DOKUZ EYLÜL UNIVERSITY
GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES

DESIGNING OF WEB-BASED LATTICE PATH CONTROLLER FOR INDOOR ENVIRONMENT BY USING MULTIPLE QUADROTORS

by
Onur KESKİN

October, 2017
İZMİR

DESIGNING OF WEB-BASED LATTICE PATH CONTROLLER FOR INDOOR ENVIRONMENT BY USING MULTIPLE QUADROTORS

A Thesis Submitted to the
Graduate School of Natural and Applied Sciences of Dokuz Eylül University In Partial Fulfillment of the Requirements for the Degree of Doctor of Philosophy in Mechatronics Engineering

by
Onur KESKİN

October, 2017
İZMİR

Ph.D. THESIS EXAMINATION RESULT FORM

We have read the thesis entitled **"DESIGNING OF WEB-BASED LATTICE PATH CONTROLLER FOR INDOOR ENVIRONMENT BY USING MULTIPLE QUADROTORS"** completed by **ONUR KESKİN** under supervision of **PROF. DR. ZEKİ KIRAL** and we certify that in our opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Doctor of Philosophy.

Prof. Dr. Zeki KIRAL

Supervisor

Prof. Dr. Mehmet KUNTALP

Thesis Committee Member

Doç. Dr. Levent ÇETİN

Thesis Committee Member

Prof. Dr. İlker Murat Koç

Examining Committee Member

Doç. Dr. Ayşegül Alaybeyoğlu Soy

Examining Committee Member

Prof.Dr. Kadriye ERTEKİN
Director
Graduate School of Natural and Applied Sciences

ACKNOWLEDGMENTS

I would like to thank to my advisor Prof. Dr. Zeki KIRAL because of suggestions, helps, supports and patience for the research and writing of this thesis. I also want to thank to Prof. Dr. Mehmet KUNTALP and Assoc. Prof. Dr. Levent ÇETİN because of their help and encouragements. And also, my colleague Mr. Uğur ŞAYLAN for his repetitive encouragements and moral support.

Finally, thanks for her helps, patience and love my wife Fulya and my son Poyraz, and for all supports my father and mother.

This thesis study is a part of the 2012.KB.FEN.116 project that is supported by Dokuz Eylül University Scientific Research Projects Coordination Unit.

Onur KESKİN

DESIGNING OF WEB-BASED LATTICE PATH CONTROLLER FOR INDOOR ENVIRONMENT BY USING MULTIPLE QUADROTORS

ABSTRACT

Unmanned aerial vehicles (UAVs) become very popular in last years by the help of increasing computing power per area and per cost. While UAVs with global positioning system (GPS) can easily operate to flight autonomously, this and such sensors data cannot be always trusted. And most of the cases for small and nano scale UAVs, we cannot use these kinds of sensors because of cost, complexity and weight. Safely and reliably operating close to unknown indoor or GPS-denied environments requires improving UAVs' sensing, localization, and control algorithms. To solve and improving for a UAV is one problem, extending it to a multiple UAVs is another problem. We study and develop a framework for multiple UAVs to command and control from web-based front-end. In our experiments, a quadcopter platform called AR Drone from Parrot Inc. is preferred and used because of its open-source API and kernel level arrangements. Test flights inside a warehouse validated that the framework is capable of control one or more quadcopters in given different lattice-based path from a web browser. UAVs can localize its position with using IMU, front and bottom camera. All UAVs are interconnected to each other and controller computer through a self-healing network, so if one or more quadcopter fails because of lack of battery or any other circumstances, the rest of the group continues its mission.

Keywords: Unmanned aerial vehicle, quadrotor, multi robot system, lattice-based path, self-healing network, web-based control

İÇ MEKANLAR İÇİN WEB TABANLI KAFES YOLU KONTROLCÜSÜNÜN ÇOKLU DÖRT ROTORLULAR KULLANILARAK TASARLANMASI

ÖZ

İnsansız hava araçları (İHA'lar) birim işlem gücünün de ucuzlaması ile oldukça popüler bir çalışma alanı haline gelmiştir. Küresel konumlama sistemleri (KKS) ile İHA'ların otonom olarak kontrol edilmesi her ne kadar kolay olsa da bu tarz algılayıcıların verilerine her zaman güvenmek mümkün olmamaktadır. Ayrıca bu algılayıcının boyut, maliyet ve karmaşıklığından dolayı ufak boyuttaki İHA'lara uygulamak mümkün olmamaktadır. İç mekanlarda, KKS'lerin kullanılamadığı alanlarda güvenli ve güvenilir şekilde çalışabilmek için İHA'ların algılama, konumlandırma ve kontrol algoritmalarında iyileştirmeler yapılması gerekmektedir. Bu çözümü tek bir İHA için uygulamak bir problem, bulunacak çözümü bir grup şeklinde görev alacak İHA'lara uygulamak ise bir başka problemdir. Bu çalışma sonucu geliştirilen yazılım çerçevesinde birden fazla İHA ya web tabanlı bir arayüz üzerinden komuta edilmesi sağlanmıştır. Çalışmalarımızda Parrot Inc. firmasının AR Drone adlı dört rotorlu platformundan yararlanılmıştır. Bu platform, açık kaynaklı yazılım geliştirme kitine sahiptir ve çekirdek seviyesinde değişikliklere izin verir. Çalışmanın deney kısmı depo gibi kapalı alanlarda bir ve birden fazla dört rotorlunun kontrol edilebildiğini göstermek üzere gerçekleştirilmiştir. Deneylerde görevleri barındıran ızgara tabanlı yollar, web tarayıcı üzerinden oluşturulup paylaşılmıştır. İHA'lar konumlandırma için ataletsel ölçüm algılayıcısına, ön ve alt kameraya sahiptirler. Tüm platformlar ve kontrol bilgisayarı kendini iyileştirebilen bir ağ mimarisine sahip kablosuz ağına bağlılardır. Bu ağ mimarisi özellikle platformlardan birinde yaşanabilecek sorunlu durumda bile grubun görevine devam etmesi için kurulmuştur.

Anahtar kelimeler: İnsansız hava aracı, dört rotorlu, çoklu robot sistemi, ızgara tabanlı yol, kendini iyileştiren ağ, web tabanlı kontrol

CONTENTS

Page

LIST OF FIGURES

Page

LIST OF TABLES

Page

CHAPTER ONE
INTRODUCTION

The focus of this study is to develop control algorithms and perform stability that will provide a certain level of autonomous flight control to an unmanned aerial vehicle (UAV). Flying is always a challenge to humankind. In past centuries this problem is well studied, and various flying vehicles are invented and developed. But in recent years, challenge is to develop unmanned aerial vehicle. There has been rapid development of autonomous unmanned aircraft equipped with autonomous control devices called unmanned aerial vehicles (UAVs). In general UAV can be piloted remotely and/or autonomously. Autonomous flight missions are generally executed by predefined flight task. Although UAV can be remotely piloted, and a certain level of control is needed.

UAV is especially used in potentially dangerous flight missions and/or where small size vehicles are needed to complete specific missions. They can be classified according to their application for military or civil uses. They have very wide application areas such as; inspection of pipelines and power lines, surveillance, rescue missions, border patrol, oil and natural gas research, fire prevention, fire prevention, topography applications, agricultural applications, filmmaking, traffic monitoring, and flight in hazardous environments (chemical, radioactive, mine land etc.).

1.1 Importance of the Study

Flying is one of the biggest passions of humankind since middle ages. Pilot operated flights have become a standard transportation in modern ages. Flying have lots of advantages in both civilian and military point of view. Such as short distance time from A to B, great opportunity for information, surveillance, target acquisition, and reconnaissance (ISTAR). On the other hand, the main disadvantage of pilot operated flight is possibility of damaging or terminating human life. In order to solve this problem remote operated control is studied by various researchers. Nicole Tesla unveiled his own remote operated boat in 1898 (Figure 1.1). With enhancing technology today remotely operated also known as tele-operated “Unmanned Aerial Vehicle” (UAV) has become a key component of both civilian and military sector.

Figure 1.1 Tesla created a remote-control boat to exhibit at Madison Square Garden back in 1898 (Johnston, 2013)

Unmanned Aerial Vehicle plays a vital role that is important to achieve dangerous, timely and cost-effective mission execution. In standard tele-operated UAV, pilot or audience can get visual feedback with lots of flight diagnostics. Visual feedback can vary with different payload such as thermal or NIR (Near Infrared) camera. And with the capability of UAV, additional payloads can be carried such as cargo box, environmental sensors, missiles etc. Although tele-operated UAVs prove their importance, a pilot with certain experience always has to be in the range in order to operate them whether it is a flying camera for a new movie or an aid mission for troops. Because of this, trained human factor, developing autonomous, self-driving aerial vehicle is the essential goal of the unmanned aerial vehicle ecosystem.

Autonomous flight is a real-time complex, battery, communication and load capacity constrained system and takes place in an unstable environment. At this point, main effort of this thesis is to construct a stable and scalable autonomous group flight framework for unmanned aerial vehicles. This thesis fills a gap by focusing on implementation of a self-healing communication network between a group of UAVs driven by asynchronous lightweight control framework.

1.2 Research Objectives

The main goal of this thesis is to introduce series of procedures for flight control of multiple quadrotor type UAV under scalable, secure and resistant constraints and developing a solution platform. In the UAV control literature, multiple flight is a relatively new topic. To manage multiple UAVs and fly stable to execute separate and/or joint missions, asynchronous events, resistant to connection fluctuations is needed. This thesis presents one of the most user-friendly web-based asynchronous multiple flight control platforms with lattice-based grids and self-healing network structure in the background.

Multi agent robot control is a multi-dimensional, multi-threated problem. Because of this, asynchronous event driven architecture approach is proposed to tackle this problem. The proposed constructive and event-based platform is compared with sequential methods that are usually utilized for non-blocking procedures so application does not wait to complete previous task and can manage different task simultaneously with also using feedback from other tasks.

Aim of this thesis study is to build a web-based controller platform for mission assignment using lattice-based grid map to multiple quadrotors in order to execute single, discrete or cooperated flights.

1.3 Outline of the Thesis

Rest of the thesis involves seven chapters. The following chapter contains the theoretical background, various unmanned aerial vehicle designs, kinematics and motion control of four rotor UAV, a brief introduction to multi agent robotics for formation control of multiple UAVs.

3rd chapter presents a ready to flight quadrotor platform with its mechanical design verification studies and software developer kit's structure. Particularly, customization for embedded operating system in order to build low resource consuming computer vision algorithms and networking.

In chapter 4, the software architecture and its components are explained which is developed under this thesis study.

Chapter 5 is devoted to explaining the communication between UAVs itself and the client interface which have a self-healing behavior.

Experiments with different task assignments and their results are discussed in chapter 6.

In the last chapter, chapter 7, conclusions and recommended future works are expressed.

CHAPTER TWO
THEORTICAL BACKGROUND

2.1 Unmanned Aerial Vehicle (UAV) Designs

Unmanned aerial vehicles can also classify according to their flying characteristics; fixed wing, rotorcraft (rotary wing), blimps and flapping wing. Fixed wing UAV can be described as airplanes. This type of UAV needs a runway to takeoff and landing or in some cases a catapult to give necessary initial relative velocity. They have long durability and cruise in high speeds. Because of these properties they are generally used for military applications.

Rotorcraft UAV is also called vertically takeoff and landing (VTOL) or rotary wing unmanned aerial vehicle. Their main advantage is hovering and high maneuverability. These properties are the key features for many robotic missions and civilian applications. They have also different rotor configurations such as single rotor (conventional helicopters), axial, coaxial rotor (Sikorsky X2), tandem (Chinook), tilt (V-22), intermeshing (Kaman K-MAX) and multi-rotor (quadrotor) etc. (Watkinson, 2004). The comparison of rotorcraft and fixed wing aerial vehicles are shown in Table 2.1. The scores in the table represent our opinions, which were concluded according to the related studies in the literature and our prior experience in the field. The purpose in showing our opinions as scores is to make it clear how we decided to use a quadrotor design for our research.

Table 2.1 Comparison table of aerial vehicles (Higher is better)

Property / Vehicle	Fixed Wing	Single	Axial	Co-Axial	Tandem	Tilt	Intermeshing	Multi
Flight Time (Power usage)	4	3	5	2	2	2	2	1
Controllability	4	1	1	3	2	4	4	5
Payload	5	3	1	4	4	4	5	3
Maneuverability	1	4	1	2	3	3	2	3
Mechanic Simplicity	4	1	5	3	2	1	2	5
Aerodynamic Simplicity	2	1	1	1	1	2	1	3
Flight Speed	5	4	1	3	2	5	3	4
Miniaturization	5	4	5	4	3	1	2	4
Survivability	4	1	1	3	2	5	2	3
Stationary flight	0	3	2	4	3	3	3	5
TOTAL	**34**	**25**	**23**	**29**	**24**	**30**	**26**	**36**

Fixed wing aerial vehicles have high cargo (payload) capacity with high air speed and they can be constructed in smaller size but with high aerodynamically stable construction. Despite all these properties, this design is lack of stationary flight. And it also has the lowest maneuver ability; fixed wings cannot simply turn both sides without drawing an arc in the air.

Single rotorcraft vehicle also known as helicopter has a proven design with high maneuver capacity. But it has very complex mechanical and aerodynamic design. Control of helicopter is also very complicated. All the flight system equipment's are critical during flight and only redundant structure can be the main motor.

Axial rotors are very simple and simply constructed with less power usage. But they are not for carrying load, surveillance and like other missions.

Co-axial design is an improved design of helicopter. By using two contra-rotating propellers, this aerial vehicle does not need any tail rotor. So controllability and survivability properties are higher than conventional helicopter. Tandem rotor design is similar like co-axial rotorcrafts in many aspects.

Tilt rotor aerial vehicles are relatively a new concept. They are the combination of airplane and tandem rotorcraft. They are higher survival rate than all aerial vehicle designs. Flight speed can be high as fixed wings.

Intermeshing rotorcraft design, also known as synchropter, is a complex design of co-axial design. Two propellers have a certain tilt angle and with this design concept tail rotor is unneeded. This design advantage is to carry about 2-3 tons of payloads.

Multi rotor design has advantages of simpler controller, mechanic and aerodynamic design. With the help of the number of rotors, stability is high. But the main disadvantage of this design is lower flight time with high power consumption.

Multi rotor segment is rapidly evolving and dynamic area of study for academic, governmental and military research groups. As we see from Table 2.1, they are better suited for many civil applications than other types of UAVs. Most the applications such as; law enforcement, crisis and disaster management, infrastructure inspection, agriculture require low altitude flights with vertically takeoff and landing (VTOL) features. Multi rotor UAVs can be categorized into five classes based on some attributes such as payload capacity, size and range.

- 1st category: Full-scale (un)manned aerial vehicle or optionally piloted autonomous helicopters. General advantages are massive payload capacity, flexible, safety for test pilots etc.
- 2nd category: Medium scale unmanned aerial vehicles that can be piloted both autonomously and teleoperated. Vehicle's total weight is more than 30kg. And payload capacity is generally more than 10kg. Main advantage is to carry high quality and heavy mission-oriented sensors.
- 3rd category: Small-scale aerial vehicles are based on remotely controlled helicopters with optionally integrated autopilot systems. They can carry

payloads between 2 to 10 kg. And their total weight is less than 30kg. They are usually used for low-cost alternatives to 2nd category.

- 4th category: Mini aerial vehicles that can carried easily and suitable for both outdoor and indoor missions. Most of them is powered by DC electric voltage or some hybrid solutions also exist. Because of battery limitations they have a net flight time in a scale of minutes. Their weight is generally between 100g to 2kg. They can turn their disadvantage into various kind of advantages such as high-tech sensors, low cost, expendable, easy to maintenance, safe to operate in public.
- 5th category: Micro aerial vehicles with less than 100g are designed using bio-inspired principles. They can only have limited sensors and batteries but easily operate indoor environments for rapid information collection, novel sensing and navigation research.

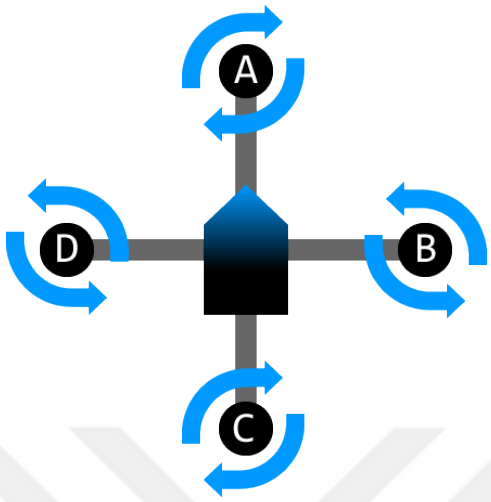
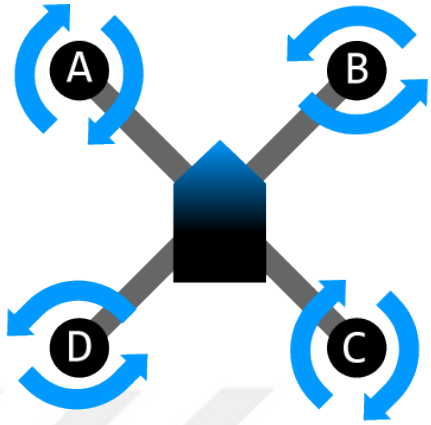
In this thesis, fourth category UAV is studied and in that manner software platform is developed.

2.2 Four rotor UAV Designs

General principle of quadrotor control is differential thrust and torque that was developed by paired counter rotation motors. It can have six degrees of freedom (DoF) with varying total thrust. Quadrotor has a unique and simple design. It has fewer moving parts than conventional helicopters. With this property quadrotor is much more robust than another rotorcraft aerial vehicle (Anderson, 2010).

Quadrotor has generally two different motor pair configurations as plus (+) or cross (x) configuration as seen in Table 2.2. These designs are basically same. The only need is to choose one of them is generally about electronic control unit or sensor array placement. For example, if quadrotor has a front camera the cross (x) configuration is more suitable.

Table 2.2 Motor pair configuration of Quadrotor

Plus (+) Configuration	Cross (x) Configuration
	

Each rotor in quadrotor generates a flow field which can be treated as an actuator disc with the fixed pitch propeller (Figure 2.1). This motor-propeller couple transfers power from drive shaft to the fluid medium which is air in our case. For simplification, no attention was paid to the details of propeller's blade number and shape.

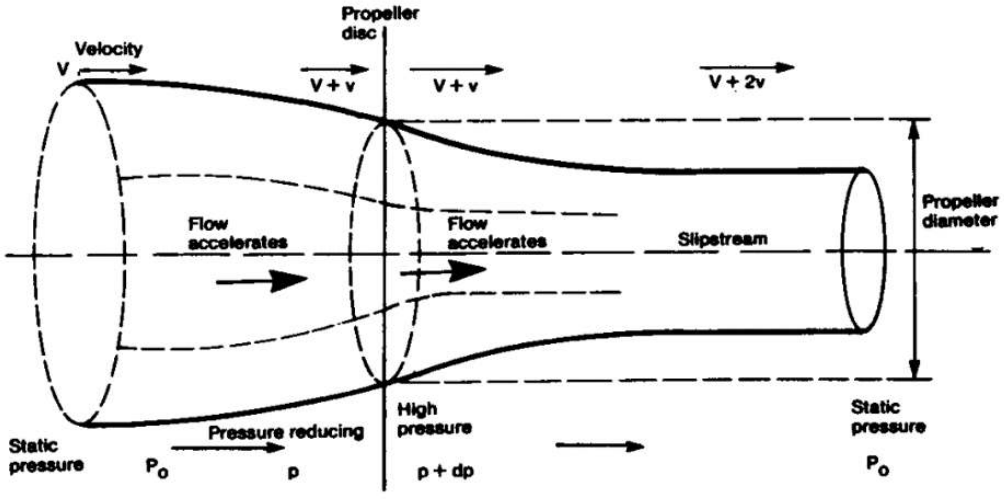


Figure 2.1 The propeller as an actuator disc in a flow field (Simons, 1994)

Thrust can be described as a reaction force by Newton's second and third laws. System accelerates mass in one direction; the mass which is accelerated will cause a

force of equal size with opposite direction to the system. In our case, thrust is the difference of the momentum flux out to the momentum flux. Required power to generate needed thrust and the force as a result of this thrust is related in non-linear form manner but in our case, we can solve this relation by assuming flow is uniform.

$$Thrust\ (T) = \rho\pi r^2 v^2 \quad \text{When } V_0 \text{ is zero} \tag{2.1}$$

$$Power\ (P) = F \times v \tag{2.2}$$

$$Force\ (F) = Area \times Dynamic\ Pressure \tag{2.3}$$

$$Dynamic\ Pressure = \frac{1}{2}\rho v^2 \tag{2.4}$$

$$Power\ (P) = \frac{1}{2}\rho\pi r^2 v^3 \text{ (From Eq. 2.2, 2.3, 2.4)} \tag{2.5}$$

$$P^2 = \frac{T^3}{4\rho\pi r^2} \text{ (From Eq. 2.1 and 2.5)} \tag{2.6}$$

Interpretation of Equation 6 is that the power is directly proportional with thrust and inversely proportional with fluid density (ρ) and actuator disc area. For example, ship's propeller can be designed in a smaller size in order to generate thrust because of the reason of water's density is very much higher than air. But in the other hand, a helicopter's propeller cross section area must be enough big in order to generate thrust for lifting and maneuver.

The theoretical maximum efficiency of any design of a propeller system (helicopter, multirotor, wind turbine etc.) is less than 59% according to Betz' Law (or Limit) (Betz, 1966 and Ragheb, 2010). By this law, no propeller can covert more than 59% of the mechanic energy of the rotor into kinetic energy for lifting the system. This limit is much lower in practical situations such as between 30% and 40%.

