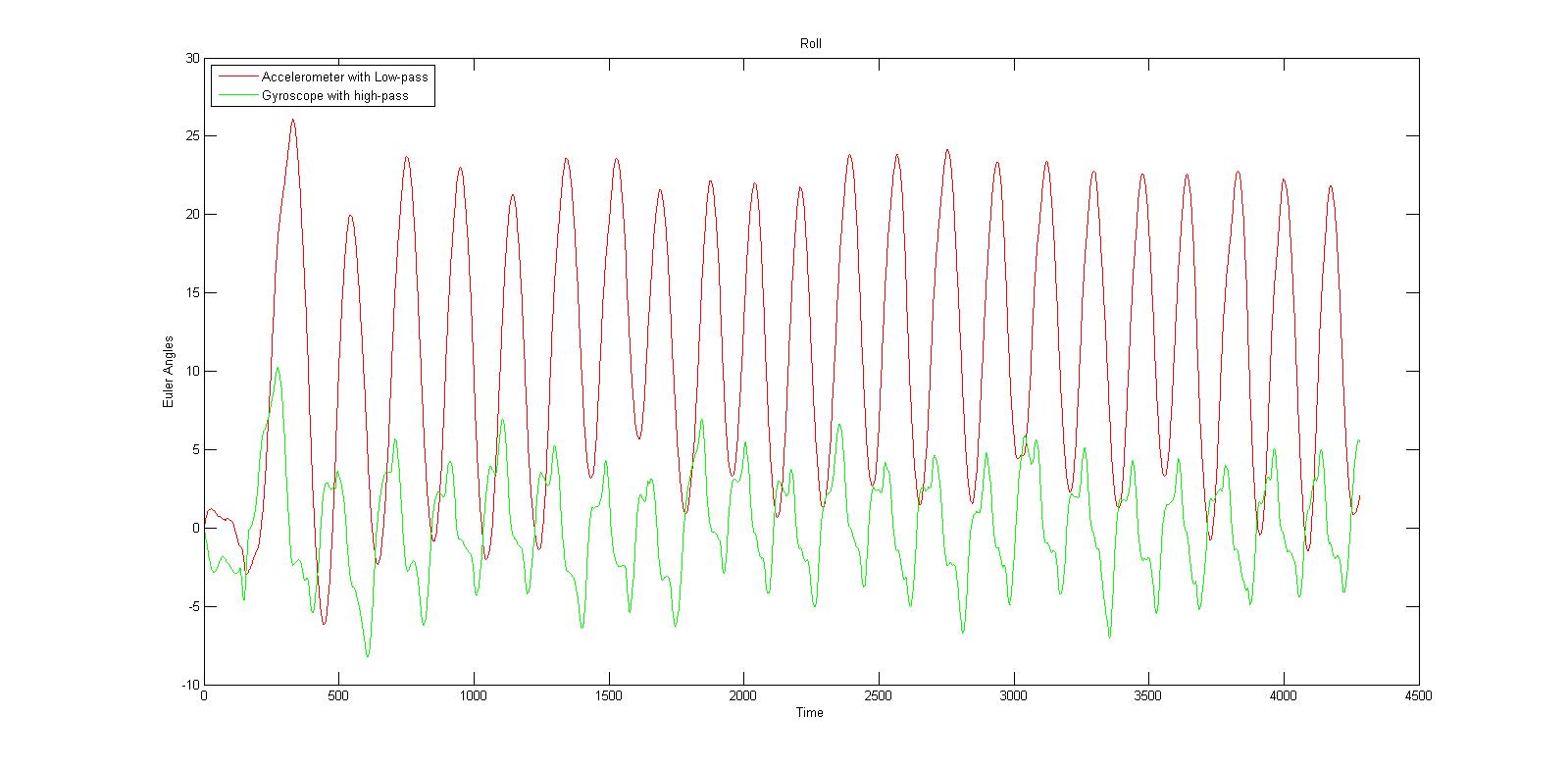


Figure 25: Roll Angle with low-pass and high-pass filter

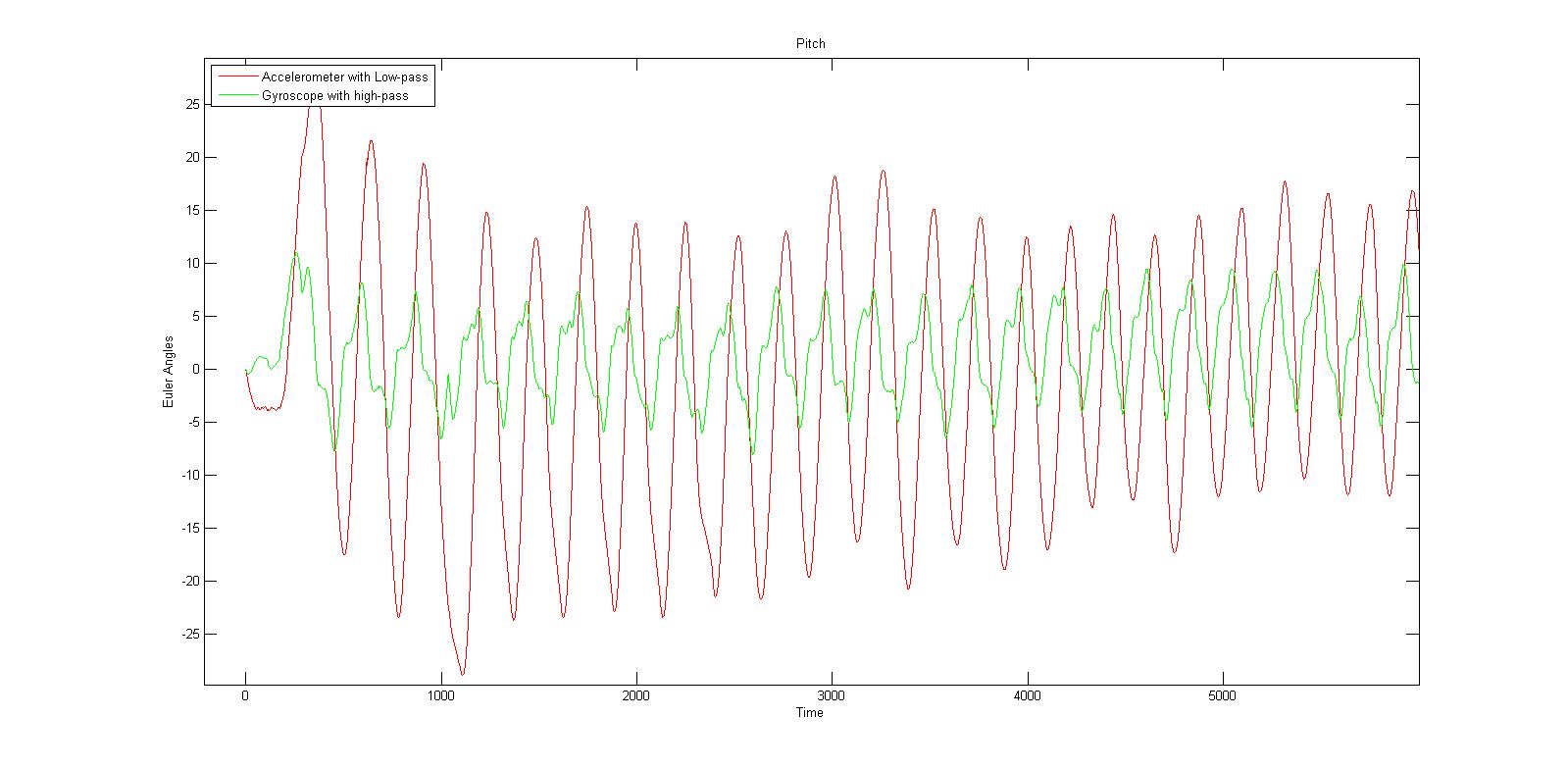


Figure 26: Pitch Angle with low-pass and high pass filter

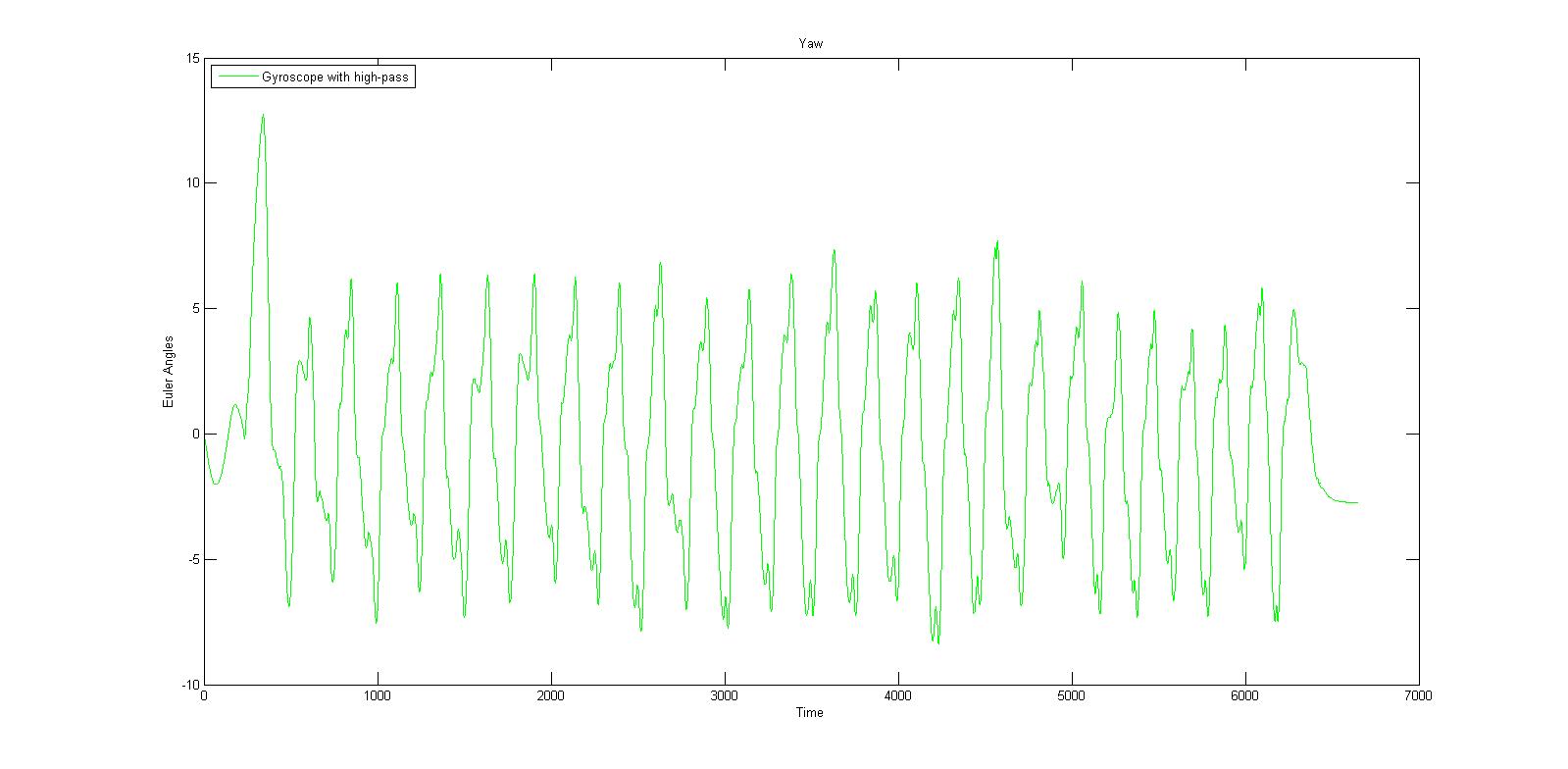


Figure 27: Yaw angle with a high-pass filter

The results became more reasonable after applying these filters now. However, the low-pass filter will increase the latency and slow down the response time of the measurement while high-pass filter require a lot of change in the input data to change its output a little bit and tend to forget prior output value quickly. Consequently, a data fusion filter of the two sensors sounded promising.

In this spectacular case, a Kalman filter is what the data needs. However, implementing such a complex filter will be far too much for a regular MicroController to handle, an alternative solution has proposed: Complementary filter [17].

A Complimentary filter is just a fusion of a Low-pass and a High-pass filter with a mathematical expectation of influence between them. More information on this filter can be found in "*The Balance Filter*" by Shane Colton, MIT [18].