Betz' Law (or Limit) can be extracted from Equations given above:

$$Power\ (P) = \dot{E} \tag{2.7}$$

$$\dot{E} = \frac{1}{2}\dot{m}(v_1^2 - v_2^2) \tag{2.8}$$

$$\dot{E} = \frac{1}{2}\rho A v(v_1^2 - v_2^2) \tag{2.9}$$

$$v = \frac{1}{2}(v_1 + v_2) \tag{2.10}$$

$$\dot{E} = \frac{1}{4}\rho A(v_1 + v_2)(v_1^2 - v_2^2) \tag{2.11}$$

$$\dot{E} = \frac{1}{4}\rho A v_1^3 (1 - \left(\frac{v_2}{v_1}\right)^2 + \left(\frac{v_2}{v_1}\right) - \left(\frac{v_2}{v_1}\right)^3) \tag{2.12}$$

From Equation 2.12, performance coefficient graph given in Figure 2.2 can be plotted in order to find maximum v_2 / v_1 value. The maximum ratio is calculated as 1/3.

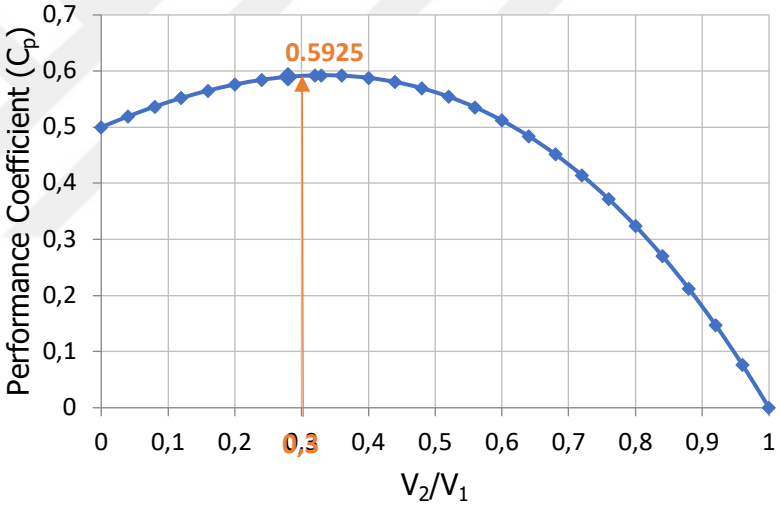


Figure 2.2 Coefficient of Performance via Betz's Law

Power Equation with respect to performance coefficient (C_p) can be derived by using v_2 / v_1 ratio in Equation 2.12. Maximum C_p is found as 16/27 = 0.59259 as used in Equation 2.13.

$$P_{max} = \frac{1}{2}\rho A v^3 \left(\frac{16}{27}\right)$$

$$P = \frac{1}{2}\rho A v^3 C_p \tag{2.13}$$

In order to calculate the thrust needed for quadrotor flight by using motor and propeller configuration by using Equations 2.13, a special calculator program called "Thrust-Power Calculator for Quadrotors – TPCalc" is written via C# programming language under .NET Framework as seen in Figure 2.3. Program takes voltage and current of DC motor, propeller's dimension as inch, air density for specific temperature and coefficient of performance as an input and generates power, thrust and weight that the motor-propeller couple can lift. If quadrotor construction is known, fly weight could be given to the calculator in order to calculate lift and cargo capacity, payload and total thrust.

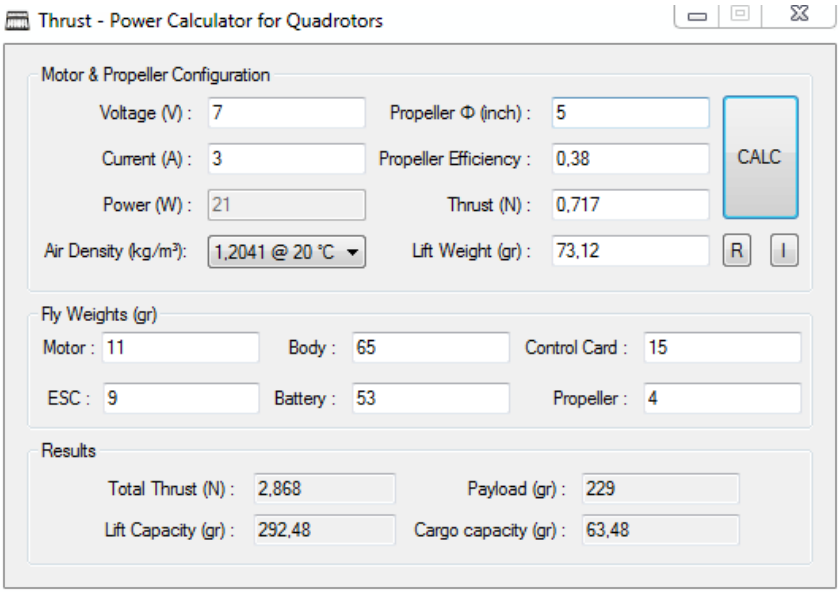


Figure 2.3 Thrust-Power Calculator for Quadrotors screenshot

The aim of our first design is to create a small size unmanned aerial vehicle with multi rotors with general components shown in Figure 2.4. Our quadrotor platform was obtained from related places and then their all-important parts are designed under SolidWorks in order to simulate by various computer program in the future of this thesis. The electronics part is constructed for tele-operated control at the beginning in order to make observations.

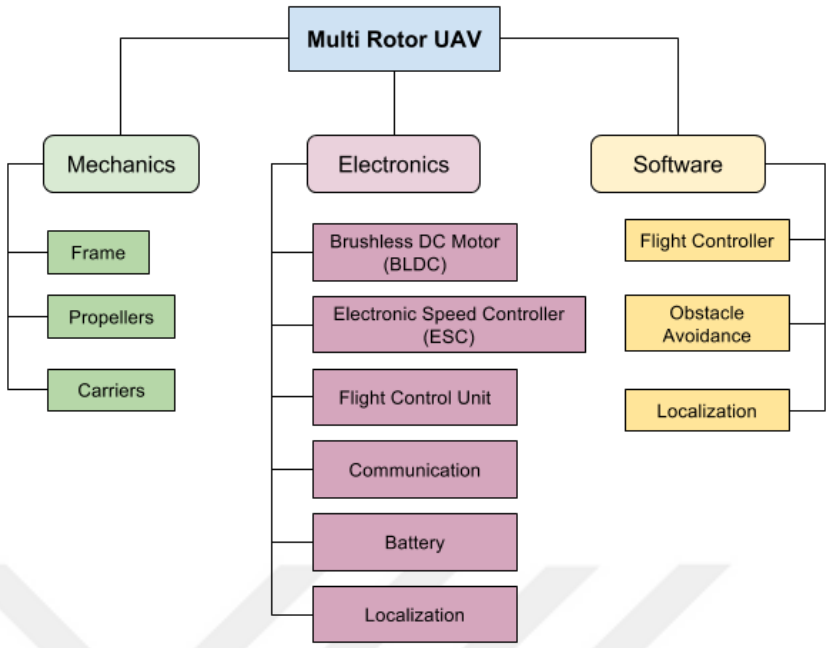


Figure 2.4 General components of a multi rotor UAV components

Quadrotor's most important part is its frame. It must be light but resistant and also provide main structure for all other parts. Our frame is constructed from fiberglass and FR-4 material (Figure 2.5). With the help of this material it has also a PCB layer for power distribution. Power distribution is used for connecting battery to whole system for example each motor through their ESC (Electronic Speed Controller), flight control unit and also communication system.

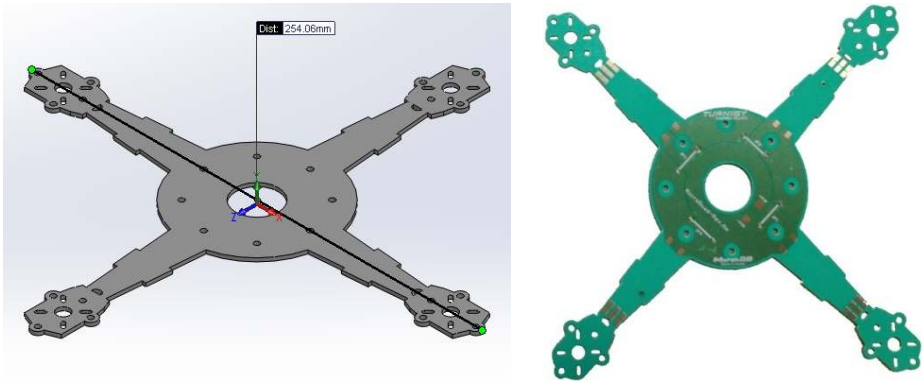


Figure 2.5 Frame's base structure design and real images (Personal archive, 2017)

Thrust is the key of multi rotor flight, and to generate this thrust, propellers are needed in Figure 2.6. 5" x 3" propellers are designed with real values in Solidworks 3D CAD program in order to use further simulation programs for flow analysis.

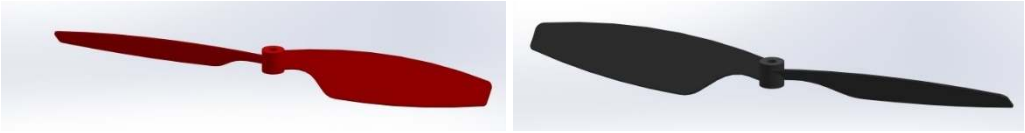


Figure 2.6 Propellers; counterclockwise pitch angle (left), clockwise pitch angle (right)

Our main electronic components are brushless DC motor, electronic speed controller, flight control unit, communication unit and battery.

Brushless DC motors are preferred in multi rotor design because of their advantages in efficiency, power, weight and durability rather than traditional brushed DC motors. Our out-runner motor is selected because of its size and power capacity (Figure 2.7). In an out-runner motor, which is the permanent magnets are outside and stationary coil windings are inside. Out runner motor produces more momentum with the help of the spinning outer case. They generally have 3 phases. It has maximum power output of 49 watts with 3800 RPM per volt value. Lift weight (capacity - Thrust / gravity) by using 5" x 3" propeller with 0.38 C_p is 129 grams.

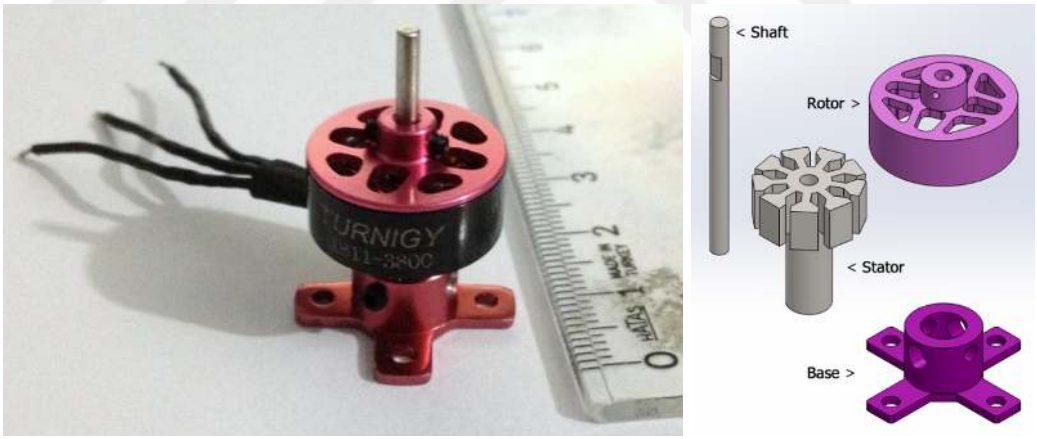


Figure 2.7 Turnigy 1811 brushless DC motor and exploded 3D drawings (Personal archive, 2017)

Electronic speed controller (ESC) is used for brushless DC motors both in runner and out runner. ESC package has N-channel MOSFET and Atmel microcontroller. Microcontroller brings ESC the ability to programming. ESCs have low internal resistance. They also have protection for overheat and overload. And for protection, auto shutdown can be used. This property turns off ESC when PWM (input signal) is lost for a while. More advance ESCs also have battery eliminating circuit (BEC) in

order to provide +5V to another circuit for example control board, receiver, etc. in Figure 2.8. Some of the programming properties of ESC; timing which increases the RPM under acceleration but effects on motor temperature and battery life, brake which can load a low resistance across the motor terminals which cause a dynamic brake, governor which adjusts the motor RPM to a given speed (heading/forward speed) and soft start which adjusts starting voltage by increasing by time, for example 1 second ramp-up from start to full RPM. If gearbox or folding propeller is used this setting must be used.

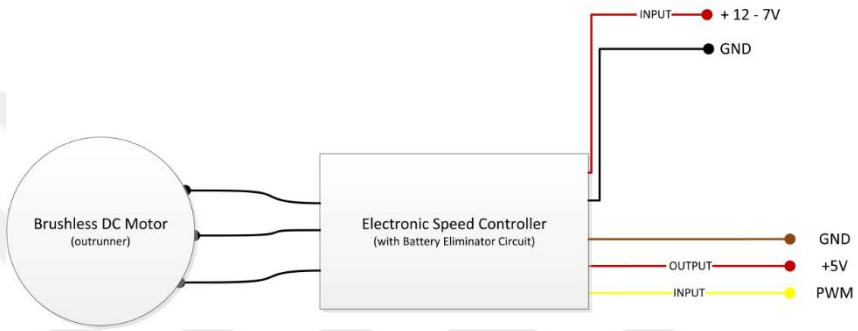


Figure 2.8 Schematic connection of a Programmable ESC with BEC support

The ESCs which are selected for our project has both programming and BEC support with linear throttle curve and can handle 10 amperes continuously. BEC output is 5 volts with up to 2 amperes (Figure 2.9).

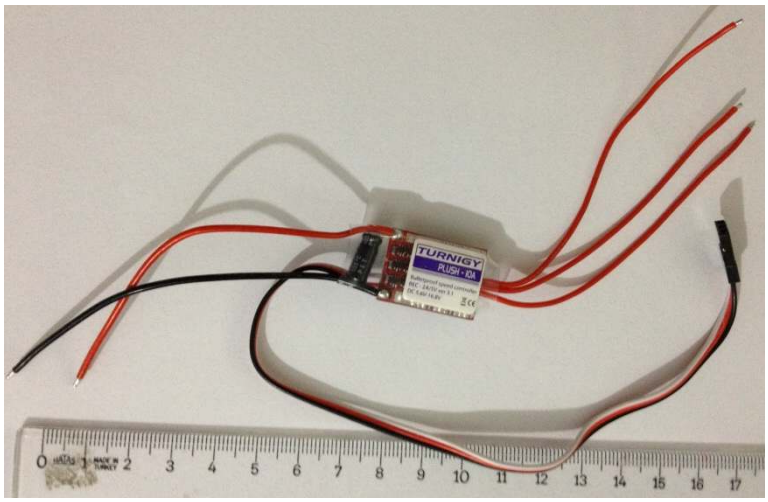


Figure 2.9 Turnigy Plush 10A Programmable ESC with BEC (Personal archive, 2017)

Flight control unit is the main unit of all system. It has Atmel Atmega168 or 328 microcontrollers with 8-bit CPU and 20MHz operating frequency. Atmega328 microcontroller provides more flash and EEPROM memory and RAM. Sensor layer has three single axis Murata ENC-03 MEMS (Micro Electromechanical System) gyroscope. This gyroscope has 50Hz response frequency and +/- 300deg/sec maximum angular velocity. On the board, there are six ESC ready motor output pins and four controller input pins for throttle, aileron, elevator and rudder signals. Board has in-system programmer header port and it is compatible with Atmel Studio IDE for programming and debugging. All this connection pins are shown in Figure 2.10.

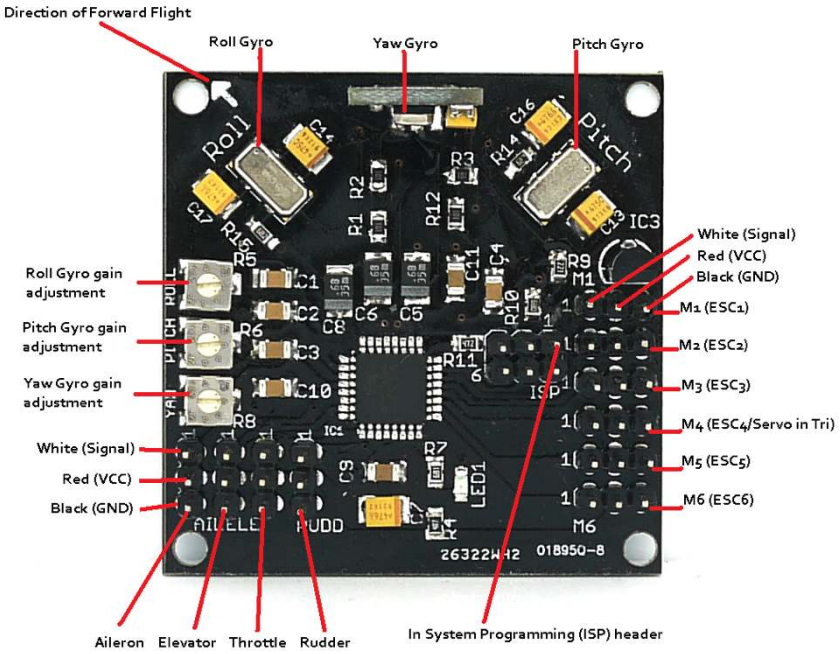


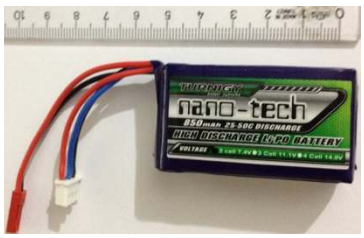
Figure 2.10 Flight control unit pins (Bakke, 2015)

For tele-operated control 9 channel 2.4 GHz radio frequency receiver and transmitter pair with a range of 1-1.5 km is used in Figure 2.11. Transmitter can also be configured for PC interface.

Figure 2.11 Transmitter and receiver pair (Personal archive, 2017)

Battery is selected with respect to ESC and motor needs shown in Table 2.3. Li-Po type batteries are preferred because of their advantages. Main advantages of Li-Po battery are light weight, high discharge rate, low internal resistance, high charge efficiency, short charge time and long lives cycles.

Table 2.3 Selected Li-Po battery properties (Personal archive, 2017)

		
Capacity [mAh]	1000	850
Nominal Voltage [V]	7.4	
Discharge [C]	15	25
Dimensions [mm]	71x35x11	60x31x16
Weight [gr]	53	47
Cost [$]	5	6

Flight time can be calculated from Equation 2.14 with battery properties and current need. In our case motor current is the total current of all four motors.

$$Flight\ Time\ [min] = \frac{Battery\ Capacity\ [Ah]}{n_{rotor} \times Motor\ Current\ [A]} \times 60\ ^{min}/_{h} \qquad (2.14)$$

For our quadrotor design and with using 1000 mAh battery at 3 amperes current usage for single motor; battery will be empty after 5 minutes theoretically. Motor current usage can be estimated from total weight of the UAV, propeller diameter, maneuverability factor for effective braking and recovery and of course from Equation 2.6.

2.3 Kinematics of Quadrotor

In order to model the quadrotor, some assumptions are needed because of its nonlinear factors. Such as; rigid and symmetric body, rigid rotors without blade flapping, negligible gravity difference with respect to altitude and Earth's spin and coincide body fixed frame and center of gravity. Movement of the quadrotor in world coordinates x, y and z axis is based on Newton's 2nd law. Following Equations are the simplified results of these movements as shown in Figure 2.12.

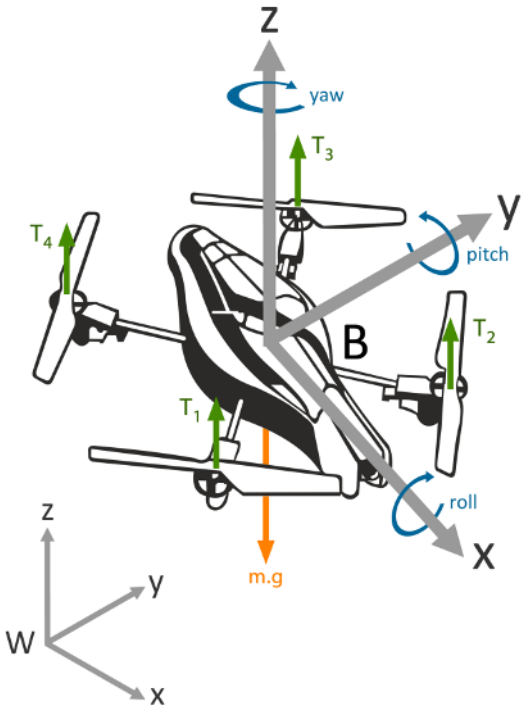


Figure 2.12 Coordinate systems of a quadrotor; W – world, B – body

$$\vec{F} = -mg_z + \sum_{k=1}^{4} \overrightarrow{T_k} \tag{2.15}$$

$$\vec{F} = m\vec{a} = m\begin{bmatrix}\ddot{x}\\ \ddot{y}\\ \ddot{z}\end{bmatrix} = {}^{W}R_B \cdot T + \begin{bmatrix}0\\ 0\\ -mg\end{bmatrix} \tag{2.16}$$

$$T = \left|\sum_{k=1}^{4} \overrightarrow{T_k}\right| \tag{2.17}$$

$${}^{W}R_B = R_z R_y R_x \tag{2.18}$$

$$R_x = \begin{bmatrix}1 & 0 & 0\\ 0 & c\phi & -s\phi\\ 0 & s\phi & c\phi\end{bmatrix}, R_y = \begin{bmatrix}c\theta & 0 & s\theta\\ 0 & 1 & 0\\ -s\theta & 0 & c\theta\end{bmatrix}, R_z = \begin{bmatrix}c\psi & -s\psi & 0\\ s\psi & c\psi & 0\\ 0 & 0 & 1\end{bmatrix} \tag{2.19}$$

$${}^{W}R_B = \begin{bmatrix}c\psi c\theta & c\phi s\psi + c\psi s\phi s\theta & s\phi s\psi - c\phi c\psi s\theta\\ -c\theta s\psi & c\phi c\psi - s\phi s\psi s\theta & c\psi s\phi + c\phi s\psi s\theta\\ s\theta & -c\theta s\phi & c\phi c\theta\end{bmatrix} \tag{2.20}$$

$$F = \begin{bmatrix}T \cdot (s\phi s\psi - c\phi c\psi s\theta)\\ T \cdot (c\psi s\phi + c\phi s\psi s\theta)\\ T \cdot (c\phi c\theta) - mg\end{bmatrix} \tag{2.21}$$

$(Where\ \phi\ is\ roll,\ \theta\ is\ pitch\ and\ \psi\ is\ yaw\ angle)$

Following Equations are for rotation maneuvers in world coordinate systems. Gyroscopic effects on quadrotor can be neglected due to the small angular velocities in Equation 2.23. Moments acting on the quadrotor can be expressed in matrix form as given in Equation 2.24 where K_l is lift coefficient and ℓ is the radius of quadrotor as shown in Figure 2.13.

$$\vec{M} = I \cdot \dot{\vec{\omega}} + \vec{\omega} \times I \cdot \vec{\omega} \tag{2.22}$$

$$\vec{M} \approx I\dot{\vec{\omega}} \tag{2.23}$$

$$\vec{M} = \begin{bmatrix}I_{xx} & \ddot{\phi}\\ I_{yy} & \ddot{\theta}\\ I_{zz} & \ddot{\psi}\end{bmatrix} = \begin{bmatrix}l & -l & -l & l\\ -l & -l & l & l\\ K_l & -K_l & K_l & -K_l\end{bmatrix}\begin{bmatrix}T_1\\ T_2\\ T_3\\ T_4\end{bmatrix} \tag{2.24}$$

$$I = \sum_{i=1}^{4} m_i {r_i}^2 + \iiint_V \rho(\vec{r}) d(\vec{r})^2 dV \tag{2.25}$$

$$I_{xx} = 4{m_m}{l_1}^2 + \frac{\rho V}{12}(l_3^2 + l_4^2)$$

$$I_{yy} = 4{m_m}{l_1}^2 + \frac{\rho V}{12}(l_2^2 + l_4^2) \tag{2.26}$$

$$I_{zz} = 4{m_m}{l_1}^2 + \frac{\rho V}{12}(l_2^2 + l_3^2)$$

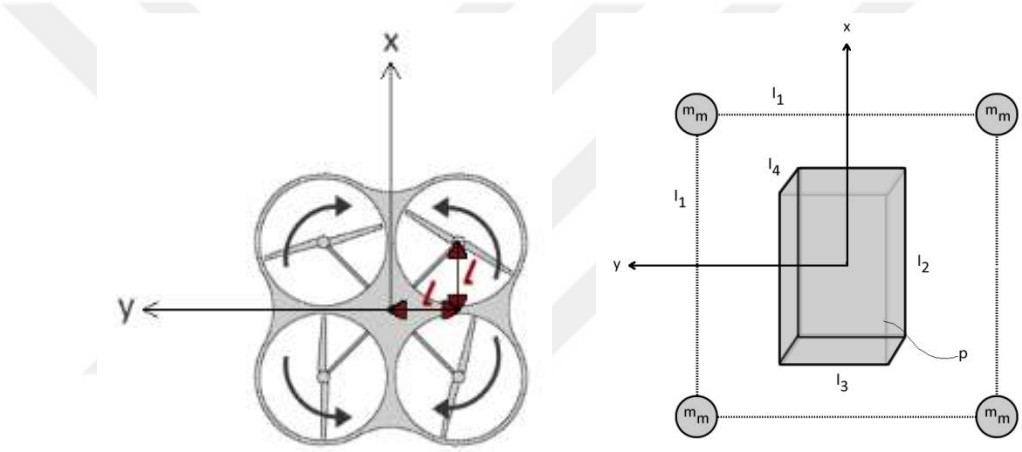


Figure 2.13 × type Quadrotor with simplified masses & volumes (where ρ is body)

For Roll (ϕ) rotation; $\ddot{\phi} = \frac{l}{I_x}(T_1 - T_2 - T_3 + T_4)$ (2.27)

For Pitch (θ) rotation; $\ddot{\theta} = \frac{l}{I_y}(-T_1 - T_2 + T_3 + T_4)$ (2.28)

For Yaw (ψ) rotation; $\ddot{\psi} = \frac{K_l}{I_z}(T_1 - T_2 + T_3 - T_4)$ (2.29)

2.4 Motion Control and Simulation

A quadrotor is a type of helicopter that has four equal rotors at the corners. With the help of independent rotors, this aircraft does not need complicated swash plate mechanisms. This mechanism helps conventional helicopter to perform high

maneuvers just with two rotors but increases mechanical complexity. Controlling more rotors with mechanics is nearly impossible. But using electronic motor control and avionic measurement systems it can be easily performed.

Quadrotor can be controlled with only four rotor speeds. But it has six degrees of freedom, three rotational and three translational. These motions are coupled in order to perform 6 DoF. Therefore, dynamics of the quadrotor is nonlinear especially when aerodynamics effects are considered. Aircrafts must supply their own damping for stopping and hovering because aircrafts have very small amount of friction than ground vehicles.

In order to start simulation application, dynamics of a quadrotor is needed to define. The first stage is kinematic formulations. As general we have two main coordinate spaces, one of them is ground and the other is the vehicle coordinate space shown in Figure 2.14. General notation of gravity is towards –z.

In ground coordinate space roll, pitch and yaw angles can be defined as $\theta = (\phi, \theta, \psi)^T$ and angular velocities $\dot{\theta} = (\dot{\phi}, \dot{\theta}, \dot{\psi})^T$. Position of the quadrotor can be defined as $x = (x, y, z)^T$ and velocity $\dot{x} = (\dot{x}, \dot{y}, \dot{z})^T$ in vehicle coordinate space. Vector pointing along the axis of rotation, which is defined as angular velocity vector, ω according to ground is different than time derivative of angles, $\dot{\theta}$ (roll, pitch or yaw). And rotational matrix from ground to vehicle, R can be defined as given in Equation 2.29. So, we can define $\vec{v}$ in ground and as $R\vec{v}$ in vehicle coordinate space.

The second need for simulation is defining physical properties such as motors, forces or torques. As we mentioned before, quadrotor has four identical motors. If we analyze of them, we can expand the information to whole. We also mentioned that propellers are group in two; clockwise and counterclockwise in order to keep aircraft balanced when they spin at the same rate.

$$\omega = \begin{bmatrix} 1 & 0 & -\sin\theta \\ 0 & \cos\phi & \cos\theta\sin\phi \\ 0 & -\sin\phi & \cos\theta\cos\phi \end{bmatrix} \begin{bmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{bmatrix} \tag{2.28}$$

$$R = \begin{bmatrix} \cos\phi\cos\psi - \cos\theta\sin\phi\sin\psi & -\cos\psi\sin\phi - \cos\phi\cos\theta\sin\psi & \sin\theta\sin\psi \\ \cos\theta\cos\psi\sin\phi + \cos\phi\sin\psi & \cos\phi\cos\theta\cos\psi - \sin\phi\sin\psi & -\cos\psi\sin\theta \\ \sin\phi\sin\theta & \cos\phi\sin\theta & \cos\theta \end{bmatrix} \quad (2.29)$$

In the thesis, we are using a ready to flight quadrotor, Parrot Inc.'s AR Drone. It has four brushless motors. The torque produced by each motor can be express as $\tau = K_t(I - I_0)$. τ is the torque of motor, I is the input current and I_0 is the current in zero load on motor, and K_t is the motor torque constant. The voltage of the motor, V, which is the sum of back EMF and resistive losses can be defined as $V = IR_m + K_v\omega$ where R_m is the resistance of motor, ω is the motor's angular velocity and K_v is motor velocity constant for defining back EMF generated by revolution per minute.

From motor torque and voltage Equations, power can be defined in Equation 2.29. In order to simplify the model for simulation, we are going to ignore the motor resistance. So, power consumption equation can become as given in Equation 2.30. And, when there is no load, I_0 and indirectly K_tI_0 is become very less than τ, so that we can also neglect the K_tI_0 term. This last simplification makes power consumption as simple as like in Equation 2.31. This power helps aircraft in hovering position on the air. From the energy conservation, the motor energy is consumed in a certain time, is also equal to force generated on the propeller times air displacement rate. This can be defined as $P = F\frac{dx}{dt}$. And power also equal to thrust times velocity. This can be also defined as $P = Tv_a$. We can express v_a by using momentum theory. With theory, in order to hover, power requirement is defined as in Equation 2.32. Where ρ is air density and A is the area of rotor disc. We can rewrite this equation for v_a such as in Equation 2.33. And we can also reconsider our simplified power Equations like in Equation 2.34. If we solve these equalities for thrust, T, we have relation given in Equation 2.35.

$$P = \frac{(\tau + K_tI_0)(K_tI_0R_m + \tau R_m + K_tK_v\omega)}{K_t^2} \quad (2.29)$$

$$P = \frac{(\tau + K_tI_0)(K_v\omega)}{K_t} \quad (2.30)$$

$$P = \frac{K_v}{K_t}\,\omega\tau \tag{2.31}$$

$$P = \sqrt{\frac{T^3}{2\rho A}} \tag{2.32}$$

$$v_a = \sqrt{\frac{T}{2\rho A}} \tag{2.33}$$

$$P = \frac{K_v}{K_t}\omega\tau = \frac{K_v K_\tau}{K_t} T\omega = \sqrt{\frac{T^3}{2\rho A}} \tag{2.34}$$

$$T = \left(\frac{K_v K_\tau \sqrt{2\rho A}}{K_t}\omega\right)^2 = k\omega^2 \tag{2.35}$$

Where k is a dimensional constant. The total thrust of quadrotor in vehicle coordinate system defined as in Equation 2.36. And for the simulations, drag force, FD, can be defined as following Equation when it is directly proportional to linear velocity in each direction. Friction constant, k_d, may be expressed in each direction in order to increase the simulation precision as in Equation 2.37.

$$T_V = \sum_{i=1}^{4} T_i = k\begin{bmatrix}0 & 0 & \sum \omega_i^2\end{bmatrix}^T \tag{2.36}$$

$$F_D = \begin{bmatrix}-k_d\dot{x} \\ -k_d\dot{y} \\ -k_d\dot{z}\end{bmatrix} \tag{2.37}$$

After defining power consumption of rotors and force generation of quadrotor, we also need to define torques. Each motor on quadrotor generate torque about z axis in vehicle coordinates. This generated torque provides thrust, creates angular acceleration and overcomes the drag forces.

In general, the drag force can be defined as $F_D = \frac{1}{2}\rho C_D A v^2$. But in this definition, A is not the area of rotor disc, it is propeller's cross section. Torque is also known as $F_D R$ so, the rewritten Equation of torque expressed in Equation 2.38. For the

simplification, we can take all dimensioned constant as b so that torque from drag is $\tau_D = b\omega^2$. And total torque about the z axis for a rotor is $\tau_z = b\omega^2 + I_M\dot{\omega}$ where I_M is inertia moment, $\dot{\omega}$ is angular acceleration and b is the drag coefficient. Angular acceleration is nearly zero if not in takeoff or landing maneuver. So, we can simplify it to $\tau_z = \pm b\omega^2$ and sign notation is used for determining clockwise (+) and counterclockwise (-) spin of the motor. Total torque from all four motors and their propellers about z axis also known as yaw torque is defined as in Equation 2.39.

$$\tau_D = R\frac{1}{2}\rho C_D A v^2 = R\frac{1}{2}\rho C_D A(\omega R)^2 \tag{2.38}$$

$$\tau_\psi = b(\omega_1^2 - \omega_2^2 + \omega_3^2 - \omega_4^2) \tag{2.39}$$

Pitch and roll torque can be generated via force, T, times (cross product) level arm distance, L in Equation 2.40 and 2.41. We can also express all torques in matrix form such as in Equation 2.42. Lastly, governing equations of motion are derived. Thrust, gravity and linear friction causes quadrotor's acceleration. The thrust vector with respect to ground coordinate can be derived using R, the rotation matrix and thrust vector from vehicle to ground coordinates. So that the linear motion will become as follow when x is the quadrotor's position, g is gravity, T_B is thrust in vehicle coordinate and F_D is drag force given in Equation 2.43. But also, we can take these motion equations from inertial center to the center of the quadrotor. These rotational Equations of motion can be generated from Euler's Equations for rigid body dynamics given in Equation 2.44, where τ is torque (externals), ω is angular velocity and I is the inertia matrix. And also, for each axis in Equation 2.45.

$$\tau_\theta = \sum r \times T = Lk(\omega_2^2 - \omega_4^2) \tag{2.40}$$

$$\tau_\phi = Lk(\omega_1^2 - \omega_3^2) \tag{2.41}$$

$$\tau_B = \begin{bmatrix} Lk(\omega_1^2 - \omega_3^2) \\ Lk(\omega_2^2 - \omega_4^2) \\ b(\omega_1^2 - \omega_2^2 + \omega_3^2 - \omega_4^2) \end{bmatrix} \tag{2.42}$$

$$m\ddot{x} = \begin{bmatrix} 0 \\ 0 \\ -mg \end{bmatrix} + RT_B + F_D \tag{2.43}$$

$$\tau = I\dot{\omega} + \omega \times (I\omega) \tag{2.44}$$

$$\dot{\omega} = \begin{bmatrix} \dot{\omega_x} \\ \dot{\omega_y} \\ \dot{\omega_z} \end{bmatrix} = I^{-1}(\tau - \omega \times (I\omega)) \tag{2.45}$$

For simplification, quadrotor's inertia matrix can be constructed from two uniform crossed rods and motors as a point mass like in Equation 2.46. As a result, we can get the governing equations with respect to vehicle coordinate system given in Equation 2.47.

$$I = \begin{bmatrix} I_{xx} & 0 & 0 \\ 0 & I_{yy} & 0 \\ 0 & 0 & I_{zz} \end{bmatrix} \tag{2.46}$$

$$\dot{\omega} = \begin{bmatrix} \tau_\phi I_{xx}^{-1} \\ \tau_\theta I_{yy}^{-1} \\ \tau_\psi I_{zz}^{-1} \end{bmatrix} - \begin{bmatrix} \frac{I_{yy} - I_{zz}}{I_{xx}} \omega_y \omega_z \\ \frac{I_{zz} - I_{xx}}{I_{yy}} \omega_x \omega_z \\ \frac{I_{xx} - I_{yy}}{I_{zz}} \omega_x \omega_y \end{bmatrix} \tag{2.47}$$

We calculate and define various kind of Equation in order to describe a quadrotor in terms of mathematical model. The main reason is to create a controller both in simulations and in real world. A quadrotor's inputs are angular velocities of its rotors which is controlled by voltages. So, we can also define our system in state space Equations as follow.

We can define quadrotor's position in space as $\dot{x_1} = x_2$. Then quadrotor's linear velocity is given in Equation 2.48, Euler angles in Equation 2.49 and angular velocity vector is in Equation 2.50.

$$\dot{x_2} = \begin{bmatrix} 0 \\ 0 \\ -mg \end{bmatrix} + RT_B + F_D \tag{2.48}$$

$$\dot{x_3} = \begin{bmatrix} 1 & 0 & -\sin\theta \\ 0 & \cos\phi & \cos\theta\sin\phi \\ 0 & -\sin\phi & \cos\theta\cos\phi \end{bmatrix}^{-1} x_4 \tag{2.49}$$

$$\dot{x_4} = \begin{bmatrix} \tau_\phi I_{xx}^{-1} \\ \tau_\theta I_{yy}^{-1} \\ \tau_\psi I_{zz}^{-1} \end{bmatrix} - \begin{bmatrix} \frac{I_{yy} - I_{zz}}{I_{xx}} \omega_y \omega_z \\ \frac{I_{zz} - I_{xx}}{I_{yy}} \omega_x \omega_z \\ \frac{I_{xx} - I_{yy}}{I_{zz}} \omega_x \omega_y \end{bmatrix} \tag{2.50}$$

We can use a PD controller for control action. This controller will perform proportional error between desired and observed trajectory and proportional to derivative of error like in Equations from 2.51 to 2.53. On our quadrotor we can only get angular velocity data from IMU which can be expressed as $\dot{\phi}, \dot{\theta}, \dot{\psi}$ and they produce some error, these will be derivative error and their integral will be actual error values. Our first aim is to stabilize the quadrotor. So, in that state, desired output is to zero for all velocities and angles. As we know, torques are related to these angular velocities of our controller output will be proportional with torques like in Equation 2.54. In hovering state, ideally T = mg but we are aware that quadrotor is continuously moving so that in order to perform realistic simulation, we need to consider projection of thrust, T_p which can be calculated by $T_p = mg\cos\theta\cos\phi$, where the angle data is gathered from gyroscopes. So, the real thrust, T, that will be needed to keep the aircraft on the air is given Equation 2.55 or in other terms in Equation 2.56.

$$e_\phi = K_d\dot{\phi} + K_p \int_0^T \dot{\phi} dt + K_i \int_0^T \int_0^T \dot{\phi} dt dt \tag{2.51}$$

$$e_\theta = K_d\dot{\theta} + K_p \int_0^T \dot{\theta} dt + K_i \int_0^T \int_0^T \dot{\theta} dt dt \tag{2.52}$$

$$e_\psi = K_d\dot{\psi} + K_p\int_0^T \dot{\psi}dt + K_i\int_0^T\int_0^T \dot{\psi}dtdt \tag{2.53}$$

$$\begin{bmatrix}\tau_\phi\\ \tau_\theta\\ \tau_\psi\end{bmatrix} = \begin{bmatrix}-I_{xx}(K_d\dot{\phi} + K_p\int_0^T \dot{\phi}\,dt)\\ -I_{yy}(K_d\dot{\theta} + K_p\int_0^T \dot{\theta}\,dt)\\ -I_{zz}(K_d\dot{\psi} + K_p\int_0^T \dot{\psi}\,dt)\end{bmatrix} = \begin{bmatrix}Lk(\omega_1^2 - \omega_3^2)\\ Lk(\omega_2^2 - \omega_4^2)\\ b(\omega_1^2 - \omega_2^2 + \omega_3^2 - \omega_4^2)\end{bmatrix} \tag{2.54}$$

$$T = \frac{mg}{\cos\theta\cos\phi} = k\sum\omega_i^2 \tag{2.55}$$

$$\sum\omega_i^2 = \frac{mg}{k\cos\theta\cos\phi} \tag{2.56}$$

After all dynamic, kinematic, force, torque and motion Equations are defined, we can construct a realistic PD and PID control simulations with MATLAB. Simulation code in Appendix 1 can execute with the single line commands as shown in Table 2.4.

Table 2.4 MATLAB simulation single line commands both for PD and PID control simulation

PD	**show**(
	  run(
	  control('pd', <<P parameter>>, <<D parameter>>),
	  <<Time start>>, <<Time end>>, <<Time step>>
	  )
	)

PID	**show**(
	  run(
	  control('pid', <<P parameter>>, <<I parameter>>, <<D parameter>>),
	  <<Time start>>, <<Time end>>, <<Time step>>
	  )
	)

For a simple control simulation these input parameters can be selected as $K_p = 3$, $K_d = 4$, $t_s = 0$, $t_e = 3$ and $d_t = 0.01$ and the corresponding command line will be *show(run(control('pd', 3, 4), 0, 3, 0.01))*

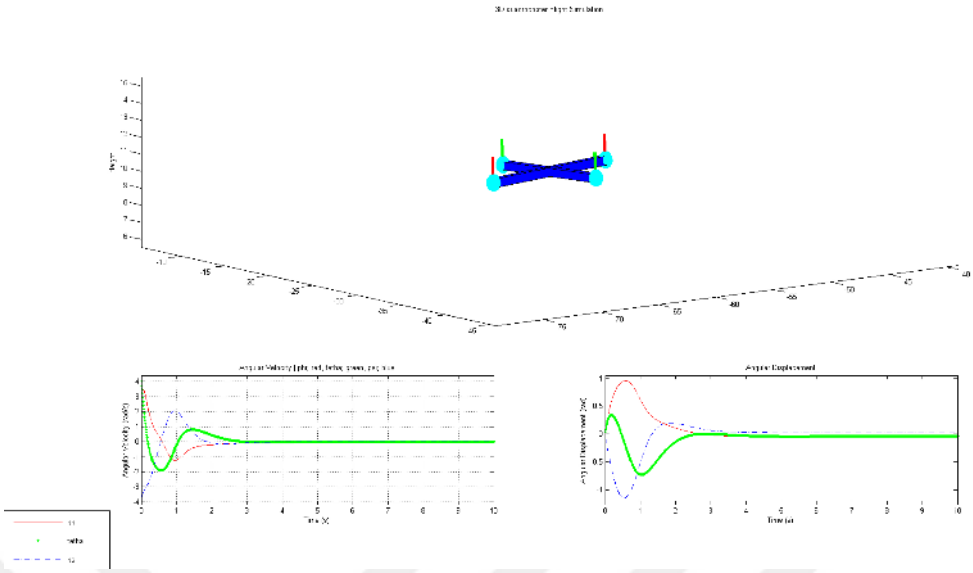


Figure 2.14 3D Quadrotor Flight Simulation for PD Controller

As we can see from angular velocity and displacement plots given in Figure 2.14, we have some steady state error. This is very common with PD controller when noise and disturbance is valid. In order to overcome this error, integral behavior is added. PID controllers decreases steady state error by adapting smoothly. With this kind of behavior, the given trajectory can be followed as seen in Figure 2.15. The inputs are $K_p = 3$, $K_i = 5.5$, $K_d = 4$, $t_s = 0$, $t_e = 3$ and $d_t = 0.01$ with *show(run(control('pid', 3, 5.5, 4), 0, 3, 0.01))* command.