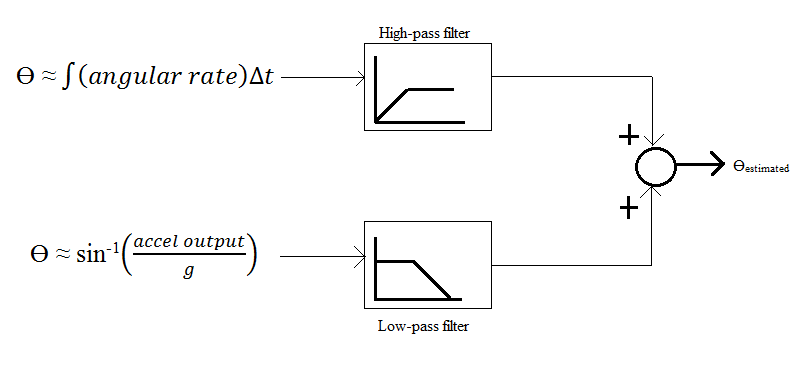


Figure 28: Complementary filter

The figures below shown the result when applied Complementary filter to Pitch and Roll angles.

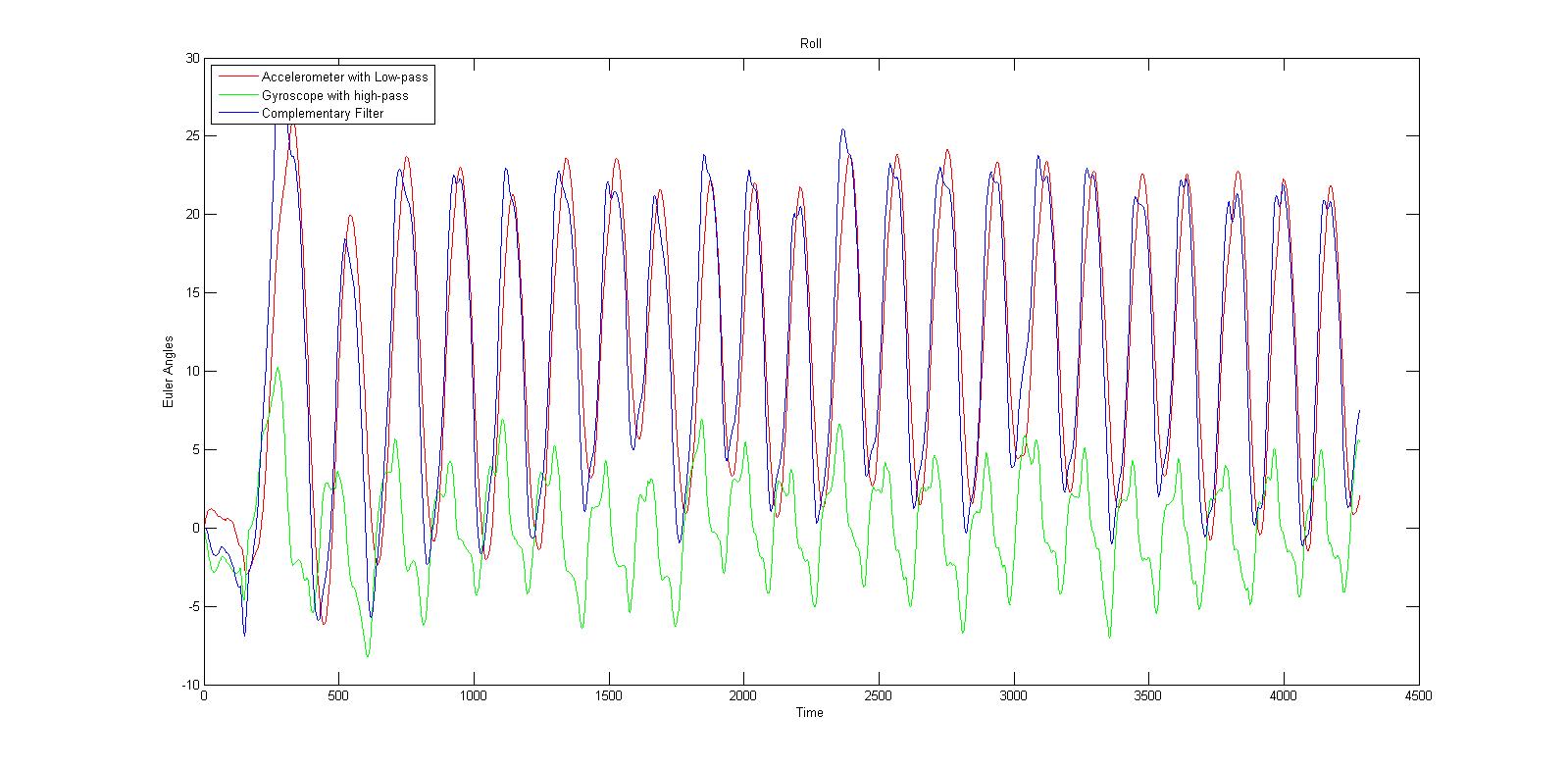


Figure 29: Roll angle with Complementary, High-pass and low-pass filter

As we can see in figure 29, Low-pass filtered data from Accelerometer is a little bit shifted compare with High-pass filtered data from Gyroscope. After these too data combined with a Complementary filter, the data (blue) becomes very smooth and is not shifted again.