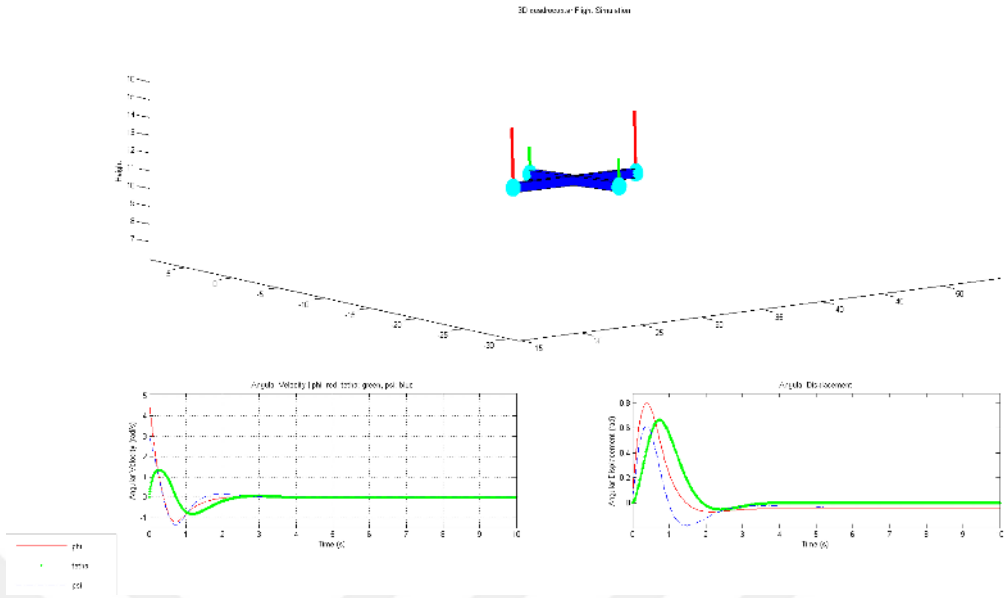


Figure 2.15 3D Quadrotor Flight Simulation for PID Controller

2.5 Introduction to Multi Agent Robotics

In general, multi Agent Systems which uses robot as an agent, are the systems that formed by a large number of relatively simple robots, placed in the same region and interacting to fulfill a common goal. The inspiration of multiple robotic agents arises from insect colonies behavior, such as ants and bees. In these colonies, individuals interact using only local communication and there is no central unit controlling each and every individual. Even so, they can perform complex global behavior and solve hard problems in the real world.

In an environment where there are multiple robots, the trend for a robot to interfere with another robot's execution is greater (Lerman and Galstyan, 2002). A common problem in the navigation of swarms is blockage, which happens when a specific number of robots must move towards the same region simultaneously. For example, it is common that robots have a common target, as in waypoint navigation (Marcolino and Chaimowicz, 2008). Even if they have different destinations, these might be located close to each other, in the same region.

Besides waypoint navigation, we can also see this kind of blockage when a swarm of robots is solving a target searching and/or passing problem (Sahin, 2008 and Sahin,

2004) For example when robots must move to locations in the environment to collect items and transport them to a specific location. This problem has many practical applications in the real world, such as transportation of materials or patrolling from specific waypoints.

The target searching and reaching problems could be solved by using a central processing unit to compute the best trajectory for each robot. However, there is a drawback; the system becomes dependent on this central unit and the solution is not scalable to many robots.

2.6 Formation Control of Multiple Quadrotors

Formation control of unmanned aerial vehicles has also become an area of interest in the robotics community. This field is motivated by the possibility to overcome tasks that a single vehicle is not able to make, changing it by a group of aerial robots of smaller capability. Effectively, the use of multiple robots has several advantages, such as low cost, high strength, performance and efficiency. Instead of making and using a single powerful robot, a multi agent system can be designed, generally, in a simpler and more economical way (Brandao, 2015). The ability to control robots working cooperatively is a main challenge for researchers of robotics and artificial intelligence areas. Interesting results have been published in the literature (Zhang, 2013).

An aerial formation can be defined as a set of two or more aerial vehicles flying together that dynamic states are linked through a common control law. This linking can be expressed in terms of rotational and translational degrees of freedom, as well as velocity or position.

In general, there are three structures for controlling multi-robot systems; leader follower, behavior-based and virtual structures, each of them has their advantages and disadvantages (Chen, 2010 and Consolini, 2008). First one is the leader-follower organization, one of the robots is considered as a leader while the others are followers. In this architecture, each follower deals its information with the leader, but knows nothing about the other followers' state. And, leader has no information about the followers. Therefore, if the leader fails there will be no way to provide that the given

mission successfully executed. This structure is easy to understand and implement. In the second case, the behavior-based structure, the behavior of the aerial vehicle formation is defined as a combination of individual behavior of each member. The main concept of this approach is to formalize it mathematically. In a worst-case scenario, it may not be easy to keep the convergence of the formation to the desired objectives (Figure 2.16). In the third case, the virtual structure can establish geometric relationships that will remain rigid between the robots and the reference system, which can be a virtual point or a virtual robot. An advantage of this method is that the given virtual leader never fails. Therefore, the formation will be kept during the entire mission.

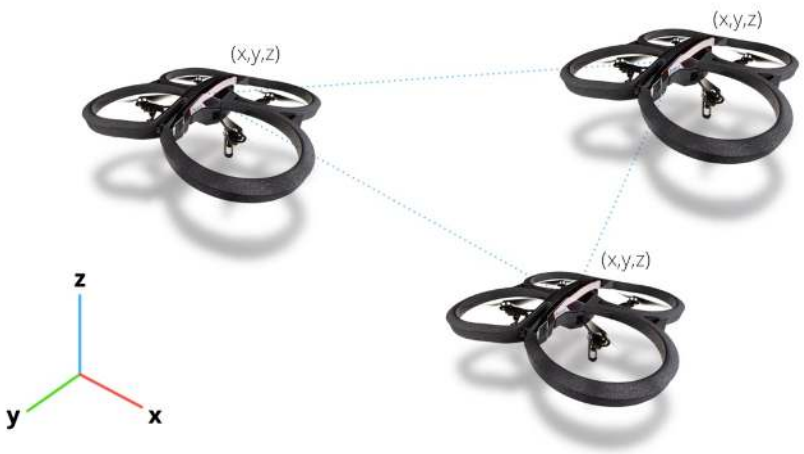


Figure 2.16 An overview of formation representation

CHAPTER THREE
QUADCOPTER PLATFORM

3.1 Parrot Inc. AR Drone 2 Specifications

AR Drone is a low-cost, highly skilled × type quadrotor from Parrot as shown in Figure 3.1. It has a water-resistant body with robust structure via carbon fiber tubes. It weighs 420 g with indoor hull. Quadrotor has four 15 Watt brushless in runner motors. Motor's maximum rotation speed is 41400 RPM. Normal speed during flight is about 28000 RPM. 8" propellers with high attack angle are driven via 1/8.75 reduction from motors axes for high torque gains. Quadrotor's power source is a single Li-Po battery which is 1000 mAh with 10C discharge rate.

Figure 3.1 Parrot AR Drone quadrotor disassembled view (Piskorski, 2010)

This ready to flight quadrotor has also fully open sourced software developer kit (SDK) for academic and personal usage. A generic control loop of quadrotor is shown in Figure 3.2. Main control board has 1 GHz 32-bit ARM Cortex A8 processor with 1 GB 200 MHz DDR2 RAM. It has also 800 MHz dedicated video digital signal processor for on-board vision system. Controller board operates with BusyBox embedded Linux distribution. And also, main controller board has a USB 2.0 extension for extending connectivity. Communication with tele-operator or centralized task controller can be performed via IEEE 802.11 wireless communication module.

Quadrotor has a well optimized internal measurement unit (IMU). In this IMU, there are 3 axis gyroscopes with 2000 °/sec precision, 3 axis accelerometers with ±50 mg precision, 3 axis magnetometers with 6° precision, pressure sensor ± 10 Pa precision (80 cm at sea level) and an ultrasonic sensor for altitude control. AR Drone quadrotor has also 2 on-board cameras. One camera is in the front and the other is at the bottom of the vehicle. Front camera has 1280x720 resolutions at 30 FPS. It has a wide-angle lens with 92° field of view (FOV). Camera capture data is sent through wireless link with using H264 encoding for low latency streaming. Bottom camera has 320x240 resolutions at 60 FPS and this camera is also used for ground speed measurement (Krajnık, 2011).

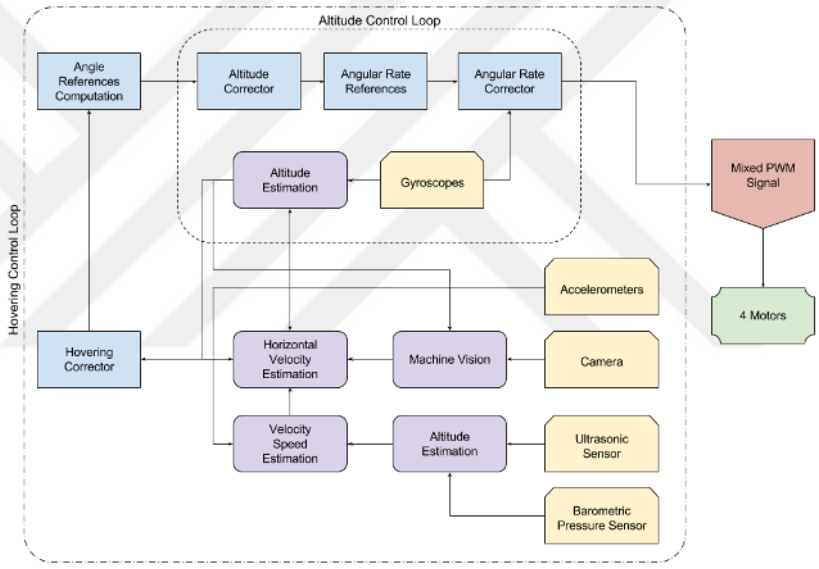


Figure 3.2 Control loop of a quadrotor

3.2 Mechanical Design Verification and Flow Analysis of the Propellers

In order to verify the quadrotor's mechanical design, the entire vehicle is redesigned in 1:1 scale with SolidWorks (Figure 3.3). So that the designed models can be both used in numerical analysis and also can be 3D printed for spare parts.

Figure 3.3 AR Drone 2.0 Solid model

3D modelled propeller by using the original part is analyzed by SolidWorks Flow Simulation tool (Figure 3.4). The calculated velocity values show that both created part and real quadrotor features are compatible.

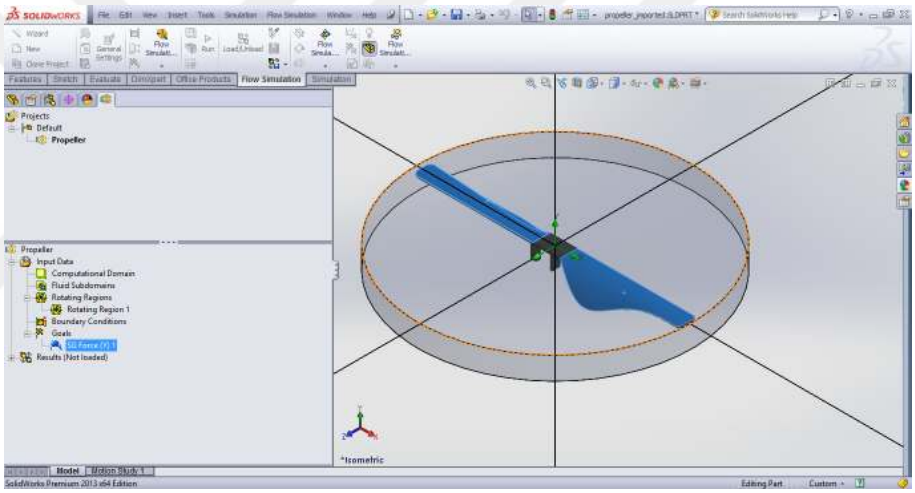


Figure 3.4 Flow simulation with selected rotating regions

As a result of flow analysis, the force (N) in Y axis in order to hover the quadrotor is calculated as 1.717 N as seen in Figure 3.5. Quadrotor's weight is 420g with indoor hull and 380g with outdoor hull. We can verify the total payload from $F = m \times g$ as 175g per motor and 700g for drone. The cargo will be 280g with indoor hull and 320g with outdoor hull. With this cargo capacity, for example professional cameras like GoPro (140g with housing) and/or USB pen drive (14g) and/or USB flight recorder with on-board GPS (31g) and/or event an iPhone 5 (115g) can be mounted.

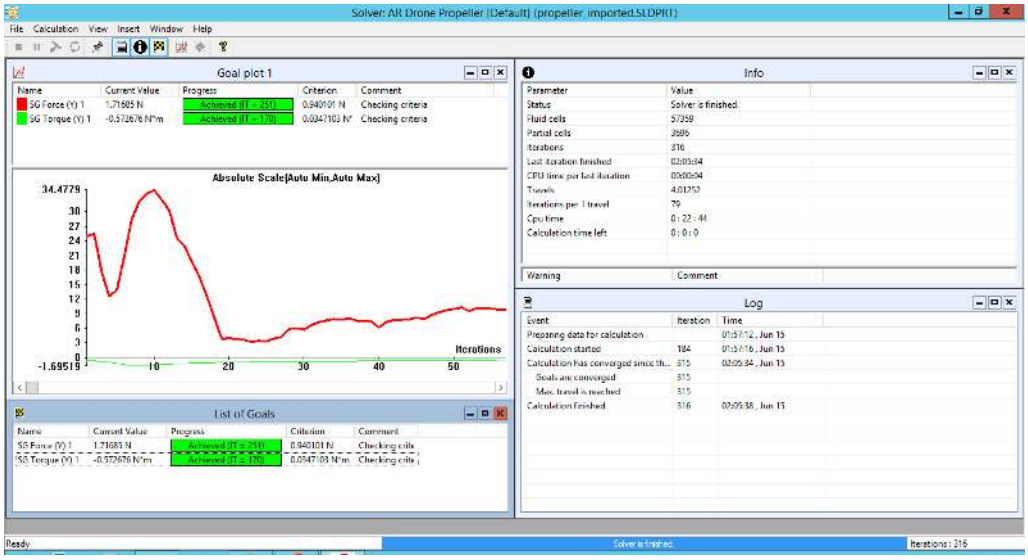


Figure 3.5 Flow simulation solver result screen

In hovering state, each propeller has a rotation speed of 28000 RPM or angular speed from $\omega = 2\pi/60 \times RPM$ as 2932.15 rad/s. The linear velocity can be calculated from $v\ \left[\frac{m}{s}\right] = r\ [m]\ \times\ \omega\left[\frac{rad}{s}\right]$ as 293.22 m/s when propeller radius is 10cm. Flow analysis gives the maximum velocity as 297.46 m/s at the tip of the propeller's blades as seen in Figure 3.6.

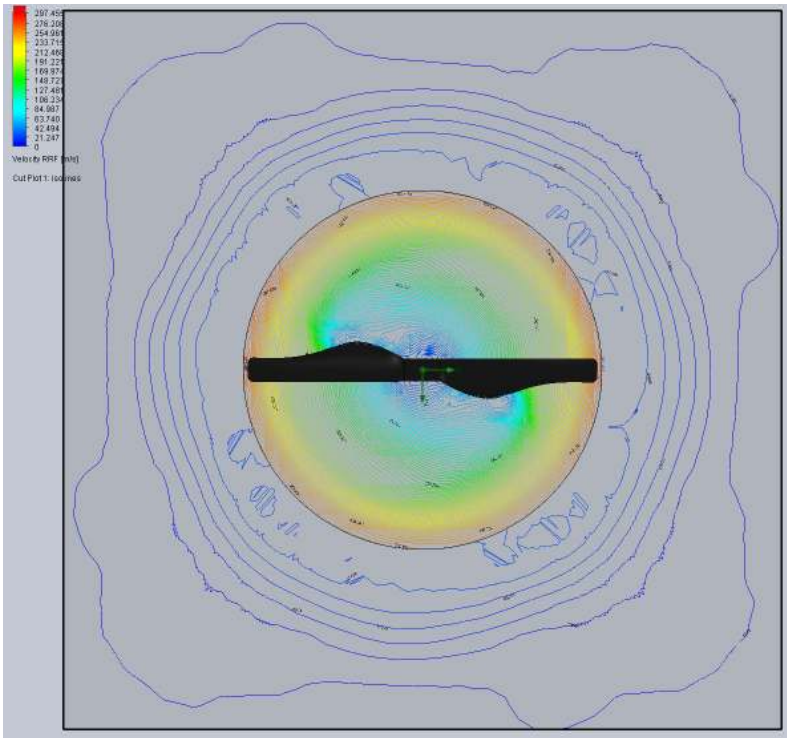


Figure 3.6 Velocity cut plot

3.3 Application Programming Interface and Its Abilities

The AR Drone 2.0 SDK allows third party developers to develop and distribute new games based on AR Drone 2.0 product for Wi-Fi, motion sensing mobile devices like the Apple iPhone, iPad, iPod touch, personal computers or Android devices.

This SDK contains three main parts;

- ARDroneLIB which provides the APIs needed to easily communicate and configure an AR Drone 2.0 product,
- ARDroneTool which provides a fully functional drone client where developers only have to insert their custom application specific code for testing and/or executing
- Control Engine library which provides an intuitive control interface developed by Parrot for remotely controlling the AR Drone 2.0 product from an iOS Device

According to SDK, quadrotor reference geometry is shown in Figure 3.7 that each pair of opposite rotors is turning the same way. One pair is turning clockwise and the other counterclockwise.

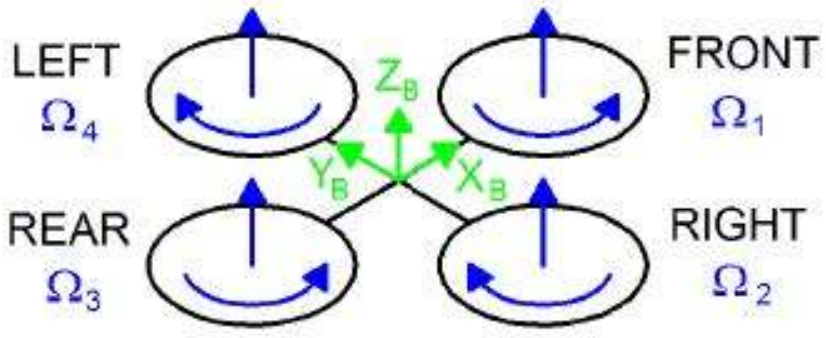


Figure 3.7 AR Drone 2.0 reference geometry (Piskorski, 2012)

Maneuvers are performed by altering pitch, roll and yaw angles of the quadrotor as illustrated in Figure 3.8. Varying left and right rotors' speed, the opposite way yields roll movement. This allows to go forth and back. Varying front and rear rotors speed

the opposite way yields pitch movement. Varying each rotor pair speed, the opposite way yields yaw movement. This allows turning left and right (Figure 3.8).

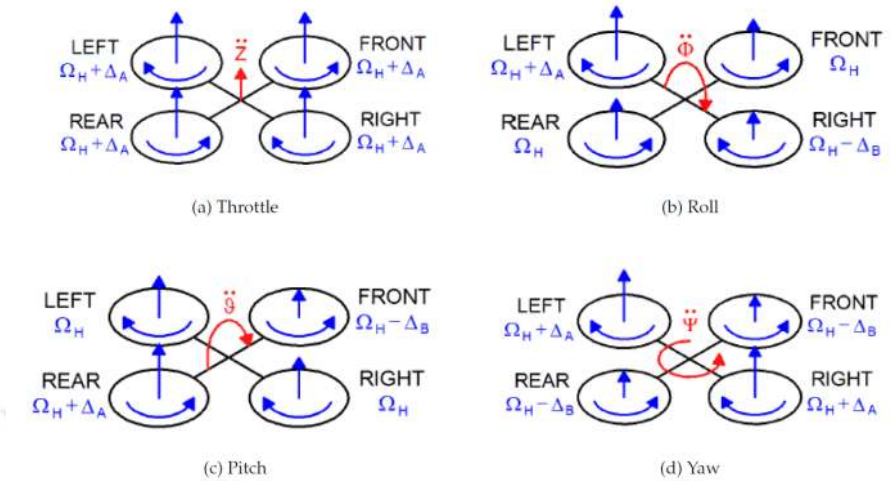


Figure 3.8 AR Drone 2.0 movements (Piskorski, 2012)

Control of the AR Drone is done through 3 main communication services;

- 5554 (UDP): by sending API commands. The default data frequency is 200Hz. These commands can handle status, position, speed, motors' rotation speed etc. (Figure 3.9).
- 5555 (TCP): streaming both (frond and bottom) videos. The codec for decoding streamed live videos are also shared in the same SDK. For bottom camera, FPS rate is 15. For front camera FPS can be changed from 15 to 30.
- 5556 (UDP): by sending AT commands. The transmission latency of the control commands is critical to the user experience and they need to be sent on a regular basis. The minimum acceptable data frequency is 30Hz.