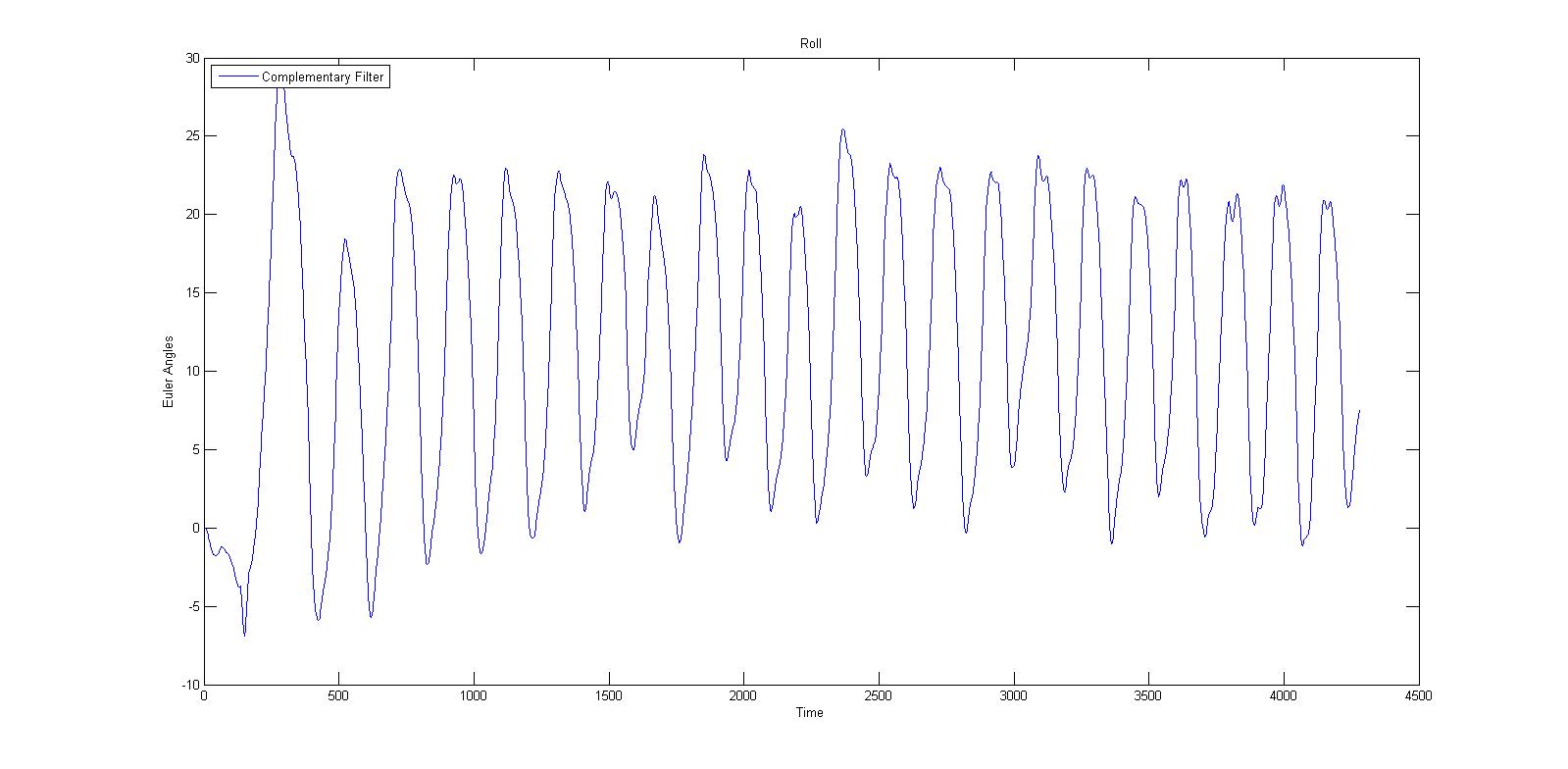


Figure : Roll angle with complementary filter only

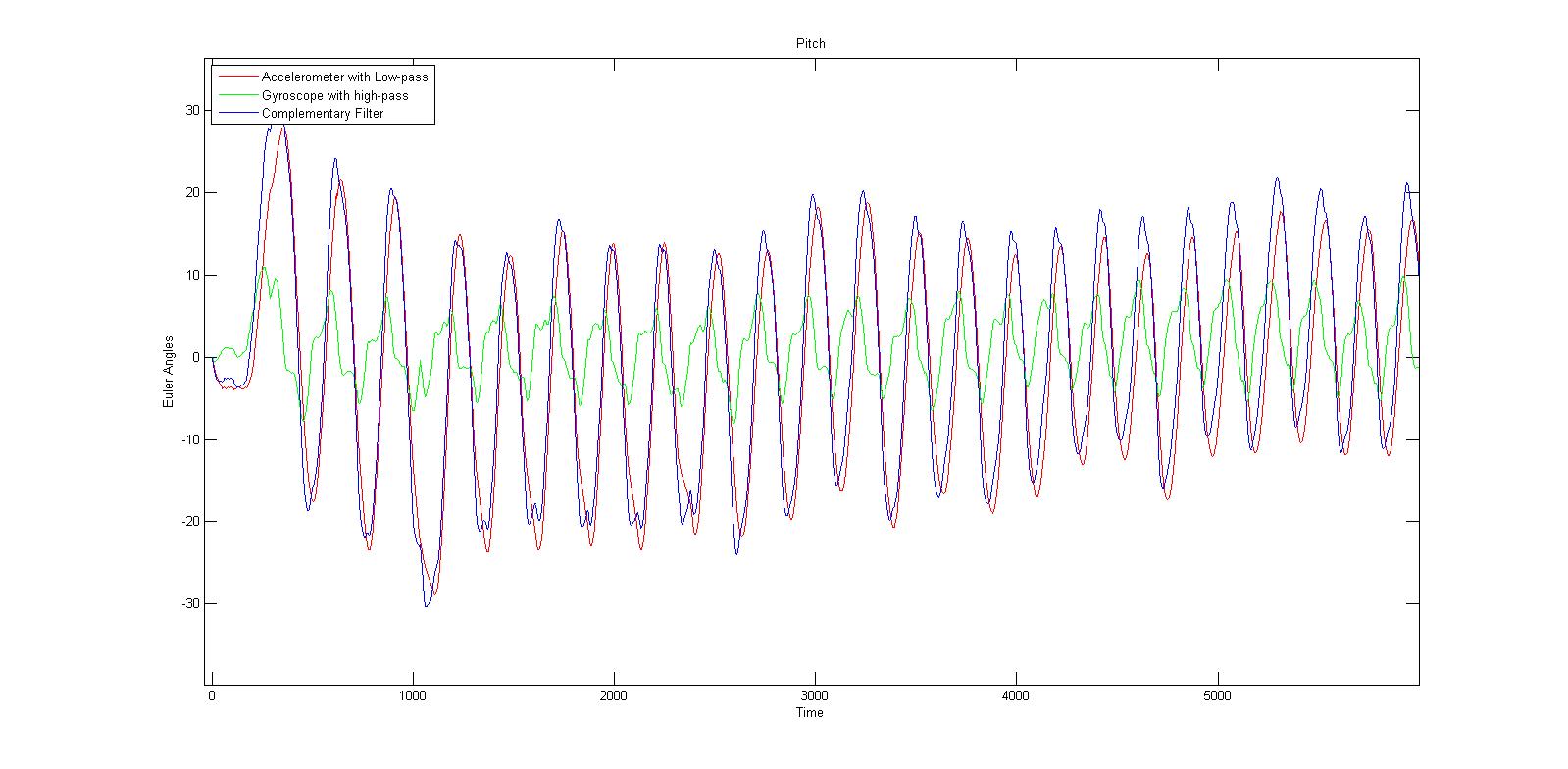


Figure : Pitch angle with Complementary, High-pass and low-pass filter

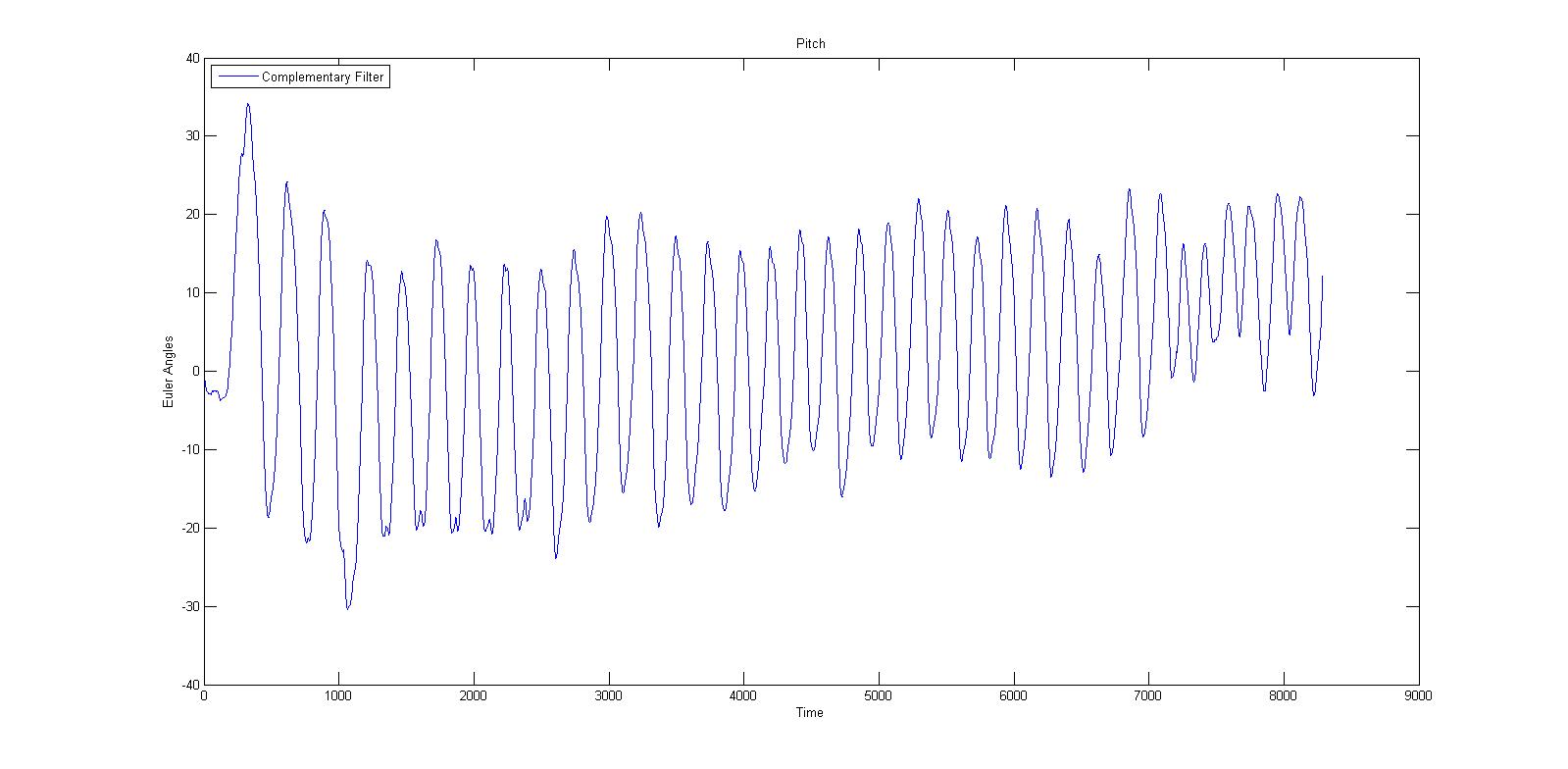
****

Figure : Pitch angle with Complementary filter only

There is no complementary filter for yaw angle in this case because only gyroscope data is used in the calculation process. However, if a magnetometer sensor is implemented, the complementary filter for yaw angle could be used as well. Moreover, in this prototyped system, small drifts in yaw angle are acceptable since it will not affect the movement much.

## Altitude measurement

The altitude of an IMU can be measured with a barometer. In theory, with the pressure and temperature, current altitude compare with sea-level can be calculated. Figure below using the equation provided by manufacturer of BMP085 barometer to calculate height.

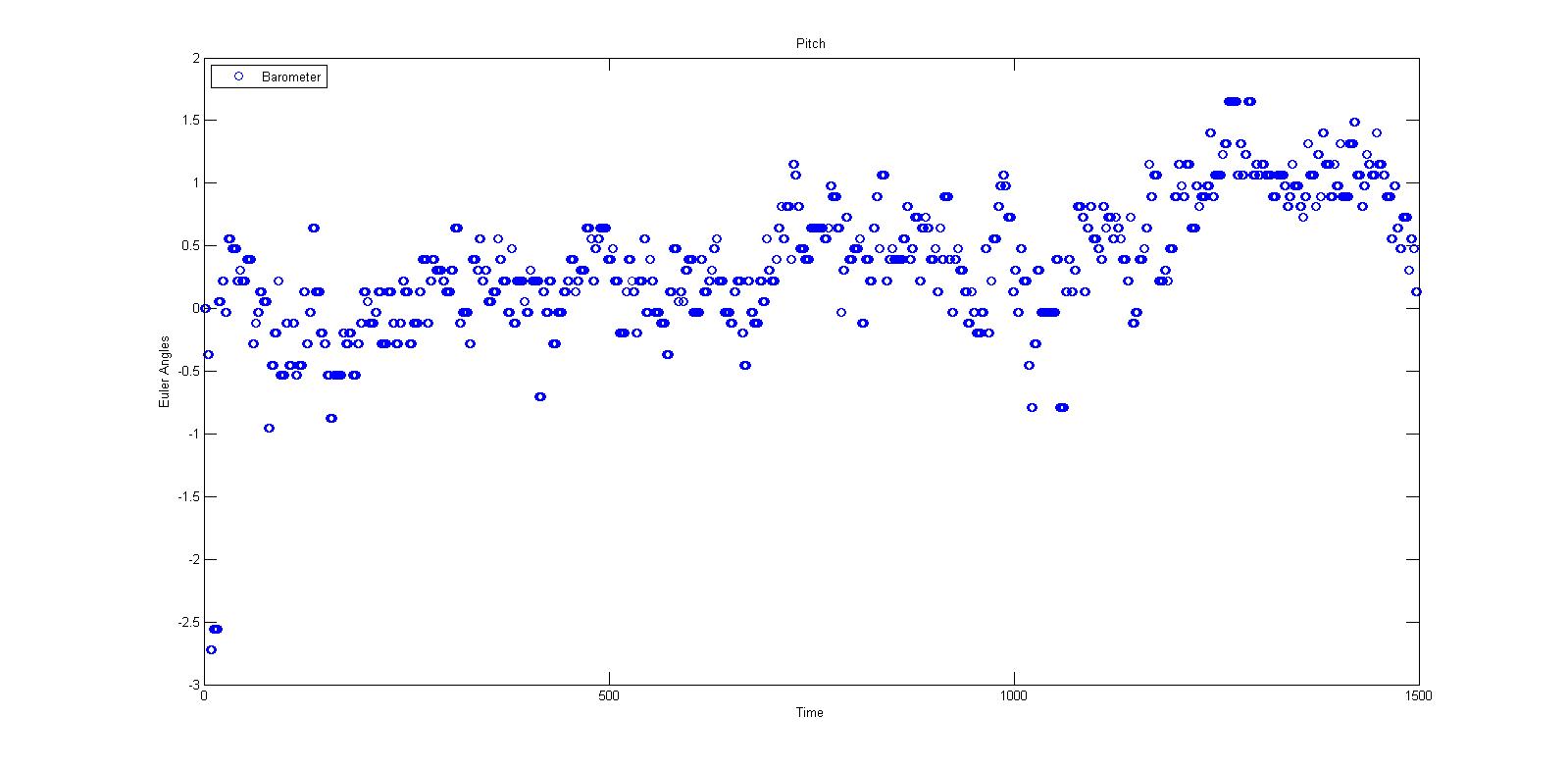


Figure : Altitude measurement using barometer

The result reveals a noisy characteristic of this signal. There are many possible causes for this. A sudden change in temperature is one good example. In operating environment, temperature may vary in space.

To solve this problem, a simple Low-pass Filter has been applied. Despite the incorrect values a low-pass filter could lead, almost noise has been filtered out of the data. This results in a more meaningful form of data.

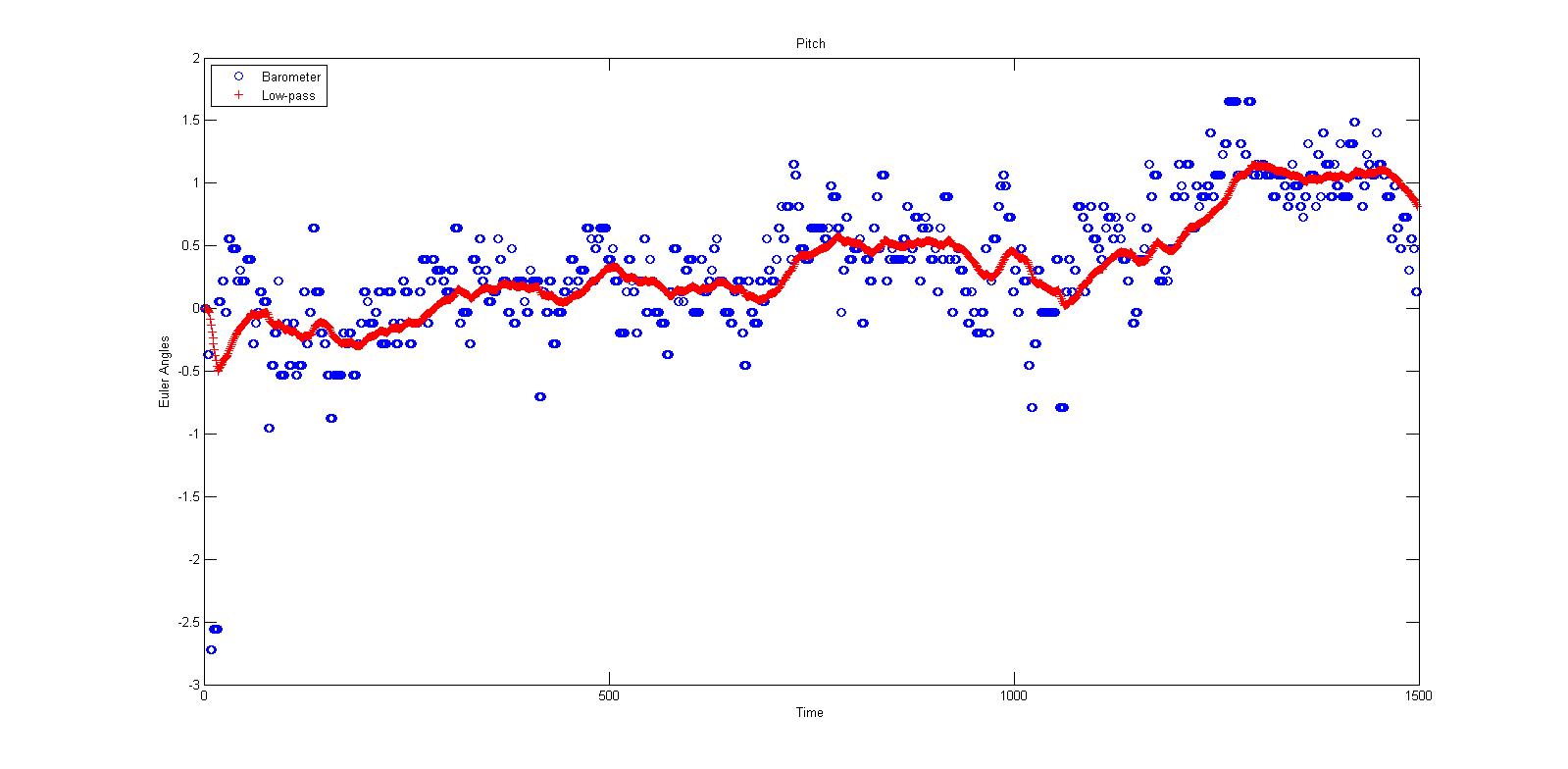


Figure : Altitude measurement with Low-pass filter

Even the data now seem to be less noisy, in fact, it still has an error range from 0.5 m to 1m. Another method has to be suggested to correct this data. However, due to the shortage of study time, the team could not try any other method but this. If an appropriate time was given, maybe a further study on calculating altitude using Accelerometer would be reasonable.