Header	Drone's State	Sequence Number	Vision Flag	Option #1			Option #2
				ID	Size	Data	...
32 bit INT	32 bit INT	32 bit INT	32 bit INT	32 bit INT	32 bit INT	32 bit INT/FLOAT Array	...
always 0x55667788	01001111100000000000010001010100 (Example state value)						

Option # 1										
ID	Size	Data								
		Control State	Battery Voltage	Pitch	Roll	Yaw	Altitude	Vx	Vy	Vz
uint16	uint16	uint32	uint32	float32	float32	float32	int32	float32	float32	float32

Figure 3.9 Navigation Data Sample

And a last post, 5559 (TCP) is also in use for sending control and configuration data. This information is the most critical ones for quadrotor's current flight. By using this port, controller can get current configuration and verify control commands (Figure 3.10).

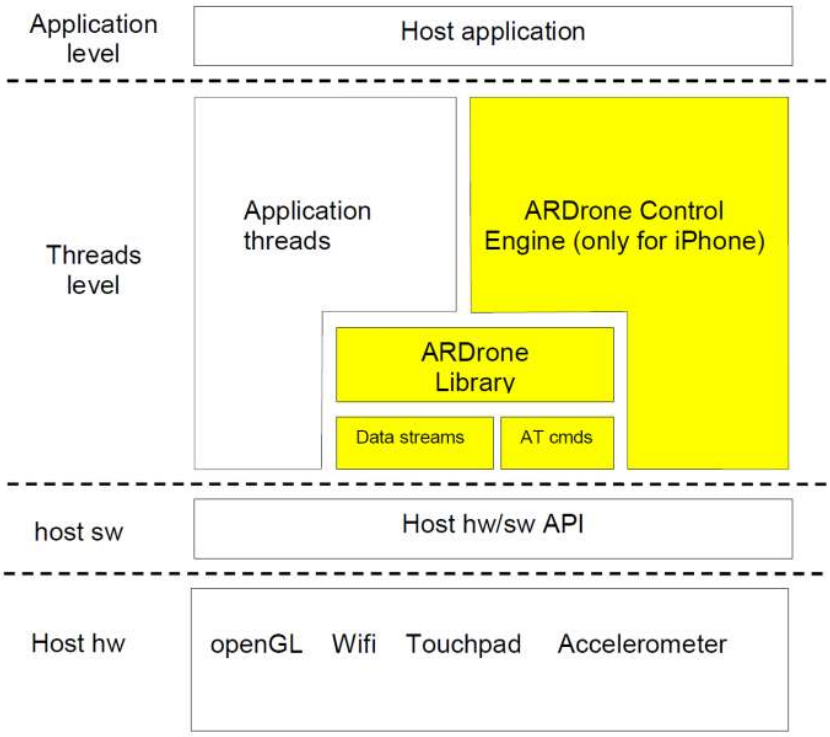


Figure 3.10 Architecture layer of a host application built upon the SDK (Piskorski, 2012)

According to official SDK documentation it only allows to write applications to remotely control of a single drone from Linux kernel-based OS with Wi-Fi, an Apple iOS or an Android device (Piskorski, 2012). And in the same documentation, it clearly states that there is no support of rewriting your own embedded software - no direct access to the drone hardware (sensors, engines) is allowed.

3.4 Hacking Flying Linux Kernel

As we mention in the previous section, original SDK and API structure does not support operating system level modification or custom module buildings. Hardcoded flight control parameters, on-board computer vision daemon for color detection, DHCP connection routines and built a secure wireless communication are tweaked. Because AR Drone 2.0 comes with open access unsecure default wireless settings and

there is no way to implement any WPA or WEP security, a custom compiled version of wpa_supplicant, wpa_cli and wpa_passphrase are needed (Araos, 2013). With this additional security, a private network for a group of quadrotor and client node can be easily configured without any need of additional router or access point.

- wpa_supplicant is a WPA Supplicant for Linux, BSD, Mac OS X, and Windows with support for WPA and WPA2 (IEEE 802.11i / RSN). It is suitable for both desktop/laptop computers and embedded systems. Supplicant is the IEEE 802.1X/WPA component that is used in the client stations. It implements key negotiation with a WPA Authenticator and it controls the roaming and IEEE 802.11 authentication/association of the WLAN driver (Malinen, 2013).
- wpa_cli is a text-based frontend program for interacting with wpa_supplicant. It is used to query current status, change configuration, trigger events, and request interactive user input (Malinen, 2007).
- wpa_passphrase is used for generating a WPA PSK from a given ASCII passphrase with a valid SSID (service set identifier aka Wi-Fi network's given name) (Malinen, 2007).

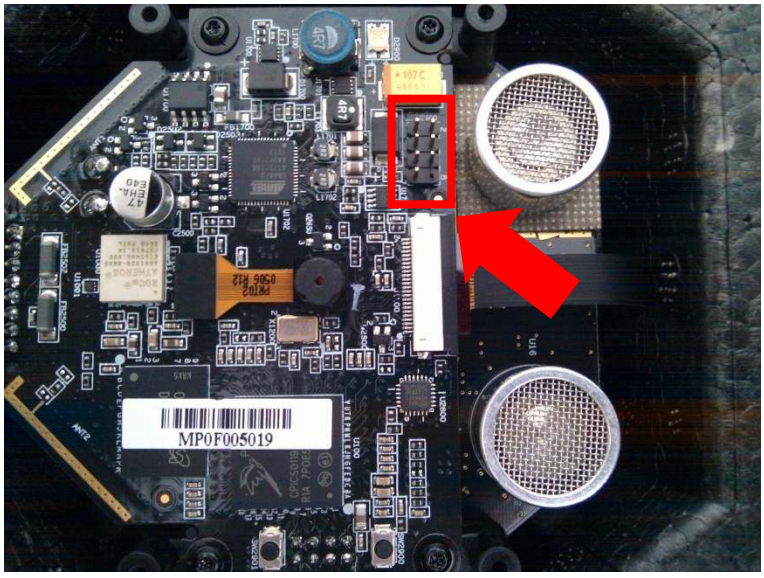
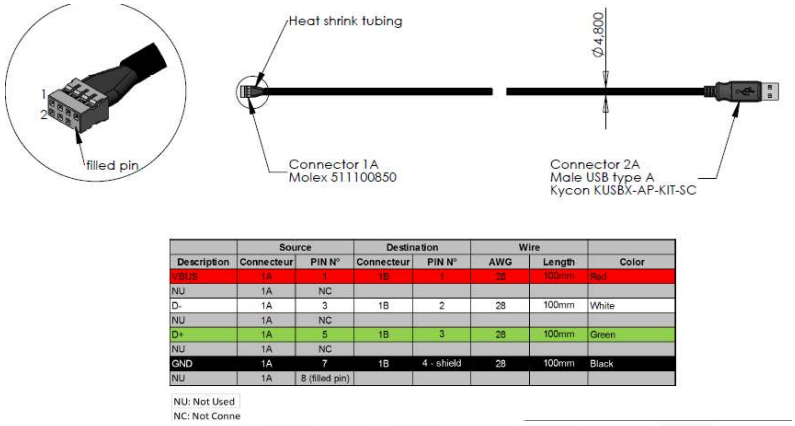


Figure 3.11 AR Drone 2.0 controller board and its debug port (Personal archive, 2017)

In order to access and build Linux kernel after setting up built tools, compliers and QEMU (Quick Emulator). A physical connection must be established for debug and

recover purposes which is shown in Figure 3.11. Luckily AR Drone 2.0 has a hidden debug port and open source community discovered the pinouts as seen in Figure 3.12.



	Source		Destination		Wire		
Description	Connecteur	PIN N°	Connecteur	PIN N°	AWG	Length	Color
VBUS	1A	1	1B	1	28	100mm	Red
NU	1A	NC					
D-	1A	3	1B	2	28	100mm	White
NU	1A	NC					
D+	1A	5	1B	3	28	100mm	Green
NU	1A	NC					
GND	1A	7	1B	4 - shield	28	100mm	Black
NU	1A	8 (filled pin)					

NU: Not Used
NC: Not Conne

Figure 3.12 AR Drone 2.0 debug pinouts and cable (Piskorski, 2012)

In order to be able to flash a custom kernel to A.R Drone, it is necessary to understand it's boot sequence. After reset, the program counter starts at the beginning P6 internal ROM. In this internal ROM, some kind of pre-bootloader exists that evaluates external pins and selects the corresponding boot mode (e.g. boot over USB, NAND, IIC, UART, etc.). Depending on the selected mode, the peripherals are set-up and it is tried to boot over this peripheral.

- Boot over NAND: The pre-bootloader starts the memory controller, copies the bootloader from NAND to internal RAM and sets the program counter to the address in internal RAM which basically starts the bootloader.
- Boot over USB: The pre-bootloader listens to USB and waits for the "Hello P6" command. If received, it replies and awaits an image (the usb_bootloader.bin). The received data is copied to internal RAM and the program counter is changed to the internal RAM address which starts the bootloader.

Depending on the started bootloader, either the UBI partitions are "mounted" and the kernel image is read, or the bootloader waits until the kernel image is sent over USB.

If the installer.plf (which is basically a kernel image) is booted over USB, the "init" command of this image awaits the actual firmware (ardrone_update.plf) over USB and installs the content to NAND (Scorp2kk, 2016).

3.5 Embedded Machine Vision

Object detection and segmentation is the most important and challenging fundamental task of computer vision. It is a critical part in many applications such as image search, scene understanding, etc. However, it is still an open problem due to the variety and complexity of object classes and backgrounds. The easiest way to detect and segment an object from an image is the color-based methods. The object and the background should have a significant color difference in order to successfully segment objects using color-based methods.

To detect colors in images, the first thing you need to do is to define the upper and lower limits for your pixel values. Once you have defined your upper and lower limits, you then make a call to a range method which returns a mask, specifying which pixels fall into your specified upper and lower range. Finally, now that you have the mask, you can apply it to your image using the per-element bit-wise conjunction function to get the image with filtered color. In our situation, we use same algorithm to detect distinct colors as shown in Table 3.1. In order to overcome with barrel disruption, simple CV module crops the original image and gives an output between 0 and 1000 for left-right and top-bottom. Distance parameter has also the same range, 0 is near and 1000 is far.

Table 3.1 Color marking pseudocode

```
for all pixels of an image {
      Get color of the current pixel as color
      if color is within (reference color - threshold) and (reference
      color + threshold) {
            Mark this pixel
      }
}
```

Connected component labeling or blob detection is an algorithmic application of graph theory, where subsets of connected components are uniquely labeled based on a

given heuristic. Connected component labeling is not to be confused with segmentation. Blob detection methods are aimed at detecting regions in a digital image that differ in properties, such as brightness or color, compared to surrounding regions. Informally, a blob is a region of an image in which some properties are constant or approximately constant; all the points in a blob can be considered in some sense to be similar to each other.

Given some property of interest expressed as a function of position on the image, there are two main classes of blob detectors: (i) differential methods, which are based on derivatives of the function with respect to position, and (ii) methods based on local extrema, which are based on finding the local maxima and minima of the function. With the more recent terminology used in the field, these detectors can also be referred to as interest point operators, or alternatively interest region operators (see also interest point detection and corner detection). There are several motivations for studying and developing blob detectors. One main reason is to provide complementary information about regions, which is not obtained from edge detectors or corner detectors. In early work in the area, blob detection was used to obtain regions of interest for further processing.

These regions could signal the presence of objects or parts of objects in the image domain with application to object recognition and/or object tracking. In other domains, such as histogram analysis, blob descriptors can also be used for peak detection with application to segmentation. Another common use of blob descriptor is as main primitives for texture analysis and texture recognition. In more recent work, blob descriptors have found increasingly popular use as interest points for wide base-line stereo matching and to signal the presence of in-formative image features for appearance-based object recognition based on local image statistics. There is also the related notion of ridge detection to signal the presence of elongated objects.

In order to overcome with barrel disruption, simple CV module crops the original image and gives an output between 0 and 1000 for left-right and top-bottom. Distance parameter has also the same range, 0 is near and 1000 is far. Rotation parameter return in degrees.

Camera calibration is a process in order to determine intrinsic camera matrix and distortion vector in order to converge ideal pinhole camera model. Camera matrix construct from focal lengths and image center. Distortion vector has radial and tangential distortion coefficients. These matrixes are used for removing distortions by remapping pixels. Radial distortion correction for each axis where k_i is radial distortion coefficient, is given in Equation 3.1 and 3.2.

$$x_{corrected} = x(1 + k_1 r^2 + k_2 r^4 + k_3 r^6) \tag{3.1}$$

$$y_{corrected} = y(1 + k_1 r^2 + k_2 r^4 + k_3 r^6) \tag{3.2}$$

And for tangential distortion correction, the relevant Equations are expressed in Equation 3.3 and 3.4. Distortion vector is defined in Equation 3.5. Remapping for correction vector is defined in Equation 3.6.

$$x_{corrected} = x + [2p_1 xy + p_2(r^2 + 2x^2)] \tag{3.3}$$

$$y_{corrected} = y + [p_1(r^2 + 2y^2) + 2p_2 xy] \tag{3.4}$$

$$Distortion\ Vector\ (D) = \begin{bmatrix} k_1 & k_2 & p_1 & p_2 & k_3 \end{bmatrix} \tag{3.5}$$

$$Camera\ Matrix\ (A) = \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \end{bmatrix} \tag{3.6}$$

$$A = \begin{bmatrix} 565.007446 & 0 & 323.791901 \\ 0 & 564.219421 & 186.193359 \\ 0 & 0 & 1 \end{bmatrix} \tag{3.7}$$

$$D = [-0.5186495 \quad 0.2849614 \quad -0.0014355 \quad -0.00138001\ 0.0] \tag{3.8}$$

Where f_x, f_y are focal lengths in pixel and c_x, c_y are image center coordinates. In our case our camera matrix and distortion vector are calculated as given in Equation 3.7 and 3.8. Visual representation of the results is displayed in Figure 3.13.

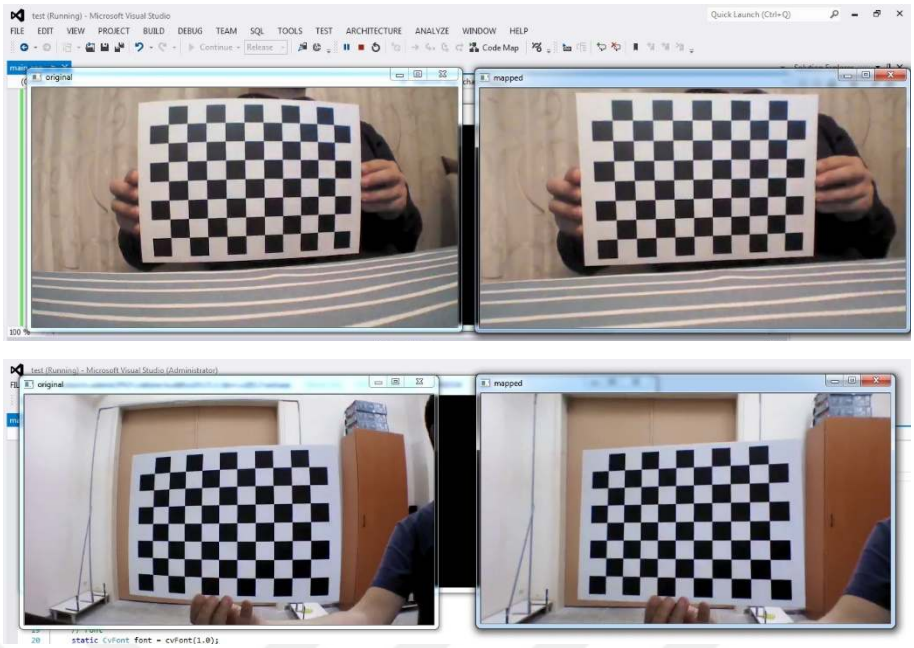


Figure 3.13 AR Drone front camera image un-calibrated and remapped (Personal archive, 2015)

CHAPTER FOUR
"PHDRONE" SOFTWARE ARCHITECTURE

4.1 Architecture Design

Node.JS is actually a software platform, which uses JavaScript programming language, and by using non-blocking I/O and asynchronous events it can handle real time application. It uses Google's V8 JavaScript engine and standard JavaScript modules in execution layer. It also has built in library for file processing, socket operation and HTTP communication with asynchronous I/O capability. Node.JS architecture does not depend on system's RAM like another server-client architecture. The standard web service technique creates a new thread for each (client) connection or requests and takes place in some portion in the system's RAM (Figure 4.1).

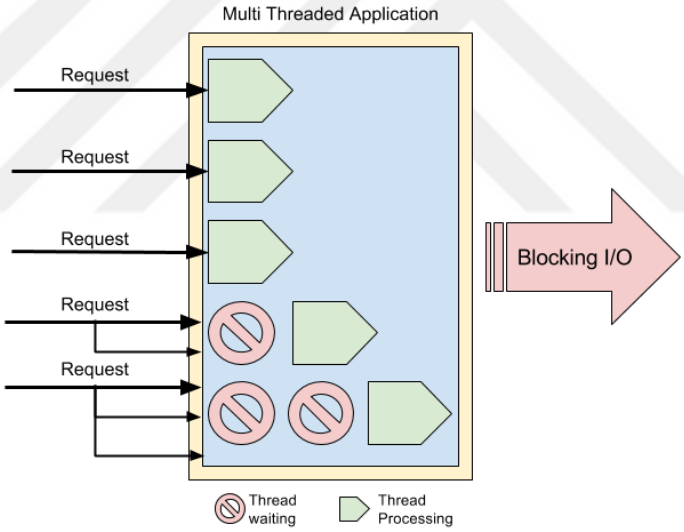


Figure 4.1 Multi-threaded application

On heavy load, more threads lead to more context switching which in turn leads to more CPU and memory usage. And eventually increasing connections consume system's RAM. But Node.JS, which is using only single thread by using non-blocking input/output calls can allow excessive amount of concurrent connection independently from RAM and also the cost of context switching between multiple threads (Figure 4.2). In Node.JS, a single thread can be shared by all requests but all the function that uses input/output operation must have a callback. A callback is a code block that passes

arguments to another code and executes it after some time. In synchronous or blocking callbacks, execution is right away. But in asynchronous or non-blocking callbacks it can happen after some specific time. Types of the callback determine data control in the runtime. When a function or code block processes, the other function or code block does not remain idle while waiting for the response. With the help of the callbacks, there is no need to use any time control (delaying, pausing etc.) or interrupts (with hardware support.)

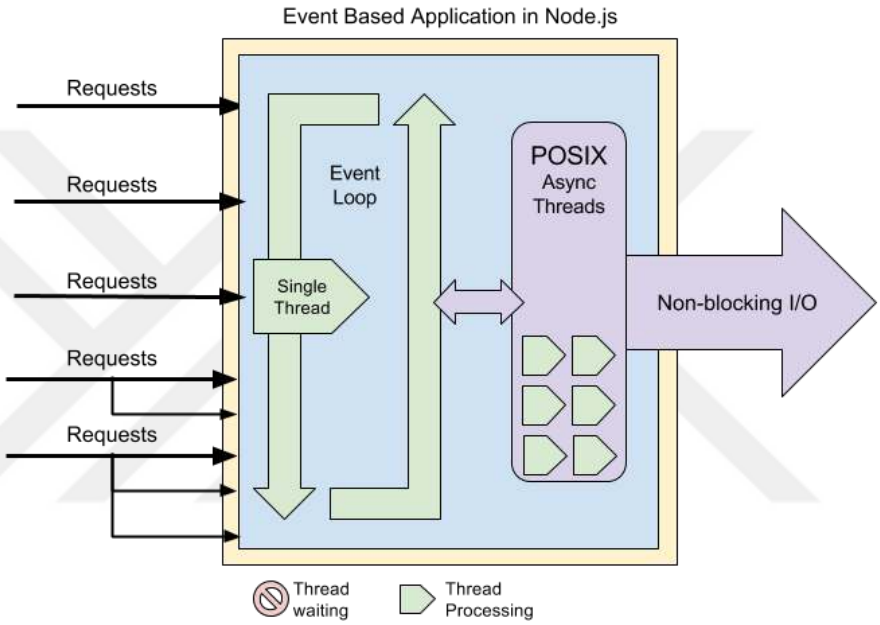


Figure 4.2 Event Based Application in Node.js

Although this platform has advantages, it has some disadvantages which must be considered while developing any robotics applications. First one is, writing style and event driven mindset is completely different than synchronous code. Callback structures may become spaghetti code if they are not planned well. Exception handling structure is also different than straight forward methods.

In the decision-making progress for platform selection, many alternative programming languages and platforms have been tried. Such as Visual C++ with multi-threaded, QT with and without multi-threaded, Python (standalone) with single threaded architecture, C++ and Python with ROS (Robot Operating System) packages. And all of them are fail for one or more things that this thesis wants to achieve.

The software project as a result of this study's code name is PhDrone. This software package thanks to Node.js can run in any operating system such as Windows, Linux, macOS. And also in different CPU architectures like x86, x64, ARM. It consists some key architecture layers in order to work (Figure 4.3). The first layer is application server which handles the main logic of the whole system. This controls whether components of the system run without any problem, if some irregularities are detected it can restart them or list of the quadrotor that can be flight with this application etc.

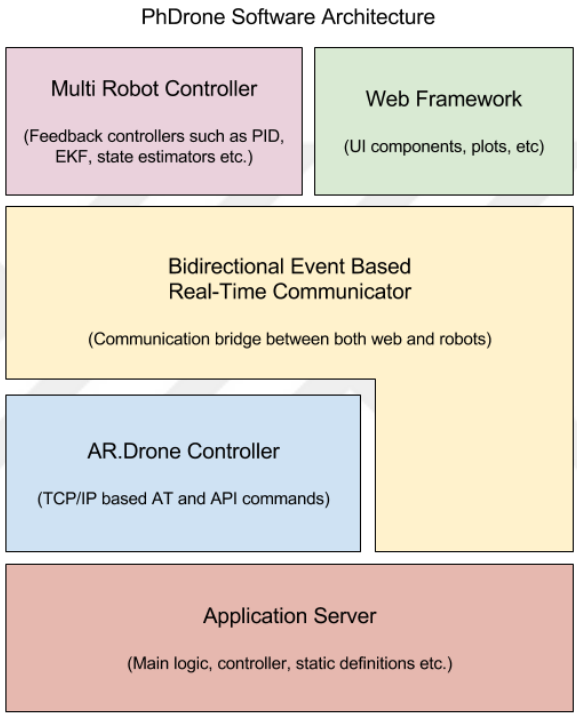


Figure 4.3 PhDrone Software Architecture

On the second layer, both AR Drone controller and communicator is active. Because while communicator acts like a bride to upper layers, AR Drone controller has some low-level settings and control routines that never published or used at the top layers.

AR Drone controller is brief copy of official SDK's logic (Geisendörfer, 2014). With the help of this controller a single quadrotor can be controlled from command prompt or static one-time runnable JavaScript files as shown in Figure 4.4. In this example quadrotor will take off right after the code is executed. After 5 seconds of hovering, it is going to make clockwise yaw maneuver for 5 seconds with a half of its

maximum permissible speed. Then it remains on that position while on the air and make another 1 second clockwise yaw movement. When this event is executed, quadrotor will stop and initiate landing sequence.

```
var arDrone = require('..');
var http    = require('http');

var client = arDrone.createClient();
client.disableEmergency();

server.listen(8080, function() {
  console.log('Serving on port 8080 ...');
  client.takeoff();

  client
    .after(5000, function() {
      this.clockwise(0.5);
    })
    .after(5000, function() {
      this.stop();
    })
    .after(5000, function() {
      this.clockwise(0.5);
    })
    .after(1000, function() {
      this.stop();
      this.land();
    });
});
```

Figure 4.4 A simple AR Drone controller single JavaScript file

AR Drone controller layer can control simple commands such as take-off, stop, land, altitude change (up/down), yaw (ccw/cw), pitch (forward/rear), roll (left/right), recover from emergency state (if or when quadrotor crashes etc.)

Communicator layer enables real-time, bi-directional communication between web clients and servers. Client-side part of the library is running in the browser, and a server-side library for applications that runs on top of Node.Js (Little, 2016). It primarily uses the WebSocket protocol with pooling as a fallback option in the same interface. It can be used a brief wrapper for WebSocket but it has more features if

needed. Such as broadcasting to multiple sockets, storing data associated with each client and asynchronous I/O.

Web framework layer is a novel web-based user interface which you can assign task to each drone in the group (Figure 4.5). The missions can be seen before the execution and also the flight log, the black box, of each quadrotor can be recalled and analyzed. This framework build on top of some CSS (Cascade Style Sheet) and JavaScript (Client-side only) but with the help of communicator layer, it can control and display real-time real-life objects from a modern web browser such as Google Chrome, Apple Safari, Microsoft Edge, Opera etc.

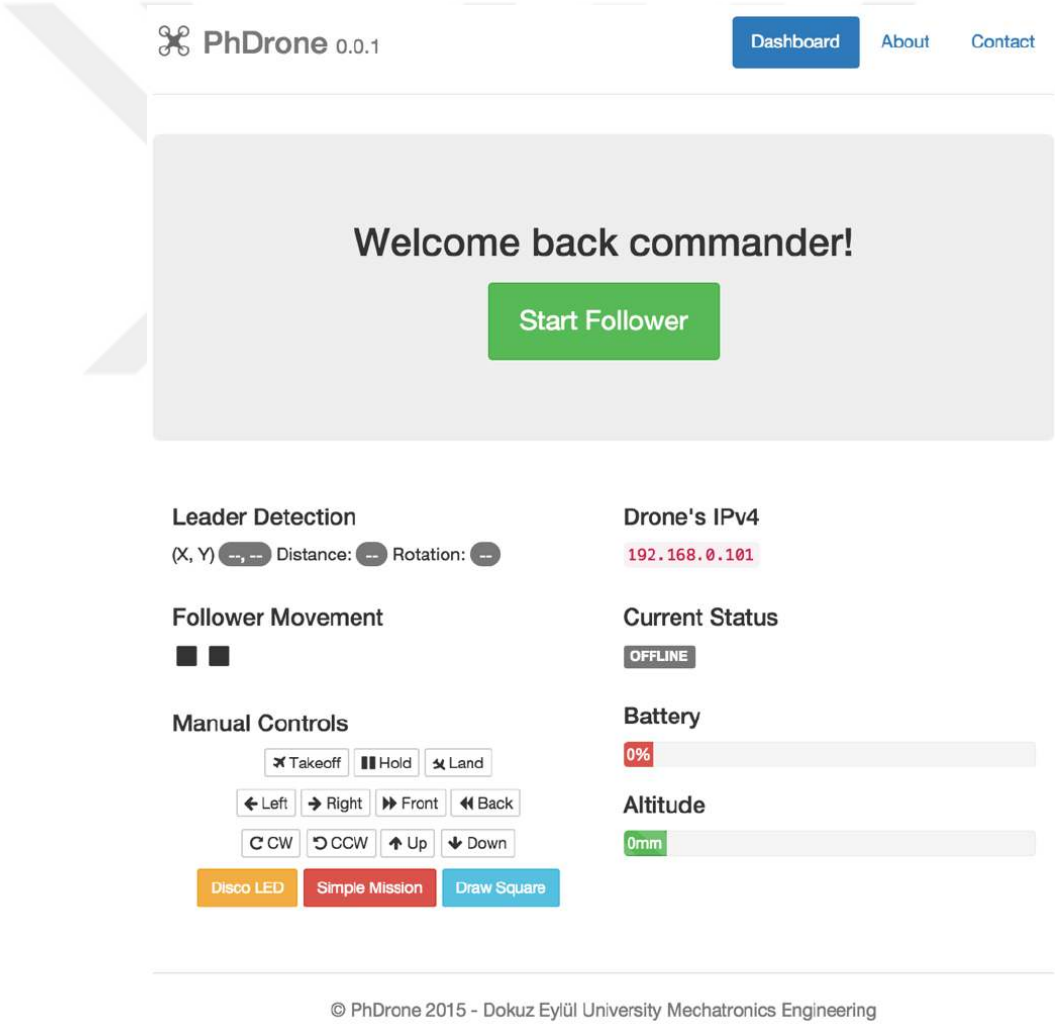


Figure 4.5 Web based graphical user interface

Last and most important layer is where whole system is relay on, multi robot or quadrotor controller. This layer's first mission is to extend the single vehicle controller to a group. This task is relatively simpler in Node.js than C++ or Python. Because the program and all data that is generated during the execution run in a single thread. So even the vehicles are defined independently like inserting a new item into an array, all the properties, variables and data can be easily shared without building a multi-threaded application and arranging points or constructing server-client structure.

Multi robot controller layer has also deals with PID controller of quadrotor. This controller can be used for solo or group flight. PID controller's structure is straightforward as seen in Figure 4.6 (Liang, 2014).

```
module.exports = PID;
function PID(kp, ki, kd) {
    'use strict';
    this.configure(kp, ki, kd);
    this.reset();
}

PID.prototype.configure = function(kp,ki,kd) {
    'use strict';
    this._kp = kp;
    this._ki = ki;
    this._kd = kd;
};

PID.prototype.reset = function() {
    'use strict';
    this._last_time = 0;
    this._last_error = Infinity;
    this._error_sum = 0;
};

PID.prototype.getCommand = function(e) {
    'use strict';
    // Compute dt in seconds
    var time = Date.now();
    var dt = (time - this._last_time) / 1000;

    var de = 0;
    if (this._last_time !== 0) {
        // Compute error derivation
        if (this._last_error < Infinity) {
            de = (e - this._last_error) / dt;
        }

        // Integrate error
        this._error_sum += e * dt;
    }

    // Update our trackers
    this._last_time = time;
    this._last_error = e;

    // Compute commands
    var command = this._kp * e
                + this._ki * this._error_sum
                + this._kd * de;

    return command;
};
```

Figure 4.6 PID controller example in Node.js

In order to support PID controller, de-facto standard extended Kalman filter (EKF) is also implemented and used. Taylor approximation to linearize the function is used.

Kalman filter is used in order to control drone more stable in order to track any image feature. Kalman filter has an advantage in predicting the next step from given state. In order to see whether this controller is suitable for autonomous flight controller, a basic Kalman filtering application is written by using OpenCV (Arnon, 2011). This program is called “Mouse Kalman” as shown Figure 4.7, which tracks mouse movement and predicts movement state when the given inputs are unregulated.

Figure 4.7 Mouse tracking with using Kalman Filter

The controller with Kalman filtering support 4 state (dynamics) parameters, 2D location, 2D velocity, 2 measurement parameters, 2D location of the tracked object. Predicted state (Equation 4.1), corrected state (Equation 4.2), prior error covariance (Equation 4.3), posteriori error covariance (Equation 4.4) and Kalman gain (Equation 4.5) is given as follows. Initialized state transition (A) (Equation 4.6), measurement (H) (Equation 4.7), posteriori error estimate covariance (P) (Equation 4.8), measurement noise covariance (R) (Equation 4.9) and process noise covariance (Q) matrixes are given as follows (Equation 4.10).

$$x'(k) = A \times x(k-1) + Bu(k-1) + w(k) \tag{4.1}$$

$$x(k) = x'(k) + K(k) \times \left(z'(k) - H \times x'(k)\right) \tag{4.2}$$

$$P'(k) = A\,P(k-1)A^T + Q \tag{4.3}$$

$$P(k) = (I\ - K\ \times H) \times P'(k) \tag{4.4}$$

$$K = P'(k) \times H^T \times (H \times P'(k) \times H^T + R(k))^{-1} \tag{4.5}$$

$$\mathrm{A} = \begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{4.6}$$

$$H = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{4.7}$$

$$P = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{4.8}$$

$$R = \begin{bmatrix} 0.1 & 0 & 0 & 0 \\ 0 & 0.1 & 0 & 0 \\ 0 & 0 & 0.1 & 0 \\ 0 & 0 & 0 & 0.1 \end{bmatrix} \tag{4.9}$$

$$Q = \begin{bmatrix} 0.00001 & 0 & 0 & 0 \\ 0 & 0.00001 & 0 & 0 \\ 0 & 0 & 0.00001 & 0 \\ 0 & 0 & 0 & 0.00001 \end{bmatrix} \tag{4.10}$$

With all sub layers which run on a standard client PC and connect the flight group via Wi-Fi network is essential (Keskin, 2013).

4.2 Features and Usage

In this thesis study, both programmer and user-friendly approach was followed. Mission assignment is a simple task for drones. Pre-defined maneuvers are given in the executed code for each drone IP such as Table 4.1. Node.JS application starts to send relevant AT command set for all connected IPs without waiting any task to finish. And, all the maneuvers are logged in a separate text file. It is also possible to construct

real time dashboard for plotting these data with Node.js or logs can be used later in order to check accuracy or execution steps.

Table 4.1 Mission assignment code block

`var df = require('dateformat')`
`  , autonomy = require('../')`
`  , arDrone = require('ar-drone')`
`  , arDroneConstants = require('ar-drone/lib/constants')`
`  , mission  = autonomy.createMission({ip:'192.168.2.101'})`
`  ;`
	Library initilizations and client (drone) IP declaration

`function navdata_option_mask(c) {`
`  return 1 << c;`
`}`
 	
`// From the SDK.`
`var navdata_options = (`
`  navdata_option_mask(arDroneConstants.options.DEMO)`
`  | navdata_option_mask(arDroneConstants.options.VISION_DETECT)`
`  | navdata_option_mask(arDroneConstants.options.MAGNETO)`
`  | navdata_option_mask(arDroneConstants.options.WIFI)`
`);`
	Activation of some hardware/ firmware features

`// Land on ctrl-c`
`var exiting = false;`
`process.on('SIGINT', function() {`
`  if (exiting) {`
`  process.exit(0);`
`  } else {`
`  console.log('Got SIGINT. Landing, press Control-C again to force exit.');`
`  exiting = true;`
`  mission.control().disable();`
`  mission.client().land(function() {`
`  process.exit(0);`
`  });`
`  }`
`});`
	If program cancels, make sure to send land signal

`// Connect and configure the drone`
`mission.client().config('general:navdata_demo', true);`
`mission.client().config('general:navdata_options', navdata_options);`
`mission.client().config('video:video_channel', 1);`
`mission.client().config('detect:detect_type', 12);`
`// Flight record`
`mission.log("mission-" + df(new Date(), "yyyy-mm-dd_hh-MM-ss") + ".txt");`
	Configuration of flight parameters and flight record

`// Execute mission`
`mission.run(function (err, result) {`
`  if (err) {`
`  console.trace("Oops, Error: %s", err.message);`
`  mission.client().stop();`
`  mission.client().land();`
`  } else {`
`  console.log("We are done!");`
`  process.exit(0);`
`  }`
`});`
	Execute the given mission unless any error occurs

A web based asynchronous application is ready to control group of drones. All drones now have a built-in DHCP server/client schema. With the help of this feature, a regular computer with only one wireless antenna can connect all drone in range. The network connection for a drone protocol starts with checking if there is mobile ad hoc network called “PhDrone” already exists. When yes, drone request a new, unique IP

address for itself. If no, drone creates the ad-hoc network by itself. This protocol also checks whether DHCP server is alive. If yes, nothing changes and continue connection as always. If no, first drone encourages this network problem, turns itself in to server to keep connection alive. At this stage, all IP address keep same, only the gateway is updated.

All the experiments are controlled, executed and reported by web application which is developed in this thesis study as shown in Figure 4.8. All paths for each drone can be assigned and previewed on web page. Paths are generated by lattice path approach. A lattice path is a walk in a lattice in some d-dimensional Euclidean space (Banderier,75 2002). Formally, a lattice path P is a sequence $P = (P_0, P_1, ..., P_n)$ of points P_n in Z^d. Figure 4.8 shows the lattice path ((0,0), (1,1), (2,1), (3,1), (3,2), (4,3)). The point P_0 is called the starting point and P_l is called the *end point* of *P*. The vectors P_0P_1, P_1P_2, ..., $P_{n-1}P_n$ are called the *steps* of *P*.

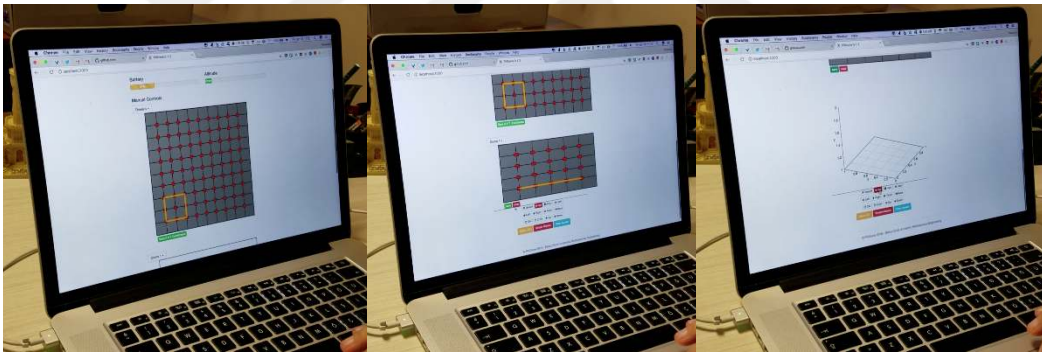


Figure 4.8 Making experiments through web application (Personal archive, 2017)

4.3 Network System Design with Self-Healing Behavior

Quadrotor has two main network design criteria. First one is to create a router with DHCP capability if there is no control network in the range. This is controlled and validated by querying SSID value in the current environment. The second one, if there is already our network is available at the range, join this network. With these features and a background daemon on quadrotor's embedded system, we can manage communication between computer and the UAVs without any interruption. Even the worst-case scenario, when main quadrotor that constructs the wireless network fails

because of empty battery or an instant crash, the background worker (daemon) of any other quadrotor can start the same wireless network. At that instant, the UAVs' missions can continue because of TCP/IP structure. If valid TCP packages are already received by quadrotors, the mission will continue to execute. In this scenario, there is no need of any centralized computing or controlling structure. All the maneuvers can be done by using on-board controller.

Network controller algorithm concept is determined in Table 4.2 by pseudocode block. This controller consumes CPU load in each drone which is around 7%. But by the help of it, we can keep the network stable and continuous. In the client side, web application is the daemon itself. It tracks and adjust network changes in the background with all other functions so that CPU load effect is negligible.

Table 4.2 Network controller daemon both for each drone and client

```
shared_const gateway, subnet
const static_ip forEach drones
const static_ip forEach clients

async.forEach drones
    if this.drone.ip is not router.ip and router.ip alive
           release router.ip
           release access_point
           join access_point
    if this.drone.ip is not router.ip and router.ip not alive
           create access_point
           publish new this.drone.ip as router.ip

async.forEach clients
    if router.ip is not alive
           release router.ip
           release access_point
           join access_point
```

CHAPTER FIVE
EXPERIMENTS AND RESULTS

5.1 Experimental Environment Overview

The experimental study consists of three steps. It is aimed to show that multiple aerial vehicles can be operated from a web application. In the first step, single quadrotor is controlled and analyzed. In the second step, multiple quadrotors are controlled. But each of them has different and discrete missions. In the last step, joint tasks are given to the multiple quadrotor in order to execute missions with constructing a formation.

All the experiments done in an indoor environment. Flight tests are performed in two different floor conditions; wooden (office floor) and epoxy coated concrete (warehouse floor) (Figure 5.1). Because of obstacle avoidance, which is not the objective of this study, generally large, clear workspaces are selected in order to perform flights.

Figure 5.1 Office floor (left) vs. Warehouse floor (right) (Personal archive, 2017)

In order to perform any flight, only a PC and quadrotors are needed. There is no need of any other router or access point to build wireless communication network. But if range extension is needed, additional hardware can be used.

Once the PC is connected to quadrotor's network, mission planner interface is ready to connect with a web browser under http://localhost:3000

5.2 Single Flight Missions

First mission is to draw a square on the sky at 1m and 2m altitude. These missions are executed with UAV 03 and their yaw angle graphs are generated from its flight record as seen in Figure 5.2. Path execution of this drone can be seen in Figure 5.3 and 5.4 on 3D coordinate system in real world. Quadrotor controls itself in order to stay in the given path by state estimation with the help of Extended Kalman Filter.