## PID Controller

A classic approach for a controller is using Proportional-Integral-Derivative (PID) control system. The PID controller is a closed-loop feedback system which will output a control signal *u* and receive feedback from the inertial sensors. The controller then calculated the difference between the desired position and orientation and the current position and orientation and adjusts *u* accordingly. The equation for a PID controller can be found in "*The PID Algorithm - how it work and how to tune it"*[19]*-*John A.Shaw is as follows:

u = Kpe(t) + Ki + Kd (38)

Define:

e(t) = ed(t) - ea(t) (39)

Where ed is the desired condition and ea is the actual condition measured by sensors and e(t) is the difference (error) between the two at each individual time step.

The following dynamic equations of motion are rewritten from equation 32 in matrices.

+ ∆ = (40)

Where the disturbance, ∆, is defined as:

∆ = (41)

Where δi mainly come from the dynamic inconsistency. Hence, it does not give instability, but poor performance, which may violate the assumption, *φ* ≈ 0 and *θ* ≈ 0. This phenomenon can be resolved using Disturbance Observer control input. However, Disturbance Observer method will make the prototype Quadrocopter becomes too complex to be done in study period, therefore it is not discussed in this study. Moreover, the impact of lacking Disturbance Observer is still acceptable in the scope of this study.

Finally, the PID control system of the Quadrocopter can be derived from equation 30 to 40 as in the follow figure:

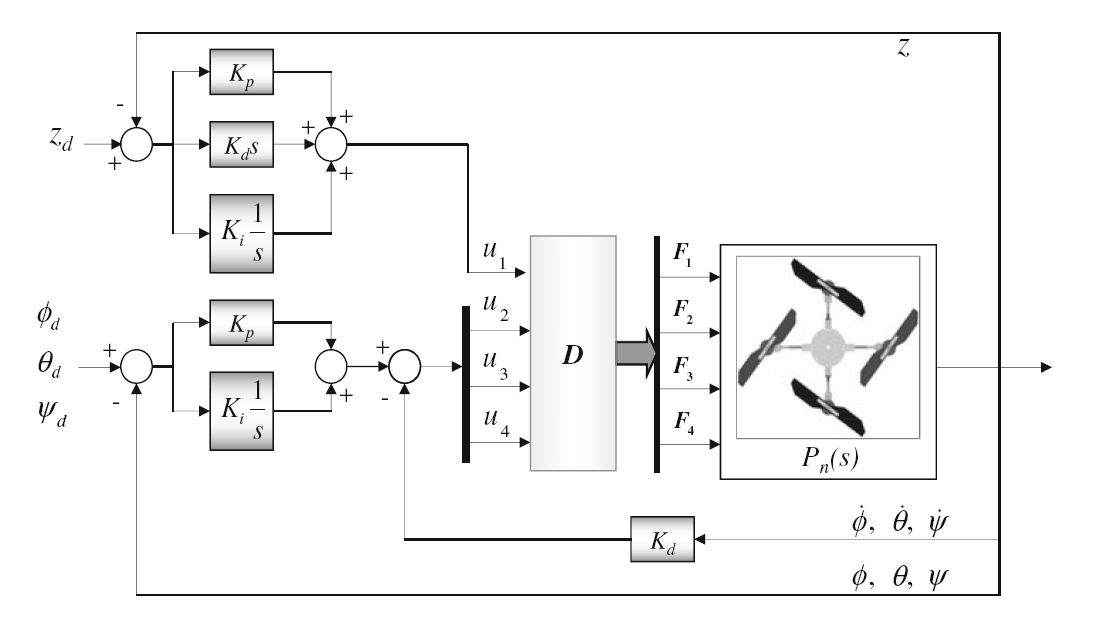


Figure : PID controller for controlling the Quadrocopter

Nevertheless, this is just theory and it still makes no sense for software engineer. Therefore, a further explanation in software term will be discussed base on figure 35.

Firstly, the following equations are derived from equation 38 and 40 in discrete term:

u1 = Kp\*Ezn + Ki\*i\*dt + Kd\*dEz/dt (42)

u2 = Kp\*E𝜑n + Ki\*i\*dt + Kd\*d E𝜑' (43)

u3 = Kp\*E𝜃n + Ki\*i\*dt + Kd\*dE𝜃' (44)

u4 = Kp\*E𝜓n + Ki\*i\*dt + Kd\*dE𝜓' (45)

where Ez, E𝜑, E𝜃, E𝜓 are:

Ez = Ezd - Eza (45)

E𝜑 = E𝜑d - E𝜑a (46)

E𝜃 = E𝜃d - E𝜃a (47)

E𝜓 = E𝜓d - E𝜓a (48)

E𝜑', E𝜑', E𝜑' are angular velocity measured from Gyroscope.

Secondly, in a real Quadrocopter system, F1, F2, F3, F4 cannot be calculated because each motor has an unique motion equation that depends on the manufacture process or even the age of the motor. In addition, the speed of the motor should be controlled via an Electronic Speed Controller (ESC) system. In the prototype Quadrocopter, the ESCs are controlled via Pulse Width Modulation (PWM) method. This leads to the assumption that each PWM level will map to a force level of a motor.

PWM1 => F1,

PWM2 => F2,

PWM3 => F3,

PWM4 => F4

(49)

Finally, the following equation is derived from equation 22 and 49 for distributing inputs into four motors.

PWM1 = u1 + u3 + u4 (50)

PWM2 = u1 - u2 - u4 (51)

PWM3 = u1 - u3 + u4 (52)

PWM4 = u1 + u2 - u1 (53)

## PID Tuning

One of the most frustrate part in designing an embedded system is tuning the controller values. There are many approaches to optimize those values. At first, the team chose a classical PID Tuning method: Ziegler-Nichols second method[20].

However, during the PID tuning process of the Quadrocopter, Ziegler-Nichols second method begins to reveal its weakness in this kind of system: there is no way to find a perfect Ultimate Gain because the Quadrocopter will never oscillate perfectly in operating environment which has too many other random forces.

The PID tuning process has to use another approach which is based on practical meanings of the PID controller:

|  |
| --- |
| Derivative term:  D = Kd (54)  The derivative of the process error is calculated by determining the slope of error over time and multiplying this rate of change by the derivative gain Kd. It slows down the rate of change of the controller output, which is used to control the magnitude of the overshoot produced by the integral component and improve the combined controller-process stability.  This leads to a hypothesis that if only Derivative term with a big enough Kd is used in the Controller system, the Quadrocopter will likely to stay in the approximately same position given how many time it is affected by an external force. The faster the rate of change is the faster the derivative term react, and if the changing rate is low, so the reaction. (I) |
| Proportional term  P = Kpe(t) (55)  The proportional term produces an output value that is proportional to the current error value. The proportional response can be adjusted by multiplying the error by a constant *Kp*. A high proportional gain results in a large change in the output for a given change in the error. If the proportional gain is too high, the system can become unstable and may oscillate.  An important characteristic of this term is the high speed of change given a high enough gain value. This is called a fast respond. (II) |
| Integral term  I = Ki (56)  The contribution from the integral term is proportional to both the magnitude of the error and the duration of the error. The integral in a PID controller is the sum of the instantaneous error over time and gives the accumulated offset that should have been corrected previously. The accumulated error is then multiplied by the integral gain (Ki) and added to the controller output.  The most important characteristic of this term is: it will always try to get the error be equal to zero. (III) |

With the characteristics I, II and III, a good PID tuning strategy for Quadrocopter system can now be illustrated by three steps:

**Step 1**: Tuning only the Derivative Term by setting Ki and Kp to zeros then slowly increasing Kd until the Quadrocopter keeps its angle/altitude in a specific level without moving or oscillating around. If Kd is high enough, the Quadrocopter will return to its specific level aggressively after being pushed/pulled suddenly.

**Step 2**: Tuning Proportional Term by setting Ki to zero but still use Kd which was found in step 1, then slowly increasing Kp. Please keep increasing Kp until you experience a fast and stable respond from the Quadrocopter toward the Set-point. May be the Derivative Term you found on the previous step will try to slow down the rate of change made by Proportional Term, but it is just because Kp is not high enough and you can keep increasing it. When to stop this step is just your decision in real practice.

**Step 3**: Tuning Integral Term by slowly increasing Ki until the Quadrocopter reach its Set-point with a reasonable speed, which only can be evaluated in real practice. A too high Ki value could make the Quadrocopter become extremely unstable under certain circumstance.

Pros and cons of this tuning strategy:

|  |  |
| --- | --- |
| Pros | Cons |
| - Easier to apply than any classic PID tuning method.  - Ensure the safety of the Quadrocopter during the tuning of Proportional Term. | - Cannot be applied in mass production.  - Depend much in experience of tester.  - Is not a mathematical recommended method for tuning PID. |

Conclusion: This strategy is a good start for beginner, however it will soon become unusable when it comes to mass production of Quadrocopter. Another method which involves more complex math should be applied to tune PID in the future.



Figure : Tool for testing PID on Pitch and Roll

Another problem for tuning PID in a Quadrocopter is an appropriate tool for testing. In this project, the team has to make a tool for testing Pitch and Roll angles. This tool must help the platform operate in an environment that as similar to the flight environment as possible. It has a frame made of steel and two ball bearing to hold the Quadrocopter like in figure 36. The tool has proven to be a good solution for PID tuning of Quadrocopter, thus leaving as a strongly recommended tool for this process.

The following figures illustrate the results of PID tuning using this strategy with Pitch angle:

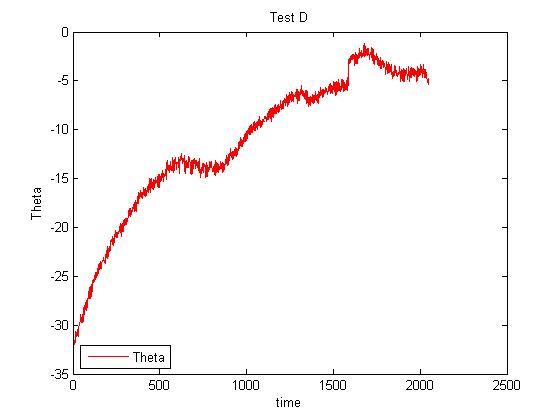


Figure : Theta angle control using only Derivative term

Figure 37 illustrate how the Quadrocopter react on Pitch (Theta) angle. It rose gradually to an angle that related to its orientation of center of mass, and then remain fluctuated in there. This fluctuation is relatively small. During the observation, we could only see that it is holding still in a certain angle.

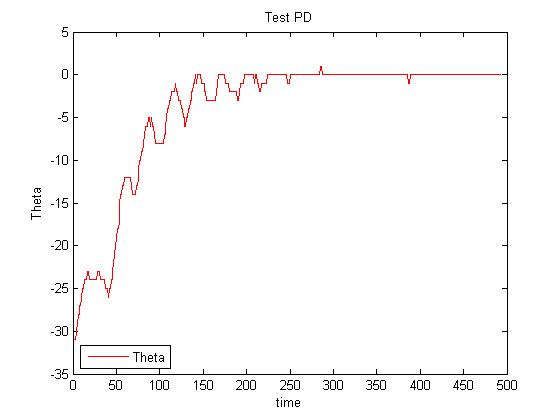
****

Figure : Theta angle control using Derivative term and Proportional term

The test given in figure 38 with Proportional gain and Derivative gain controller is far better than the test with only Derivative term in figure 37. As the figure given, almost no noise was recorded.

However, despite the system seem to be able to hold set-point at a stable rate, an Integral term should still be implemented. The only reason why this system seems to be perfect with only a PD controller is that this Quadrocopter frame itself is designed to be perfect balanced without adding any additional force. During real flight, there will be a lot of external force that will affect the system and a PD controller is not good enough to protect the Quadrocopter from being damaged. Moreover, Integral term will always ensure that the system moves to and stays at Set-point regardless of any minor physical change as shown in figure 39.

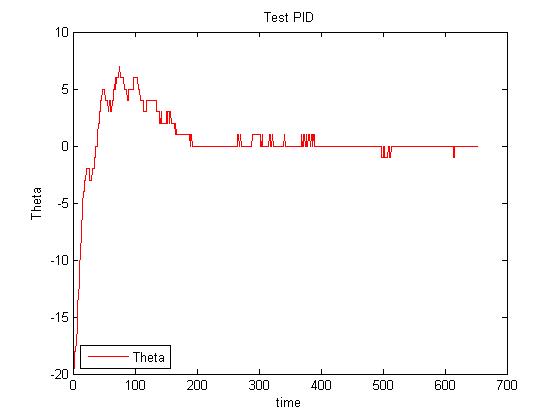


Figure : Theta angle control using all PID term

## Software and firmware design

### Software system architecture



Figure : Use case diagram of FUFO system

The system will be divided into three components:

- The Application on Computer (AOC)

- The Application on Phone (AOP)

- The firmware of Quadrocopter.

Each component is complete independence software on a relevant platform.

The figure below shows the Component model of the whole software system.



Figure : Component model of FUFO System

The FSM of the system is illustrated below:



Figure : FSM of the whole system

This FSM is applied to the whole system, however, in each specific platform, it will have a variant that pays attribute to the main states. The variants FSM on each platform is provided in Appendix A.

### Software on Phone and Computer

Detail design and description of the software on Phone and Computer can be found in **Appendix B**. The following activity diagram is derived from Use case and Component diagram of FUFO system.

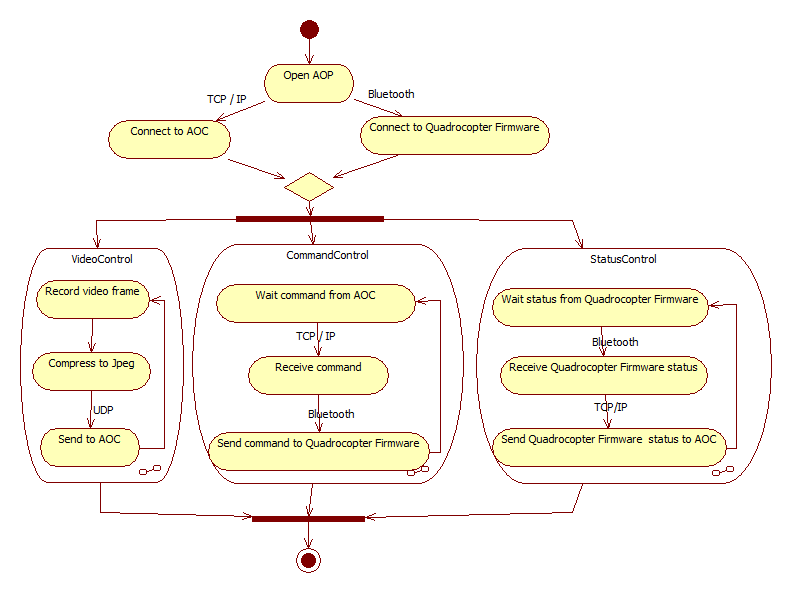


Figure : Activity diagram of application on Phone

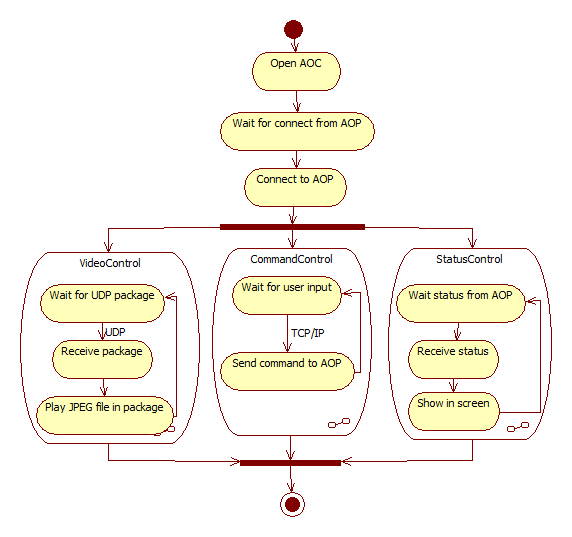


Figure : Activity diagram of software on Computer

### Firmware

The firmware on Quadrocopter must use a variant of the FSM shown in Figure 44 because it is just a slave device. The states of the system can only be applied to it when it has connection with PC and Android Phone. Figure 45 illustrates this FSM:



Figure : FSM model of FUFO system

To be more specific, this FSM will be described fully in the figures below:

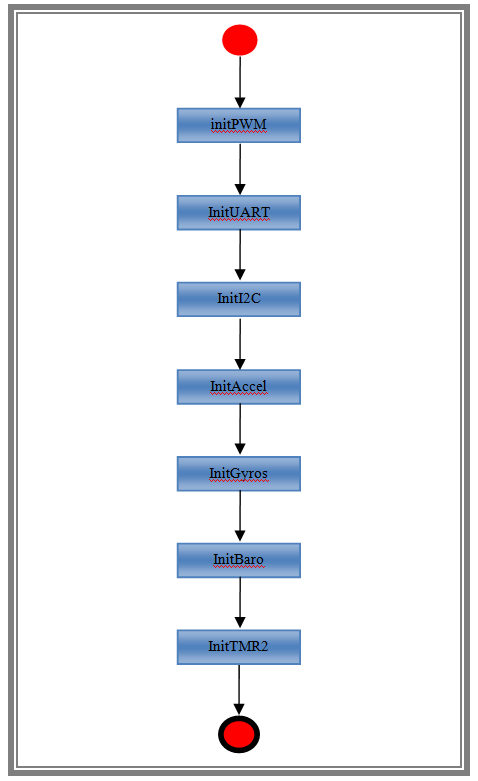


Figure : State "Start" diagram

After power is turned on, firmware enters its first state: “Start”. In this state, it initializes communication methods such as I2C, UART, and modules such as PWM, Accel, Gyros, Baro, TMR2.

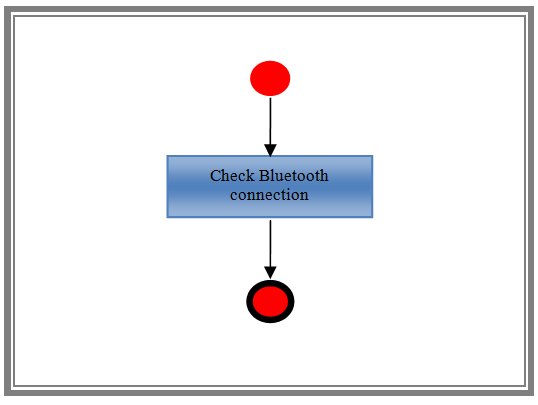


Figure : State "Start" diagram

Firmware checks Bluetooth connection between Quadrocopter and Phone.

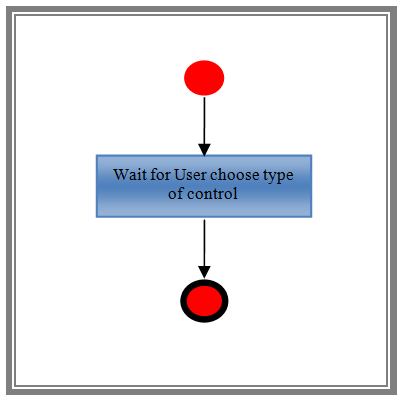


Figure : State "Verify" diagram

Waiting for user to choose between two modes: Control by phone or Control by PC.

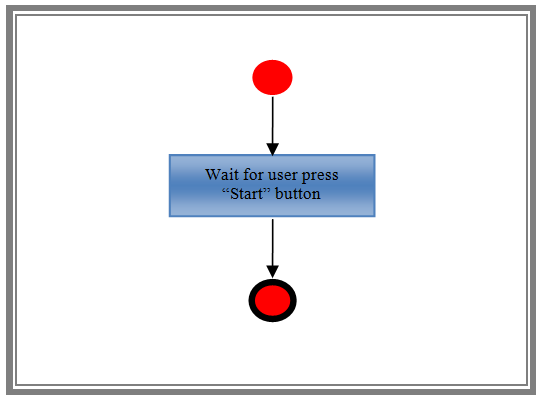


Figure : State "Pending" diagram

In this state, firmware waits in a loop until Start button is pressed.

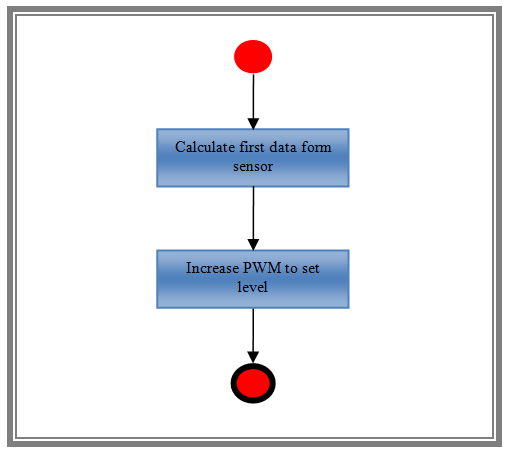


Figure : State "SetupForFlight" diagram

Firmware increases PWM to a base level (40% of thrust level) and calculates data received from sensors to reduce noise of gyroscope data.

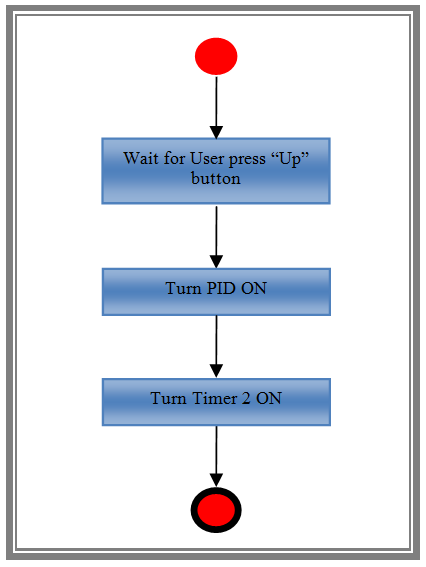


Figure : State "Ready" diagram

After finishing increasing PWM and calculating sensor data, firmware switches to state "Ready". In this state, firmware will wait for user to press “Up” button to increase the altitude Set-point of Quadrocopter, turn on TMR2, and turn on PID so the Quadrocopter can rise to that altitude (0.5 m).

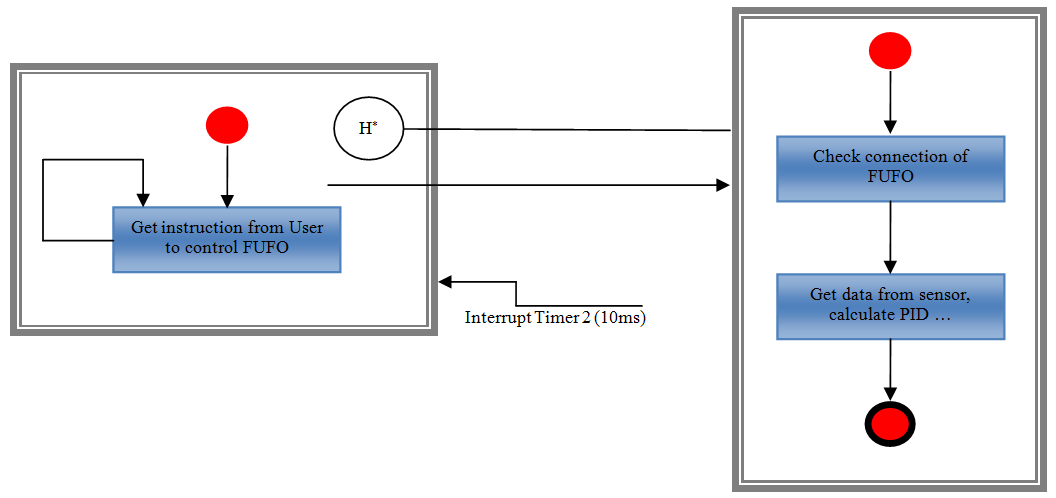


Figure : State "Hovering" diagram

In state "Hovering", firmware gets instruction from user (Up, Down, Left, Right, Backward, Forward, Level up, level down). If timer interrupt of Timer 2 occurs (in every 10ms), firmware will calculates data received from sensors to control the Quadrocopter.

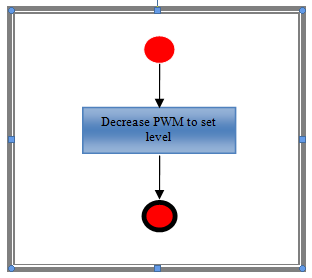


Figure : State "Landing" diagram

If user sets altitude Set-point to 0m or the Quadrocopter loses it connection to phone, firmware will enter state "Landing". It will decrease PWM to base level (40% of thrust level), and then enter state "Pending" to setup for the next flight or just go to state "Ending".

The above state diagrams of firmware FSM described clearly how the firmware behaves in FUFO system. However, this FSM is not clear enough for understanding the PID control flow of the controller, which works with angular changes and altitude changes. The following control flow describes it perfectly.