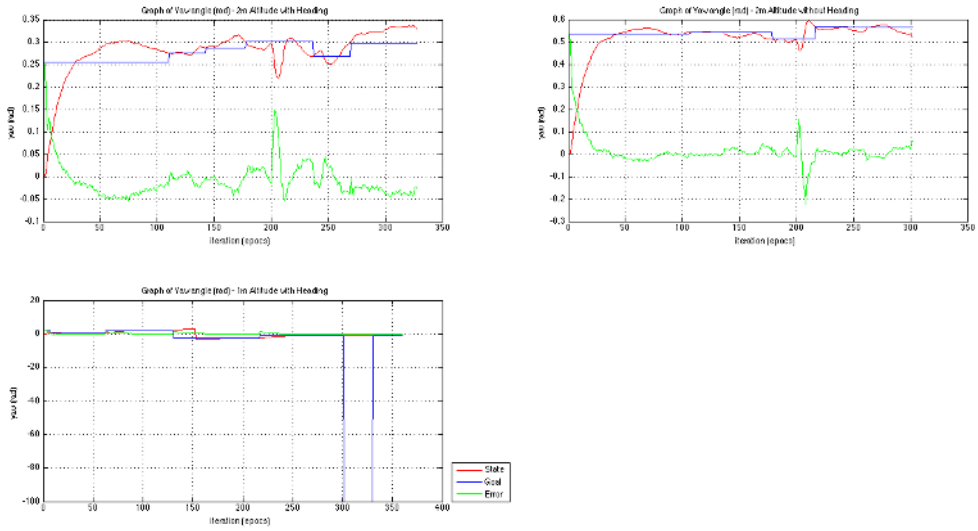


Figure 5.2 Yaw angle state (red), goal (blue), error (green) in 3 given missions; 2m altitude with heading (top-left), 2m altitude without heading (top-right), 1m altitude with heading (bottom-left)

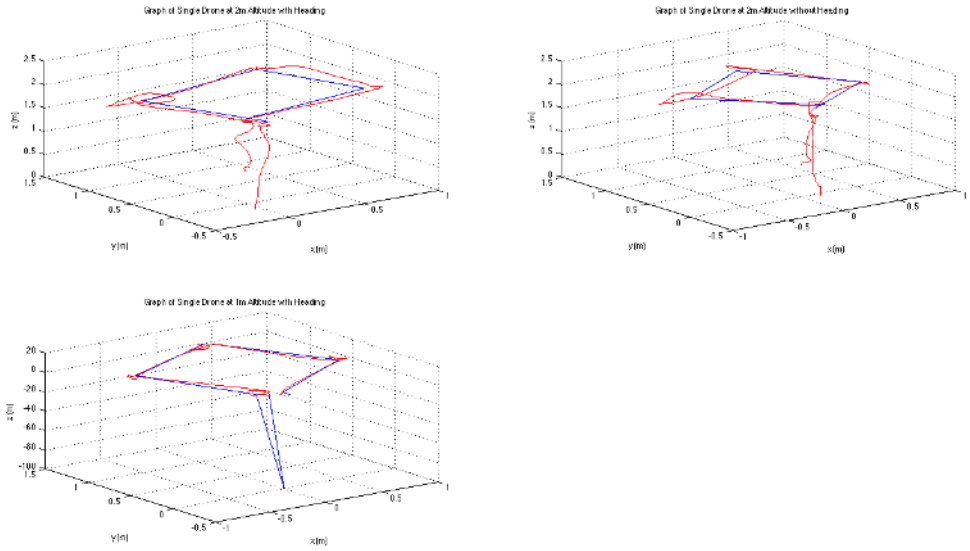


Figure 5.3 Square path mission execution; state (red), goal (blue) in 3 given missions; 2m altitudes with heading (top-left), 2m altitude without heading (top-right), 1m altitude with heading (bottom-left)

This mission's lattice path is given in Equation 5.1

$$Map_{P1}\begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 \end{bmatrix} \tag{5.1}$$

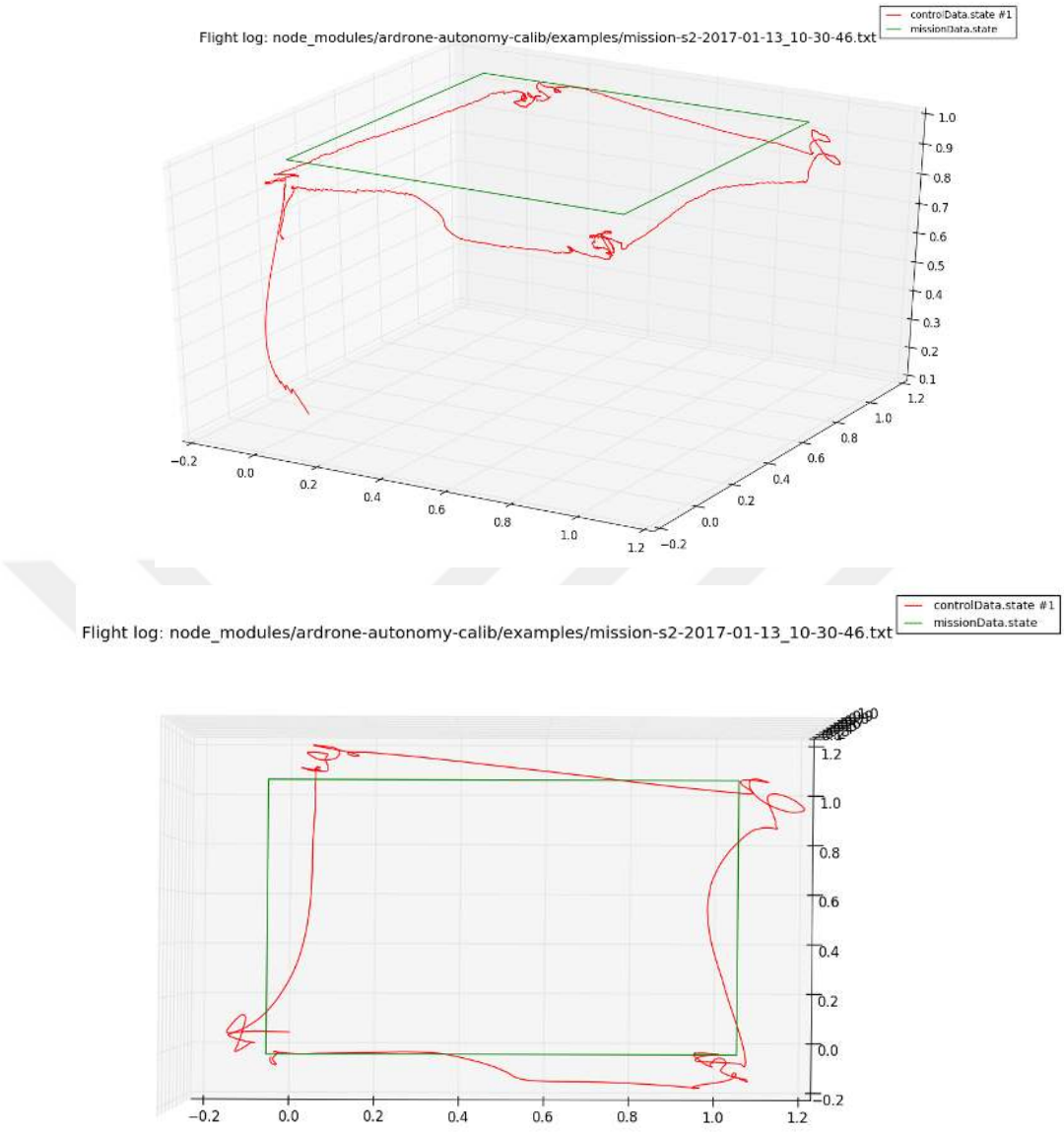


Figure 5.4 Experiment # 1 – Drawing a 1m x 1m square at 1m altitude without fixed heading

In order to analyze flight logs a Python script may also be useful (Appendix 2).

As seen in Figure 5.4 quadrotor travels of the path at all edges. In the first experiment, control commands do not keep drone's heading. At all movement, drone's front is turning in clockwise or counter clock-wise direction. But on-board PID controller, web application server's PID controller with Kalman filter and on-board optical flow feedback from bottom camera tries to return the relevant corner.

In Figure 5.5 quadrotor miscalculates its initial position so that about a 0.5m offset is produced. But at the corners, the maneuvers' error is less than the first experiment which can easily detect from plots.

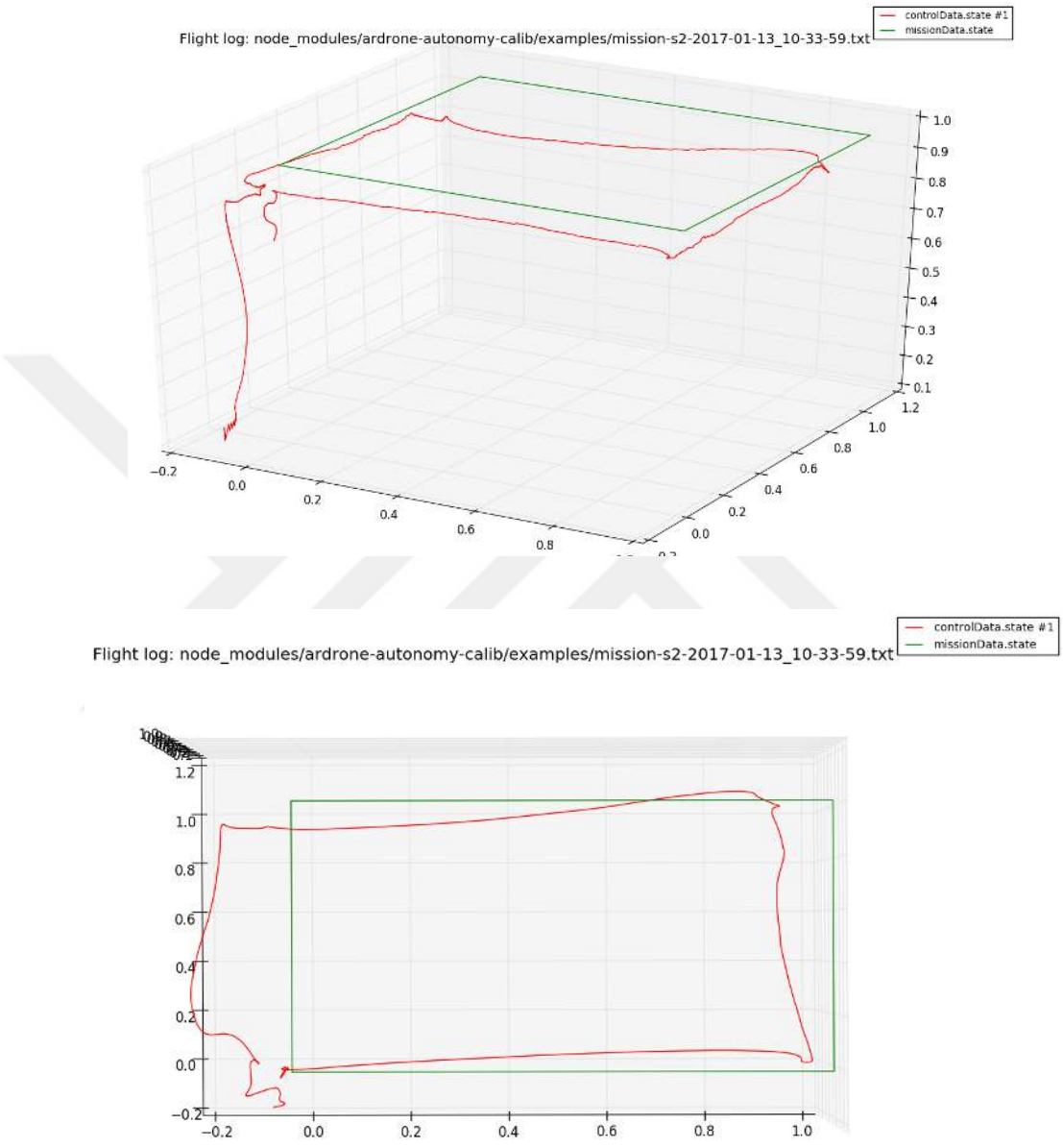


Figure 5.5 Experiment # 2 – Drawing a 1m x 1m square at 1m altitude with fix heading

In Figure 5.6 after calibrating internal measuring unit and vertical camera parameters, the same quadrotor starts to follow the mission path. Optical flow measurement is crucial in our experiments. Because in our case studies, we do not use any global locating or positioning sensors, algorithms and method. Optical flow, measures displacement (in our case) by relative motion between an observer and a scene.

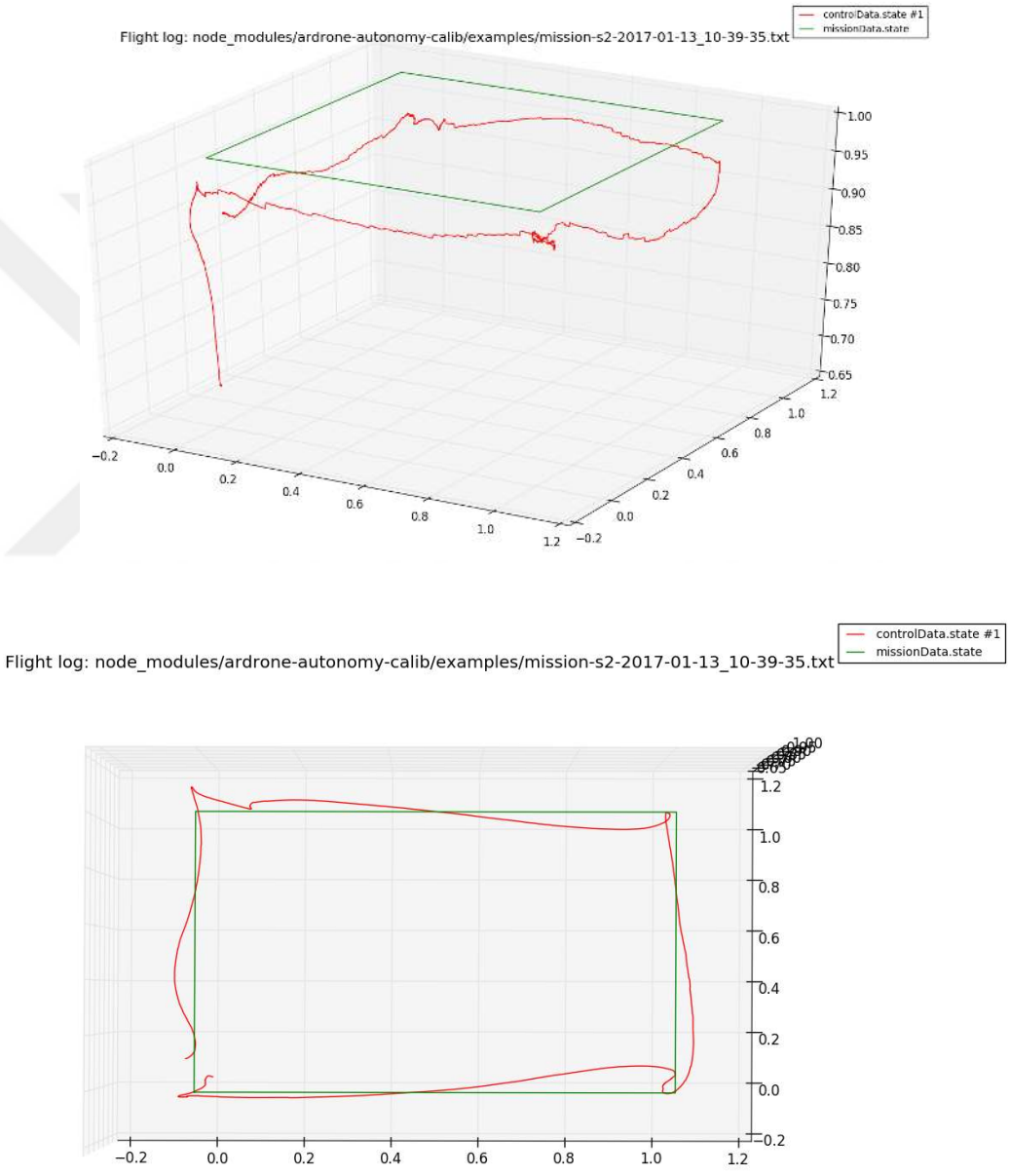


Figure 5.6 Experiment # 3 – Drawing a 1m x 1m square at 1m altitude with fixed heading

Drone has become more stable during flight given in Figure 5.7. There are still issues at the landing stage. Main reason of this situation, feedback from vertical camera becomes unreliable when field of view decreases.

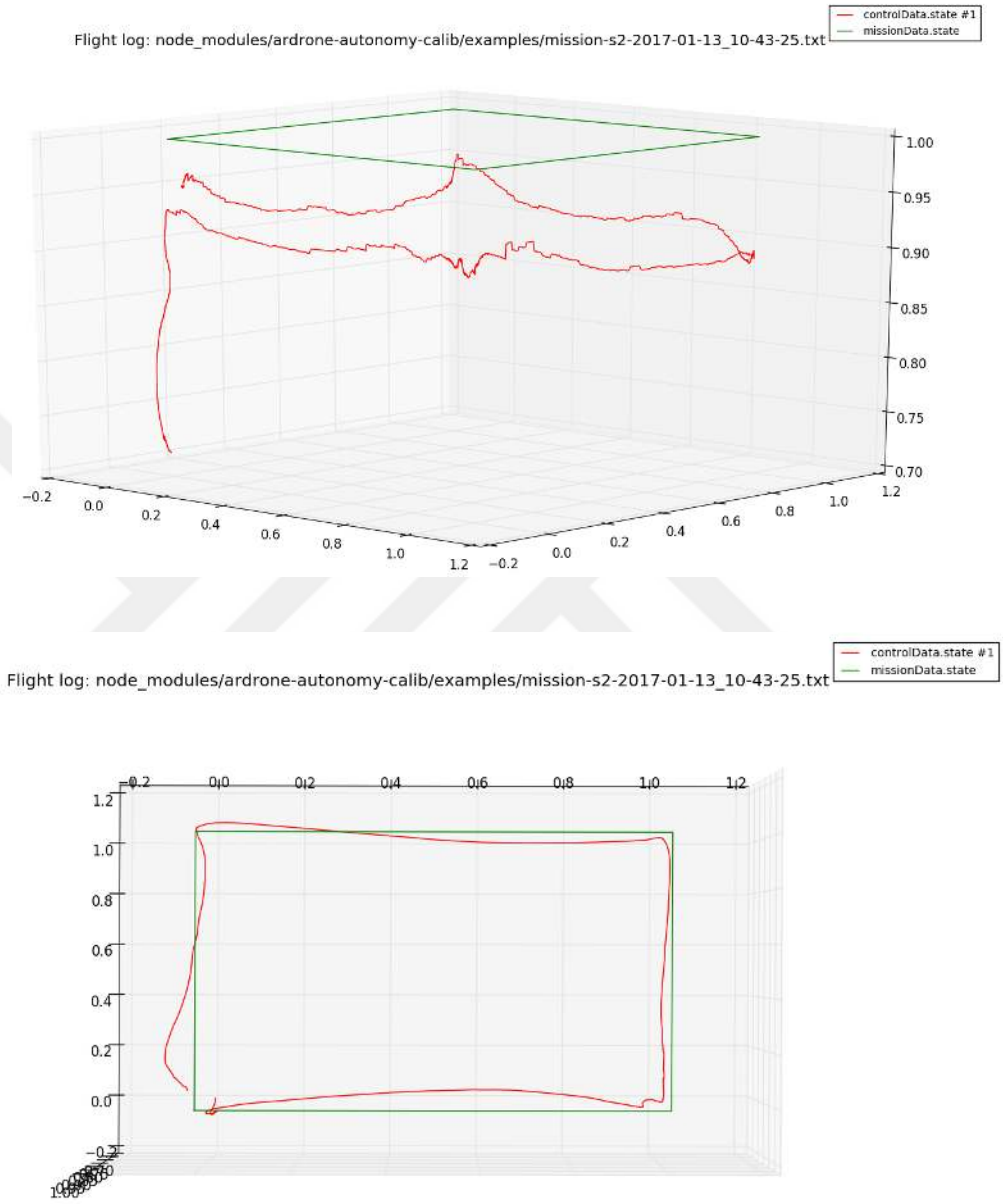


Figure 5.7 Experiment # 4 – Drawing a 1m x 1m square at 1m altitude with fixed heading

Triangle shaped mission path is also studied (Figure 5.8). The lattice map is defined as given in Equation 5.2.

$$Map_{P4b}\begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 1 & 0.5 & 0 \\ 1 & 1 & 1 & 1 \end{bmatrix} \tag{5.2}$$

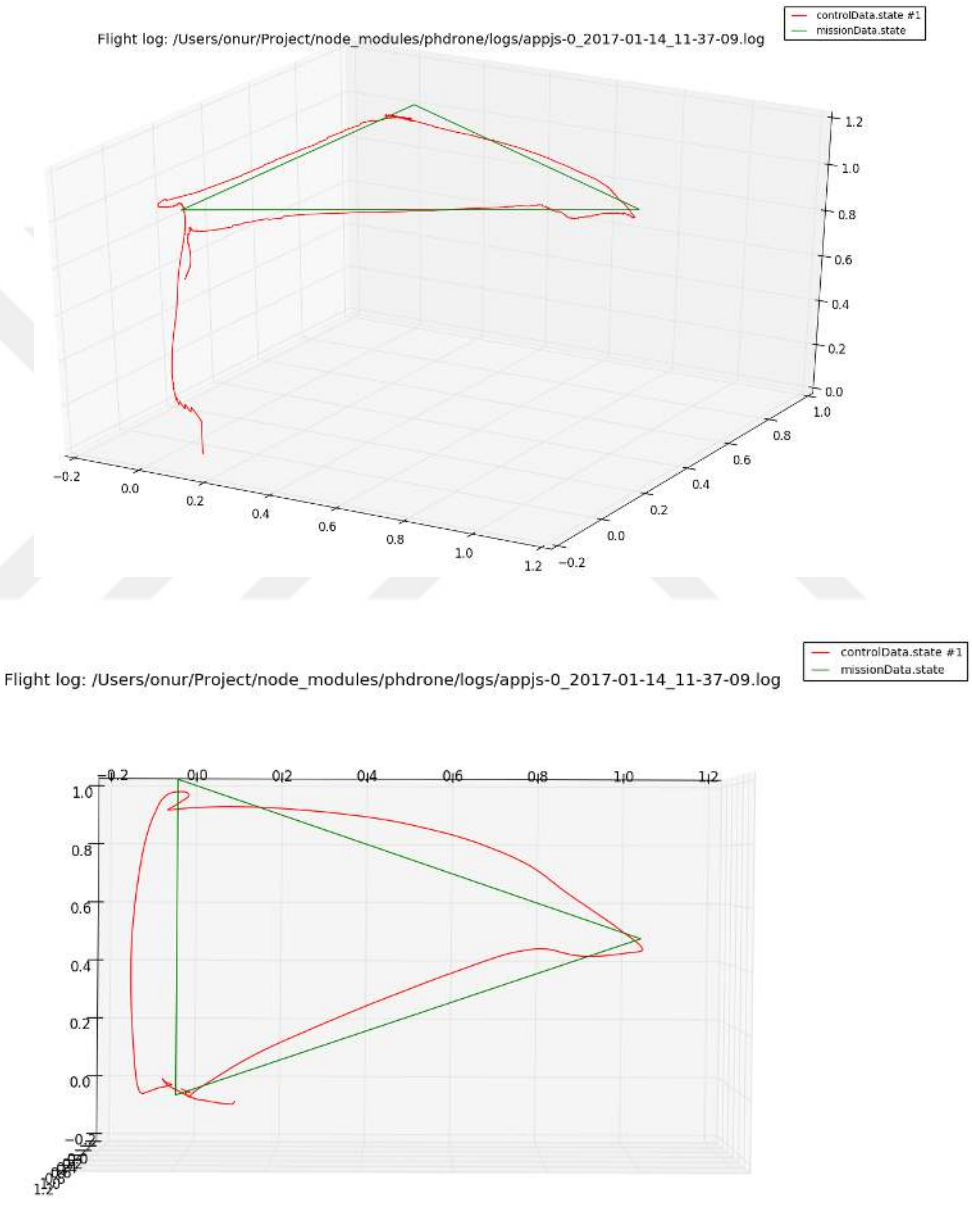


Figure 5.8 Experiment # 5 Drawing an isosceles triangle at 1m altitude with fixed heading