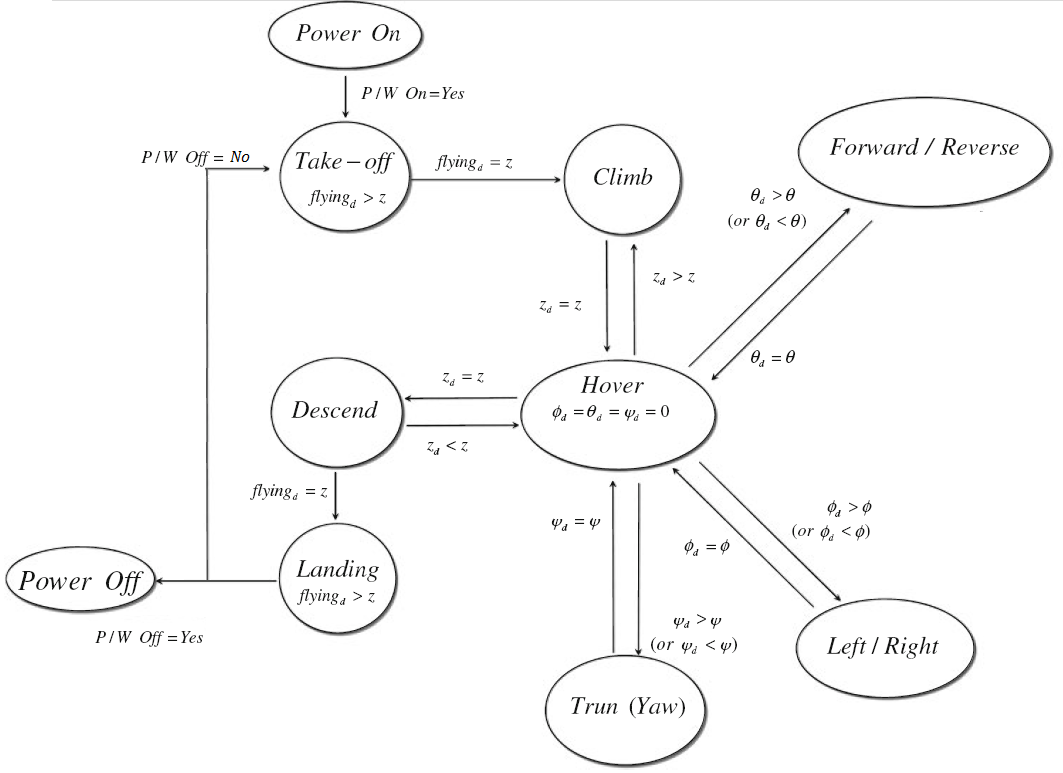


Figure : Control flow of the controller

Figure 54 shows the control flow of this robot system. All flying movement except for takes-off start on the hovering state, and if a certain movement is finished, it will come back to the hovering state again.

The firmware contains many risks in developing it. One of the most noticeable risks is hardware related problems. During the development process of FUFO project, the team has to face and solve many problems caused by unstable hardware, which can affect both firmware stability and algorithms reliability. Those problems and solutions for them are given in **Appendix C** of this report.

# Experimental results

## Experimental setup

The prototyped Quadrocopter has been done and is ready to fly.



Figure : The Prototyped Quadrocopter

Figure 55 shows the fabricated Quadrocopter for purpose of testing its flight capability regards to the theory. It has 4 rigid propellers driven by 4 brushless motors mounted at the end of a crossing body frame. An IMU which has an accelerometer, gyroscope and barometer is equipped to the main circuit for sensing angular and altitude changes.

Table 1 gives the specification and parameters of the Quadrocopter frame:

|  |  |
| --- | --- |
| Description | Value |
| Weight | 1.4 Kg |
| Diameter | 490 mm |
| Height | 150 mm |
| Distance between the motor and the Center of Mass | 220mm |
| Propeller | 8" x 4" |

Table : Quadrocopter frame specification



Figure : PC software Interface

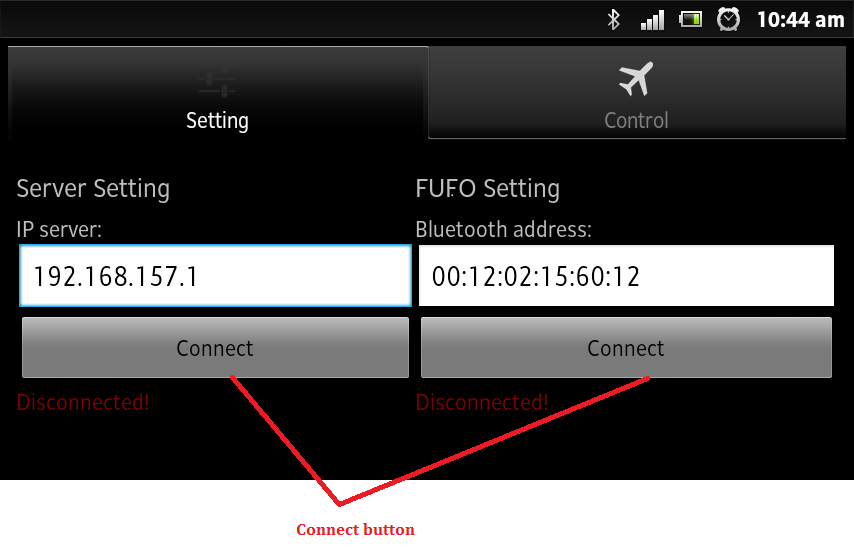


Figure : Tab 'Setting' on Android Application

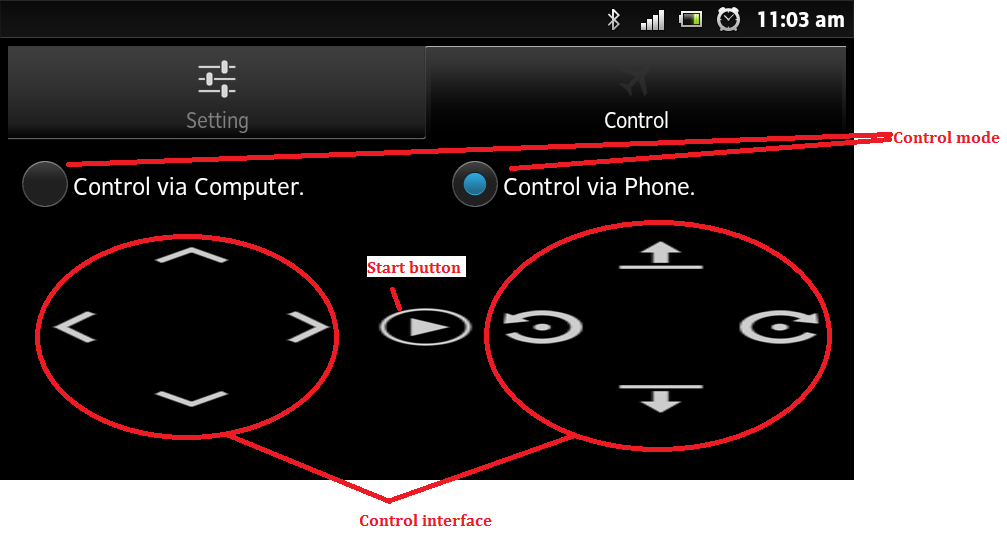


Figure : Tab 'Control' on Android Application

The environments for experiments are both indoor and outdoor.

## Experiments results and analysis

Test cases for both indoor and outdoor experiments of this prototype are just the same:

- Take-off.

- Hold a certain altitude.

- Set set-point for theta angle to -16o.

Figures 59 and 60 show the experimental results for indoor hovering performance in term of roll, pitch, yaw angles and altitude.

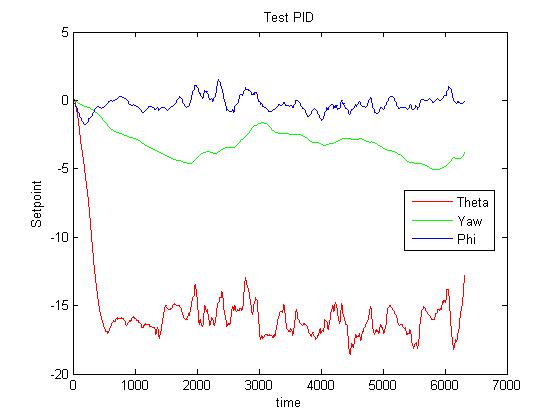


Figure : Test statistics of indoor flight

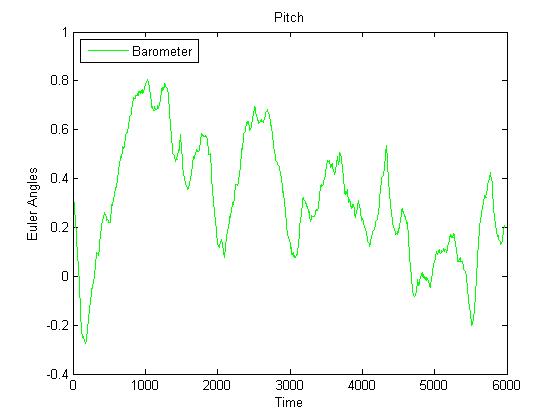


Figure : Test Altitude in room

Indoor flights without wind affection of this prototype are generally good. However, sometime PID controller still cannot fix the drift of the vehicle. This is mainly because of the low response speed of ESCs. Another problem of indoor flight is the inaccurate of Barometer since it can be easily affected by an air conditioner.

Figures 61 and 62 show the experimental results for outdoor hovering performance in term of roll, pitch, yaw angles and altitude.

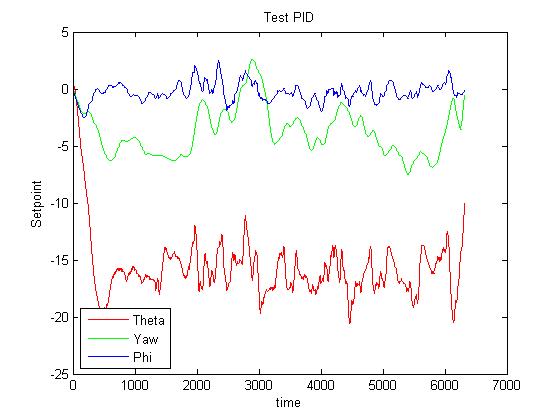


Figure : Test statistics of outdoor flight



Figure : Test Altitude outside

Overall, outdoor flight shows some descent fluctuations under the effect of free wind. However, it still can hold its set-point pretty well. Barometer measurement data seem to be more accurate outdoor than indoor.

The results declare that the proposed algorithm and theory for both angular movements and altitude control of the Quadrocopter work as well as expected.

# Conclusion and suggestion for future development

In this study, a low-cost prototyped Quadrocopter is developed to evaluate its flight capability using a proposed control theory and algorithm. The rigorous dynamic models of a Quadrocopter were obtained both in the reference and body frame coordinate systems using the Quasi-Lagrange equation. The experiments were carried out to show the validity of the proposed control algorithm. The results show the angular movements and height control results are as stable as expected for a cheap system.

There are still some tasks that this prototype Quadrocopter system cannot archive compares with a commercialized Quadrocopter system, thus leave the clear path for future development:

- Find better method for PID tuning.

- Develop a higher response system.

- Hold altitude in narrow and low area with precision of +-0.1 m

- Hold a specified position on map or moving on a track.

- Obstacle detection and avoidance.

- CMOS camera's video transmission over long distance.

- Gimbals stabilization camera holder.

- Object detection based on image processing.

# List of References

|  |  |
| --- | --- |
| **No.** | **Source** |
| [1] | "**UAV**" - http://en.wikipedia.org/wiki/UAV |
| [2] | "**Quadrotor**" - http://en.wikipedia.org/wiki/Quadrocopter |
| [3] | "**Electronic speed control**" - http://en.wikipedia.org/wiki/Electronic\_Speed\_Controller |
| [4] | “**PID Controller**” - http://en.wikipedia.org/wiki/PID\_controller |
| [5] | Jinhyun Kim, Min-Sung Kang and Sandeok Park (2009) - "***Accurate modeling and robust hovering control for a Quad-rotor VTOL aircraft***". |
| [6] | Albert Einstein (1920) - "***Relativity: The Special and General Theory*** ". |
| [7] | "**Euler Angles**" - http://en.wikipedia.org/wiki/Euler\_Angles |
| [8] | *Tutorials, Magnet Lab U.*. National High Magnetic Field Laboratory. Retrieved 2008-04-30 - " ***Right and left hand rules*** ". |
| [9] | Fossen, T.I., Wiley, New York (1994) ***" Guidance and Control of Ocean Vehicles***". |
| [10] | Meirovitch, L., Wiley, New York (1990) "Dynamics ***and Control of Structures* ".** |
| [11] | Weyl, Hermann , Princeton: Princeton University Press. ISBN 0-691-02374-3. "***Symmetry*** " |
| [12] | " **Hamiltonian (control theory)"** - http://en.wikipedia.org/wiki/Hamiltonian\_(control\_theory) |
| [13] | "**Inertial Measurement Unit**" - http://en.wikipedia.org/wiki/Inertial\_Measurement\_Unit |
| [14] | Laura Salhuana , AN (2012), Frescale Semiconductor - "***Tilt Sensing using linear accelerometer***" |
| [15] | "**High-pass Filter**"- http://en.wikipedia.org/wiki/High-pass\_filter |
| [16] | "**Low-pass Filter**"- http://en.wikipedia.org/wiki/Low-pass\_filter |
| [17] | "**Kalman Filter Vs Complementary Filter**" - http://robottini.altervista.org/kalman-filter-vs-complementary-filter |
| [18] | Shane Colton, MIT (2007) - "***The Balance Filter***" |
| [19] | John A.Shaw ,Process Control Solution (2001) -"***The PID Algorithm - how it work and how to tune it****"* |
| [20] | Ziegler, J.G and Nichols, N. B. (1942). - "**Optimum settings for automatic controllers***"*. Transactions of the ASME. **64**. pp. 759–768. |
| [21] | Merryanna L.Swartz, Daniel F.Wallace, John M.Libert, Sharon Tkacz, Daniel Soloman *(1990). - "****UAV Imagery Frame Rate and Resolution Requirements Study*** " |

Table : References

# Appendix A: Variants of FSM in FUFO system



Figure : FSM of the whole system



Figure : FSM of Application on Android Phone



Figure : FSM of software on PC



Figure : FSM of Quadrocopter firmware

# Appendix B: Software Detail Design diagrams

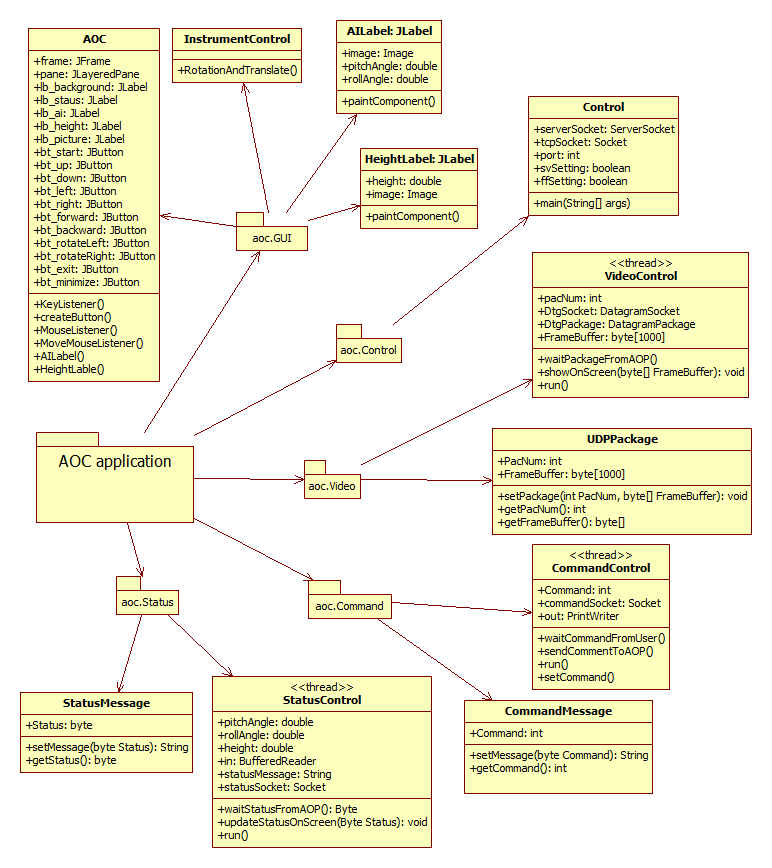


Figure : Class diagram of PC software

|  |  |  |
| --- | --- | --- |
| No | Class Name | Description |
| 01 | CommandControl | This class uses to receive pressing keyboard from users and send it to Android phone via TCP. |
| 02 | CommandMessage | This class includes “set” and “get” method to establish command message structure. |
| 03 | Control | This class is main class uses to control all of threads and GUI in application on computer. |
| 04 | AOC | This class uses to create new Graphic User Interface for Application On Computer. It also Key Listener and Mouse Listener to monitor behavior of user |
| 05 | AILabel | Uses to draw attitude indicator |
| 06 | HeightLabel | Uses to draw height indicator |
| 07 | InstrumentControl | This class includes method to draw height indicator, attitude indicator on Graphic User Interface. |
| 08 | StatusControl | This class uses to receive status from AOP and update it to screen. |
| 09 | StatusMessage | This class includes “set” and “get” method to establish status message structure. |
| 10 | VideoControl | This class uses to receive video frame from AOP and update it to screen. |
| 12 | UDPPackage | This class includes “set” and “get” method to establish UDP package structure. |

Table 8: PC software's class descriptions

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| No | Attribute | Type | Default | Note | Description |
| 01 | commandSocket | Socket | Null |  | TCP socket uses to send command to AOP |
| 02 | out | PrintWriter | Null |  | Uses OutputStream to send command to AOP |
| 03 | command | Int | 0 |  | Uses to send to AOP |
| 04 | county | Int | 0 |  | Uses to check time to send ‘y’ signal to AOP |

Table 9: Attributes of CommandControl class

|  |  |  |
| --- | --- | --- |
| No | Method | Description |
| 01 | CommandControl | Default constructor |
| 02 | CommandControl | Constructor uses to input TCP Socket |
| 03 | setCommand | Set command for CommandControl class |
| 04 | setCommandSocket | Set commandSocket for CommandControl class |
| 05 | run | This method will run when thread starts. |
| 06 | waitCommandFromUser | Uses to wait command from user ‘s input |
| 07 | sendCommandToAOP | Uses to send command to AOP |

Table 10: Methods of CommandControl class

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| No | Attribute | Type | Default | Note | Description |
| 01 | serverSocket | ServerSocket | Null | private/static | This is Server Socket on AOC |
| 02 | tcpSocket | Socket | Null | private/static | This is Socket to connect with AOP |
| 03 | PORT | int | 8888 | final/static | Port of server |
| 04 | stct | StatusControl | Null | public/static | Thread status control |
| 05 | cmct | CommandControl | Null | public/static | Thread command control |
| 06 | vdct | VideoControl | Null | public/static | Thread video control |
| 07 | svSetting | boolean | false | public/static | Flag to check socket between AOP and AOC |
| 08 | ffSetting | boolean | false | public/static | Flag to check socket between AOP and FUFO |
| 09 | start | int | 0 | public/static | Flag to check socket whether start button pressed |

Table 11: Attributes of Control class

|  |  |  |
| --- | --- | --- |
| No | Method | Description |
| 01 | main | main method of the system |

Table 12: Method of Control class

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| No | Attribute | Type | Default | Note | Description |
| 01 | frame | JFrame | null |  | Main Frame of application |
| 02 | pane | JLayeredPane | null |  | Pane contains all component of GUI |
| 03 | lb\_background | JLabel | null |  | Set image background by this label |
| 04 | lb\_circle | JLabel | null |  | Circle of attitude indicator |
| 05 | lb\_status | JLabel | null |  | This label checks connection status of system |
| 06 | lb\_picture | JLabel | null |  | Video frame streamed from AOP |
| 07 | lb\_setPoint | JLabel | null |  | Height that user wants |
| 08 | lb\_text | JLabel | null |  | Text in label status |
| 09 | lb\_ai | AILabel | null |  | Attitude indicator |
| 10 | lb\_height | HeightLabel | null |  | Height indicator |
| 12 | bt\_start | JButton | null |  | Button start |
| 13 | bt\_right | JButton | null |  | Button right |
| 14 | bt\_left | JButton | null |  | Button left |
| 15 | bt\_forward | JButton | null |  | Button forward |
| 16 | bt\_backward | JButton | null |  | Button backward |
| 17 | bt\_up | JButton | null |  | Button up |
| 18 | bt\_down | JButton | null |  | Button down |
| 19 | bt\_rotateLeft | JButton | null |  | Button rotate left |
| 20 | bt\_rotateRight | JButton | null |  | Button rotate right |
| 21 | bt\_exit | JButton | null |  | Button exit |
| 22 | bt\_minimize | JButton | null |  | Button minimize |

Table 13: Attributes of AOC class

|  |  |  |
| --- | --- | --- |
| No | Method | Description |
| 01 | AOC | constructor |
| 02 | createButton | create a button in frame base on position, path of icon button |
| 03 | mouseListener | Mouse listener |
| 04 | keyListener | Key listener |
| 05 | moveMouseListener | Mouse move listener |
| 06 | AILabel | Label for attitude indicator |
| 07 | HeightLabel | Lable for height indicator |

Table 14: Methods of AOC class

|  |  |  |
| --- | --- | --- |
| No | Method | Description |
| 01 | RotateAndTranslate | Method to rotate and translate image base on pitch angle and roll angle |

Table 15: Method of InstrumentControl class

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| No | Attribute | Type | Default | Note | Description |
| 01 | aoc | AOC | Null |  | GUI AOC |
| 02 | statusSocket | Socket | Null |  | Uses to receive status from AOP |
| 03 | in | BufferedReader | Null |  | Uses to read buffer |
| 04 | sm | StatusMessage | Null |  | Uses to analyze message |
| 05 | pitchAngle | Double | 0 |  | Pitch angle update to attitude indicator |
| 06 | rollAngle | Double | 0 |  | Roll angle update to attitude indicator |
| 07 | height | Double | 0 |  | Height update to height indicator |
| 08 | statusMessage | String | Null |  | Status message |

Table 16: Attributes of StatusControl class

|  |  |  |
| --- | --- | --- |
| No | Method | Description |
| 01 | StatusControl | Default constructor |
| 02 | StatusControl | Constructor uses to input TCP Socket and AOC |
| 03 | waitStatusFromeAOP | Uses to wait status from AOP via TCP |
| 04 | updateStatusOnScreen | Uses to update pitch, roll angle, height to screen. |
| 05 | run | This method will run when thread start. |

Table 17: Methods of StatusControl class

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| No | Attribute | Type | Default | Note | Description |
| 01 | pitchAngle | Double | 0 |  | Pitch angle update to attitude indicator |
| 02 | rollAngle | Double | 0 |  | Roll angle update to attitude indicator |
| 03 | height | Double | 0 |  | Height update to height indicator |
| 04 | statusMessage | String | Null |  | Status message |

Table 18: Attributes of StatusMessage class

|  |  |  |
| --- | --- | --- |
| No | Method | Description |
| 01 | setStatusMessge | Uses to create message from pitch angle, roll angle and height. |
| 02 | getStatus | Uses to get pitch angle, roll angle and height from message. |

Table 19: Methods of StatusMessage class

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| No | Attribute | Type | Default | Note | Description |
| 01 | aoc | AOC | Null |  | GUI AOC |
| 02 | udpSocket | DatagramSocket | Null |  | Uses to receive video from AOP |
| 03 | udpPackage | DatagramPackage | Null |  | Datagram package |
| 04 | pk | UDPpackage | Null |  | Package to receive video from AOP |
| 05 | pacBuffer | Byte[] | Null |  | Buffer uses to save package |

Table 20: Attributes of VideoControl class

|  |  |  |
| --- | --- | --- |
| No | Method | Description |
| 01 | VideoControl | Default constructor |
| 02 | VideoControl | Constructor uses to input AOC |
| 03 | waitPackageFromeAOP | Uses to wait package from AOP via UDP |
| 04 | showOnScreen | Update image to screen. |
| 05 | run | This method will run when thread start. |

Table 21: Methods of VideoControl class

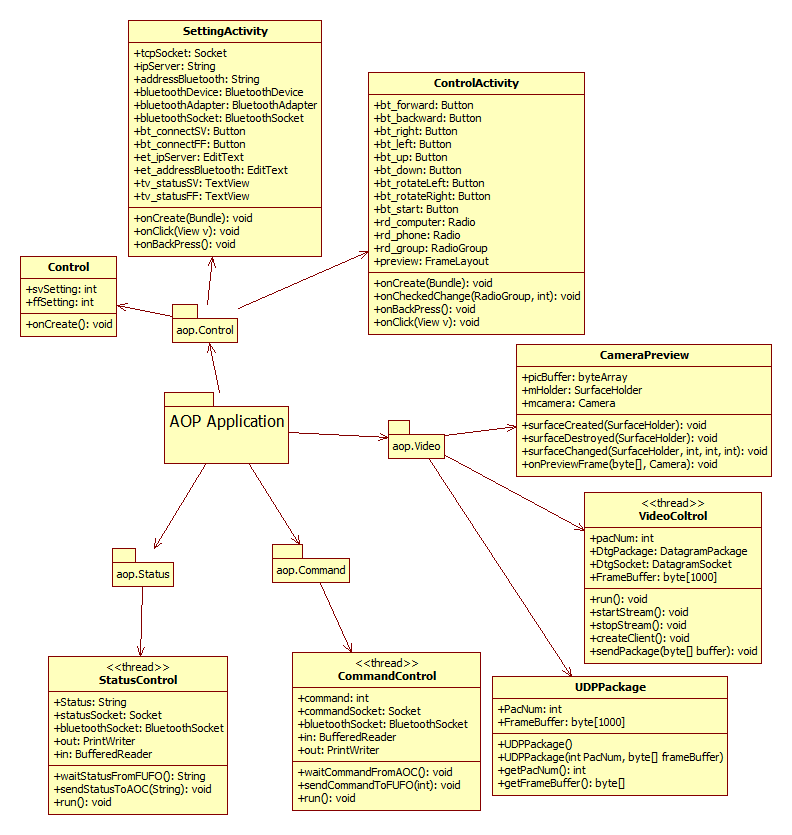


Figure 68: Class diagram of application on Phone

|  |  |  |
| --- | --- | --- |
| No | Class Name | Description |
| 01 | Control | This class is the activity executes when application starts. |
| 02 | SettingActivity | This class uses to controls operation on Setting tab |
| 03 | ControlActivity | This class uses to controls operation on Control tab |
| 04 | CameraPreview | This class uses to get video frame |
| 05 | VideoControl | This class includes methods to control video |
| 06 | StatusControl | This class includes methods to control status |
| 07 | CommandControl | This class includes methods to control command |
| 08 | UDPPackage | This class includes “set” and “get” method to establish UDP package structure. |

Table 22: Application on Phone's class descriptions

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| No | Attribute | Type | Default | Note | Description |
| 01 | svSetting | Int | 0 | static | Flag to check connection between AOC and AOP |
| 02 | ffSetting | Int | 0 | static | Flag to check connection between Firmware and AOP |

Table 23: Attributes of Control class

|  |  |  |
| --- | --- | --- |
| No | Method | Description |
| 01 | onCreate | Default method of this Activity |

Table 24: Method of Control class

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| No | Attribute | Type | Default | Note | Description |
| 01 | tcpSocket | Socket | Null |  | TCP socket for system |
| 02 | ipServer | String | Null |  | IP address of AOC |
| 03 | addressBluetooth | String | Null |  | Address of Bluetooth device |
| 04 | bluetoothDevice | BluetoothDevice | Null |  | Uses to connect firmware |
| 05 | bluetoothAdapter | BluetoothAdapter | null |  | Uses to connect firmware via Bluetooth |
| 06 | bluetoothSocket | BluetoothSocket | Null |  | Uses to send and receive data with firmware |
| 07 | bt\_connectSV | Button | Null |  | Button connect to server |
| 08 | bt\_connectFF | Button | Null |  | Uses to connect to firmware |
| 09 | et\_ipServer | EditText | Null |  | Uses to input IP server |
| 10 | et\_addressBluetooth | EditText | Null |  | Uses to input Bluetooth device address |
| 11 | tv\_statusSV | TextView | Null |  | Uses to show connection status with AOC |
| 12 | tv\_statusFF | TextView | Null |  | Uses to show connection status with Firmware |

Table 25: Attributes of SettingActivity class

|  |  |  |
| --- | --- | --- |
| No | Method | Description |
| 01 | onCreate | Default method of this activity |
| 02 | onBackPress | Method when users press “Back” button |
| 03 | onClick | Method when users click button |

Table 26: Method of SettingActivity class

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| No | Attribute | Type | Default | Note | Description |
| 01 | bt\_forward | Button | Null |  | Button forward |
| 02 | bt\_backward | Button | Null |  | Button backward |
| 03 | bt\_right | Button | Null |  | Button right |
| 04 | bt\_left | Button | Null |  | Button left |
| 05 | bt\_up | Button | null |  | Button up |
| 06 | bt\_down | Button | Null |  | Button down |
| 07 | bt\_rotateLeft | Button | Null |  | Button rotate left |
| 08 | bt\_rotateRight | Button | Null |  | Button rotate right |
| 09 | bt\_start | Button | Null |  | Button start |
| 10 | rd\_computer | Radio | Null |  | Radio button computer |
| 11 | rd\_phone | Radio | Null |  | Radio button phone |
| 12 | rd\_group | RadioGroup | Null |  | Radio group |
| 13 | preview | FrameLayout | Null |  | Frame layout uses for camera preview |

Table 27: Attributes of ControlActivity class

|  |  |  |
| --- | --- | --- |
| No | Method | Description |
| 01 | onCreate | Default method of this activity |
| 02 | onBackPress | Method when users press “Back” button |
| 03 | onClick | Method when users click button |
| 04 | onCheckChange | Method when users change radio button |

Table 28: Method of ControlActivity class

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| No | Attribute | Type | Default | Note | Description |
| 01 | picBuffer | Byte[] | Null |  | Buffer of picture |
| 02 | mHolder | SurfaceHolder | Null |  | Surface holder |
| 03 | mCamera | Camera | Null |  | Camera of phone |

Table 29: Attributes of CameraPreview class

|  |  |  |
| --- | --- | --- |
| No | Method | Description |
| 01 | CameraPreview | Constructor uses to input Context and Camera |
| 02 | surfaceCreate | Uses to create the surface |
| 03 | surfaceDestroyed | Uses to when surface destroy |
| 04 | surfaceChanged | Uses to when surface change |
| 05 | onPreviewFrame | Uses to get frame buffer |

Table 30: Method of CameraPreview class

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| No | Attribute | Type | Default | Note | Description |
| 01 | statusSocket | Socket | Null |  | TCP socket user to send status to AOC |
| 02 | bluetoothSocket | BluetoothSocket | Null |  | Uses to receive status to Firmware |
| 03 | out | PrintWriter | Null |  | Uses OutputStream to send status to AOC |
| 04 | Status | String | Null |  | Status include pitch angle, roll angle and height |
| 05 | in | BufferedReader | null |  | Uses to read buffer from Firmware |

Table 31: Attributes of StatusControl class

|  |  |  |
| --- | --- | --- |
| No | Method | Description |
| 01 | StatusControl | Default constructor |
| 02 | StatusControl | Constructor uses to input TCP Socket and Bluetooth Socket |
| 03 | Run | This method will run when thread start. |
| 04 | waitStatusFromFF | Uses to wait status from firmware |
| 05 | sendStatusToAOC | Uses to send status to AOC |

Table 32: Method of StatusControl class

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| No | Attribute | Type | Default | Note | Description |
| 01 | commandSocket | Socket | Null |  | TCP socket user to receive command from AOC |
| 02 | bluetoothSocket | BluetoothSocket | Null |  | Bluetooth Socket uses to send command to Firmware |
| 03 | out | PrintWriter | Null |  | Uses OutputStream to send command to firmware |
| 04 | command | Int | 0 |  | Command |
| 05 | in | BufferedReader | null |  | Uses to read buffer from AOC |

Table 33: Attributes of CommandControl class

|  |  |  |
| --- | --- | --- |
| No | Method | Description |
| 01 | CommandControl | Default constructor |
| 02 | CommandControl | Constructor uses to input TCP Socket and Bluetooth Socket |
| 03 | run | This method will run when thread starts. |
| 04 | waitCommandFromAOC | Uses to wait command from AOC |
| 05 | sendCommandToFUFO | Uses to send command to Firmware |

Table 34: Method of CommandControl class

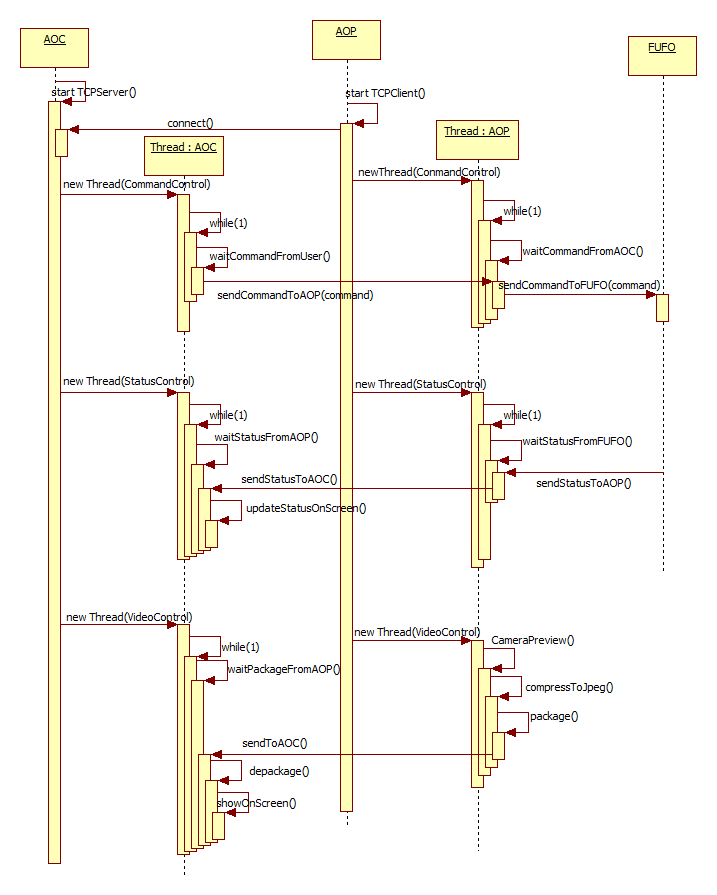


Figure : Sequence diagram of the system

# Appendix C: Firmware and Hardware recorded problems

|  |  |  |  |
| --- | --- | --- | --- |
| No. | Problem | Cause | Solution |
| 1 | Circuit has no electricity | One pin of LMS117 voltage regulator is not soldered to the circuit board. | Carefully examining the circuit board for any part that is not properly soldered and solder them. |
| 2 | No function can work on the circuit. | Crystal's capacitor positions are wrongly designed. | Remove the capacitors. |
| 3 | LCD module cannot work. | 1) LCD module cannot work in 3.3V environment.  2) No resistor installed | 1) Redesign the circuit with a Bus Transceiver 74HC245 for converting 3.3V signal to 5V signal.  2) Redesign the circuit for adding a adjustable resistor. |
| 4 | Cannot plug in ICD2 programmer. | The nearby ICs occupied too much space. | 1) Use Pickit2 instead.  2) Redesign the circuit for a bigger space between the ICs. |
| 5 | Unstable measured angular data. | The plug-in slot for sensor module is too loose. | Solder the sensor module on the circuit. |
| 6 | Sometime motors loss their power and stop. | Wires from batteries to the motors and circuit are broken inside. | Replace them. |
| 7 | ESC burned | Cannot find any cause. | More careful when connecting ESC with circuit and batteries |
| 8 | Suddenly, Quadrocopter becomes unstable after some configuration with hardware and software | Most likely because of changes in software by negligence. | Double check every time a new version of code comes out before testing with hardware. |
| 9 | Batteries are low on voltage. | Use batteries until they are out of power. | Use warning battery’s voltage level warning tools. |
| 10 | Quadrocopter flipped and caused accident during programming process. | Sometimes, when programming, something happens. It causes propeller’s unusual rotation. | Take propellers out before programming. |
| 11 | An accident had occurred while PID was being tested | Propeller’s headers are loosed. | Wring propeller’s header tightly. |
| 12 | Quadrocopter rotates out of control. | 1) Define directions of sensors wrongly.  2) PID coefficients are not reasonable. | 1) Define carefully directions of sensors regards to datasheets  2) Use test tool to guarantee safety. |

Table : Firmware and Hardware recorded problems

# Appendix D: Development process

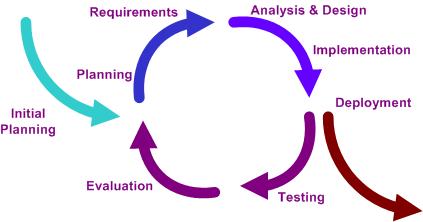


Figure : Iterative model using in FUFO project



Figure : Embedded system developing model

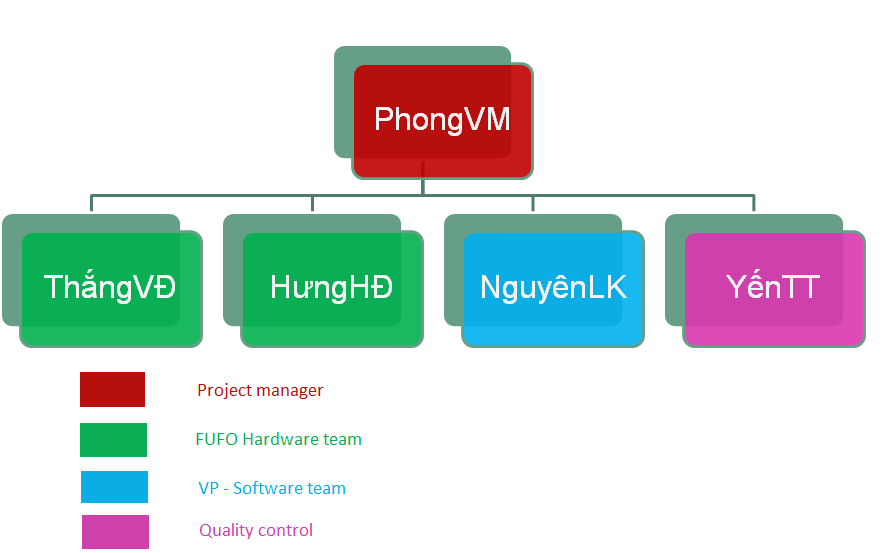


Figure : Human resource

# Appendix E: Test case and Test report