In the sixth experiment, “L” like shaped is defined and executed (Figure 5.9). Although, warehouse’s floor disturbs displacement calculations, straight movements are better than cross movement. This experiment’s lattice map is given in Equation 5.3

$$Map_{P2}\begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0.5 & 0.5 & 1 & 1 & 0 \\ 0 & 1 & 1.25 & 0.5 & 0.5 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix} \tag{5.3}$$

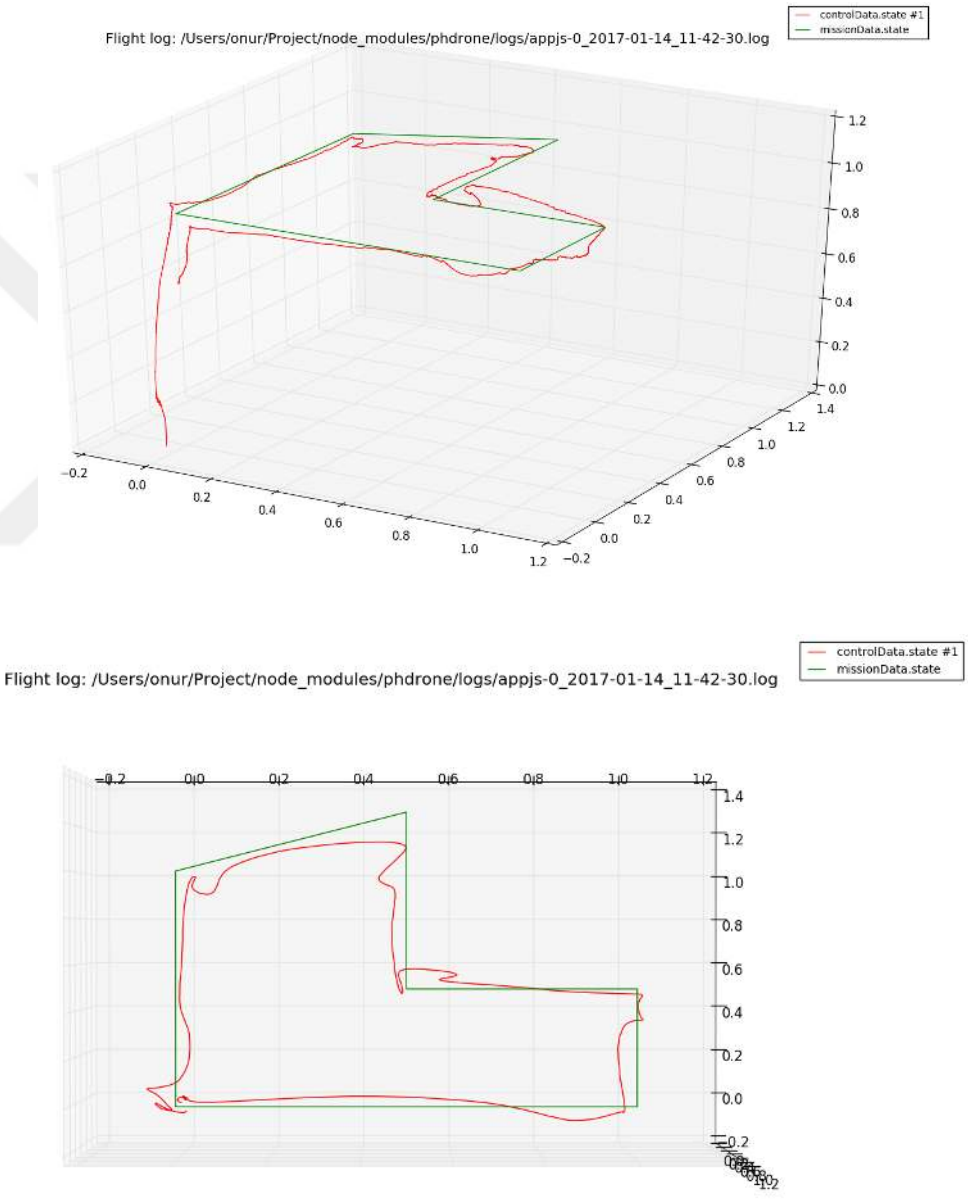


Figure 5.9 Experiment # 6 Drawing “L” shaped mission 1m altitude with fixed heading

We also run some real case scenarios, such as taking off, going to a specific avenue in a large indoor facility (Figure 5.10). So, we create a flag like shape in experiment # 7.

$$Map_{P10}\begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} 0 & 0 & 2 & 1 & 1 & 0 \\ 0 & 4 & 4 & 4 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix} \tag{5.4}$$

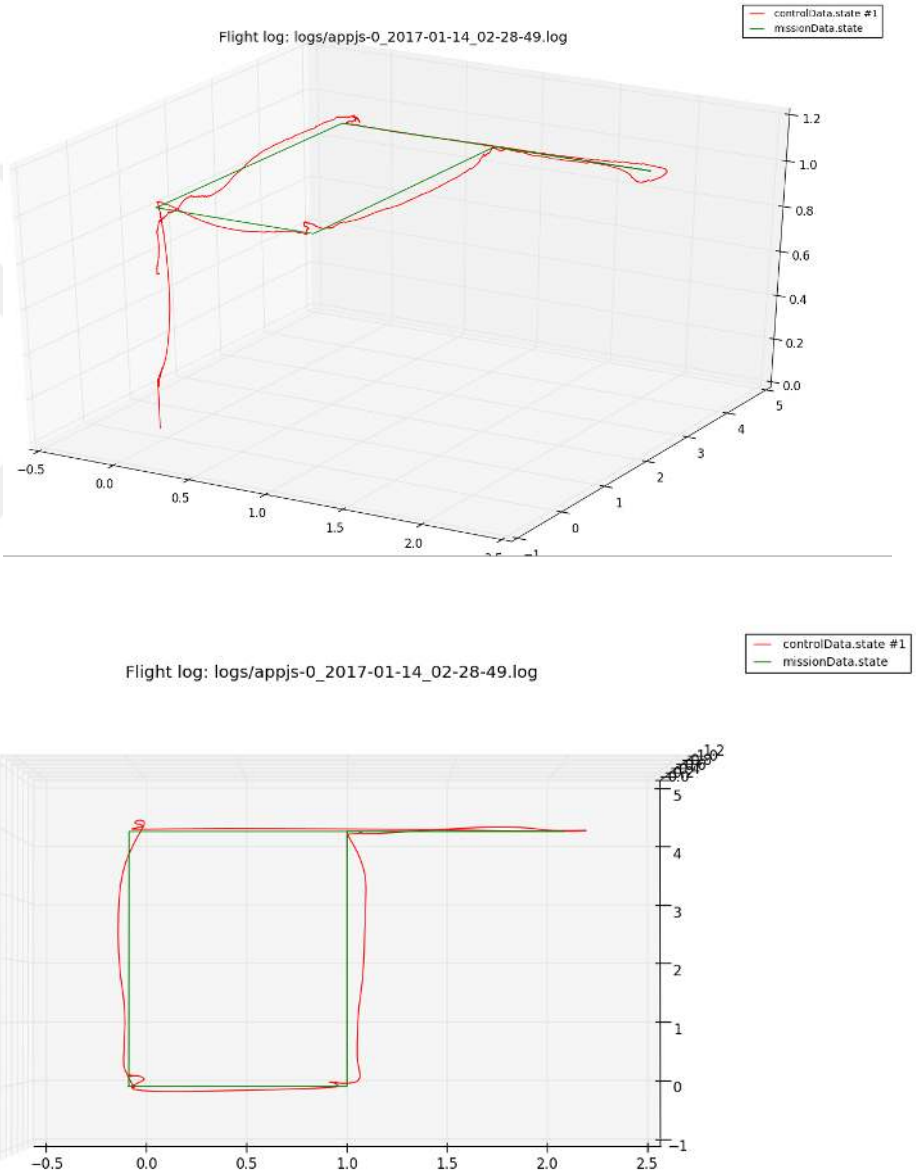


Figure 5.10 Experiment # 7 Flag like path at 1m altitude with fixed heading

On the “mountain” like shaped mission in experiment # 8, another variable can be used in order to stay on the path which is piece wise movement (Equation 5.5). By this strategy, we send short commands, so we let drone to make more calculation in order to correct its last state (Figure 5.11).

$$Map_{P7}\begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} 0 & 0 & 1 & 1.5 & 2.5 & 0.5 & 0.5 & 0 \\ 0 & 1 & 2 & 2 & 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix} \tag{5.5}$$

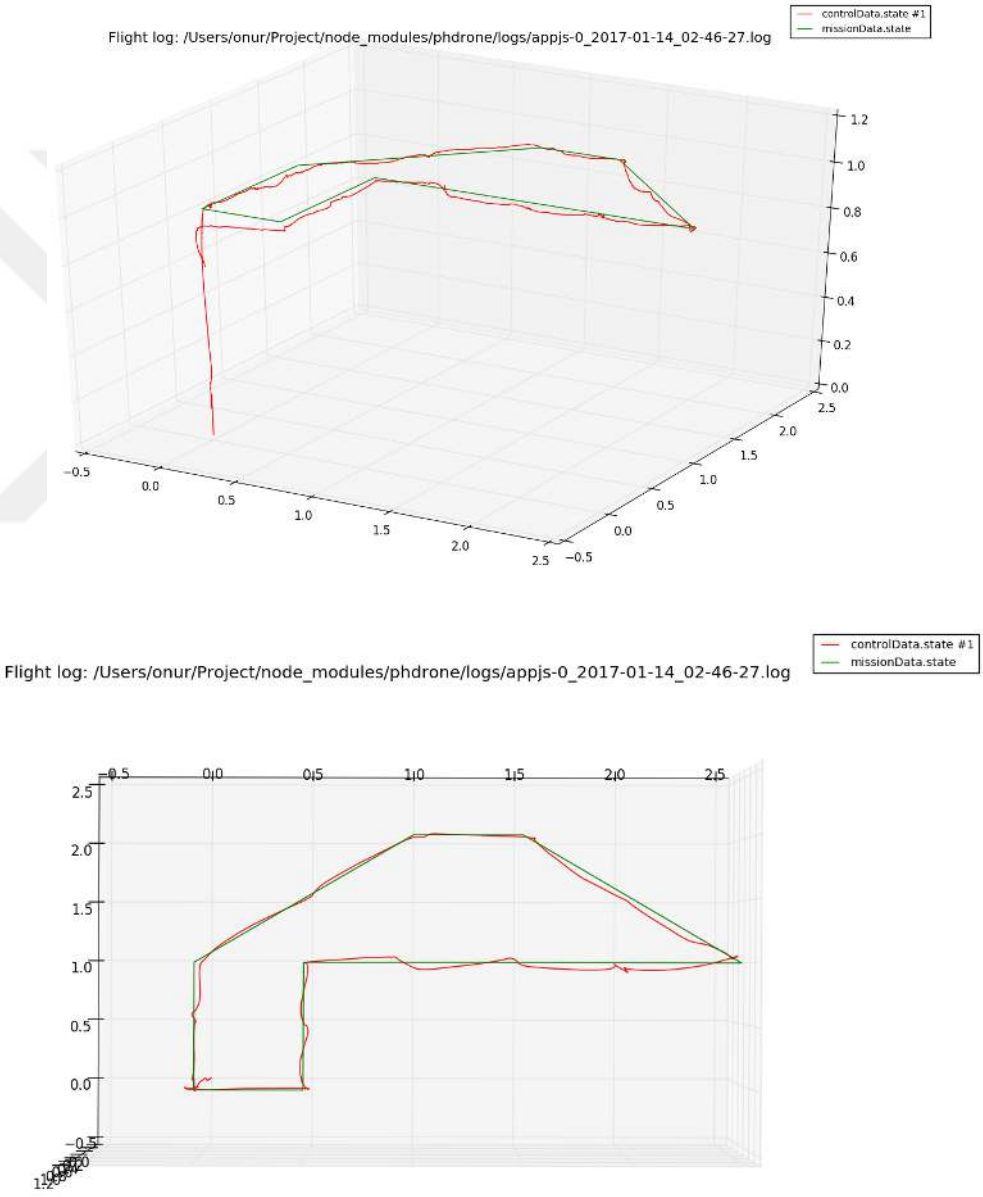


Figure 5.11 Experiment # 8 Piece wise movement at 1m altitude with fixed heading

In order to support our experiment # 7, we also go inside an avenue in the warehouse, and also go to another shelf and then come back to our origin position in 9th experiment (Equation 5.6). We prove that, on a straight movement, our controllers work acceptable even by GPS systems (Figure 5.12).

$$Map_{P6}\begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} 0 & 0 & 2.5 & 2.5 & 0.5 & 0.5 & 0 \\ 0 & 2 & 2 & 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix} \tag{5.6}$$

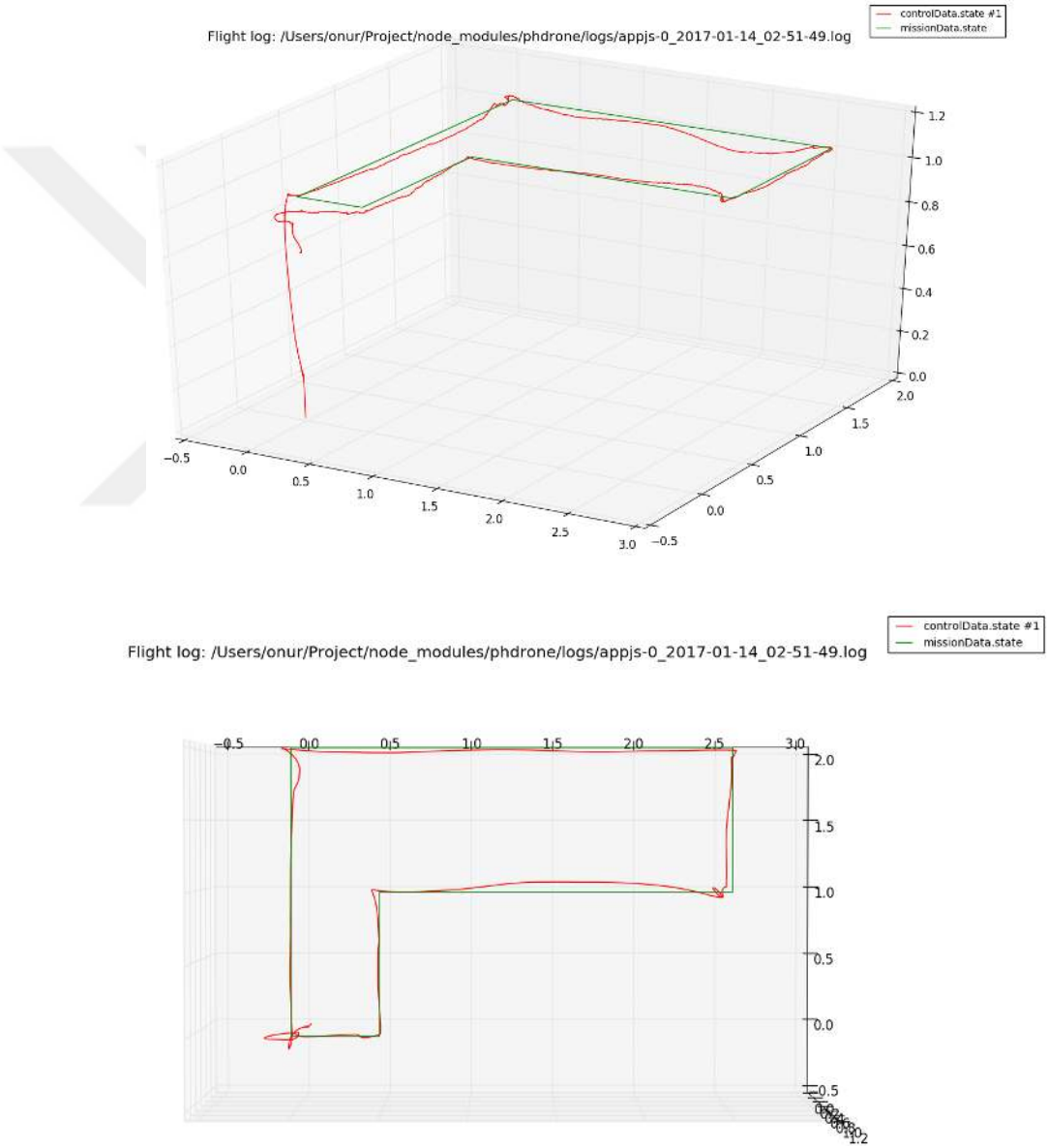


Figure 5.12 Experiment # 9 “gun” like path at 1m altitude with fixed heading

After getting results like experiment # 9, we want to focus cross movements and see in a basic map which can be seen in Figure 5.13.

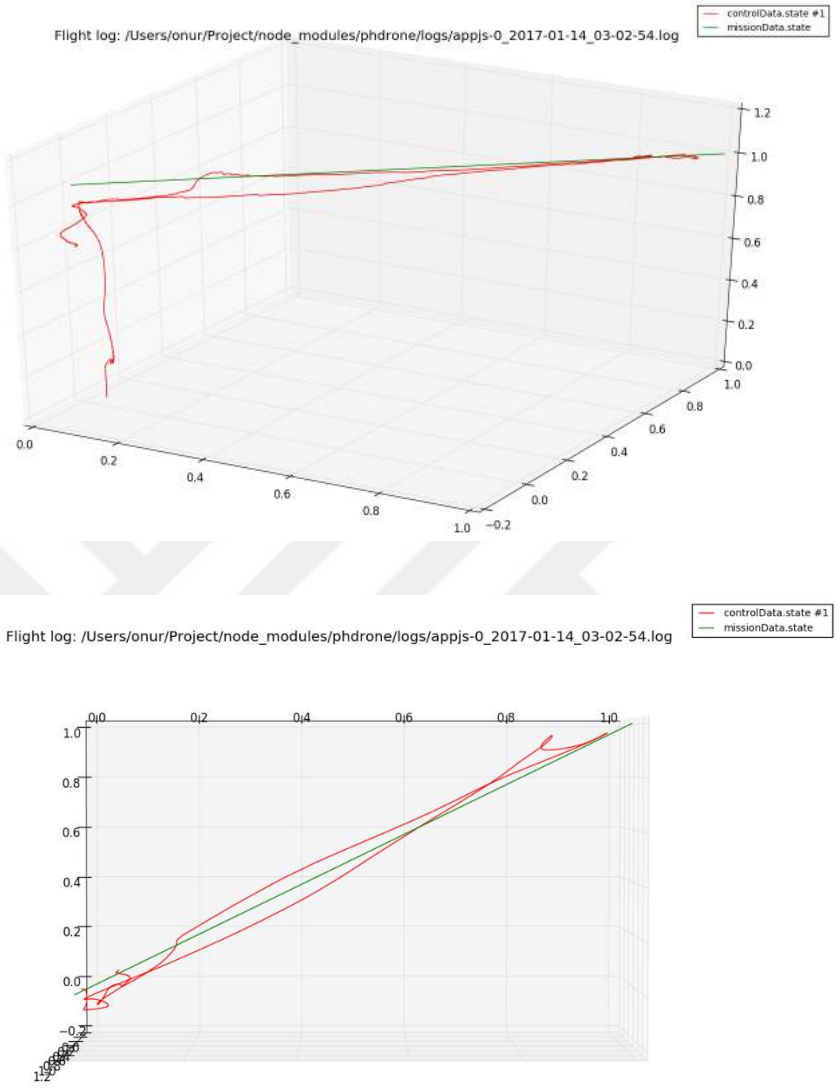


Figure 5.13 Experiment # 10 “slash” like path to examine position errors in cross movement

But when we look R^2 of the mission path, we realized that we almost approach 1.0, our real R^2 value is 0.99277 as seen in Figure 5.14. But in real time systems especially aerial control systems, frequency of the position and/or displacement sensor has to be more than 60Hz. In our case, our drone's vertical camera can get 60Hz frames but processing it on-board has some limitations.

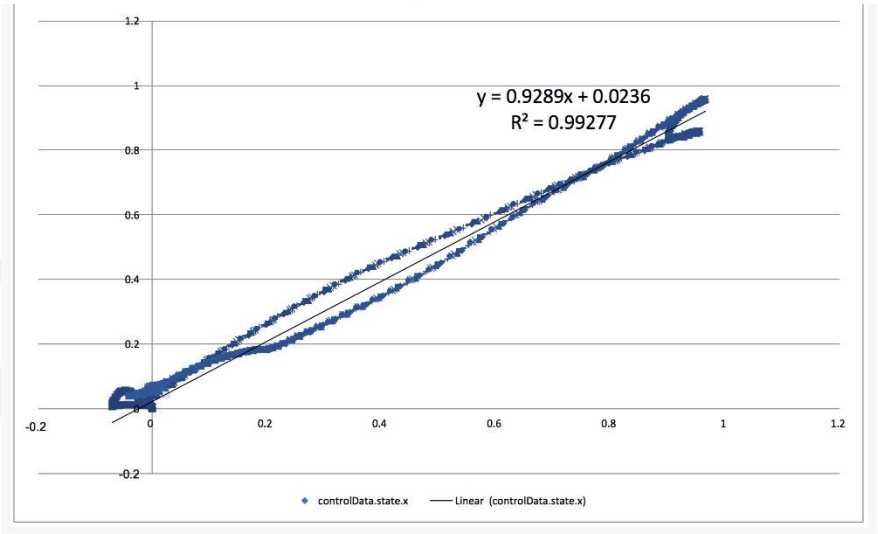


Figure 5.14 Simple linear regression and R^2 calculation for basic "slash" like mission

5.3 Independent Multiple Flight Missions

Second stage of the experiments are flying multiple quadrotors. We drive 2 (two) drones which have 20cm distance at the resting point. Although their air flow interfered with each other, mission is successful (Figure 5.15).

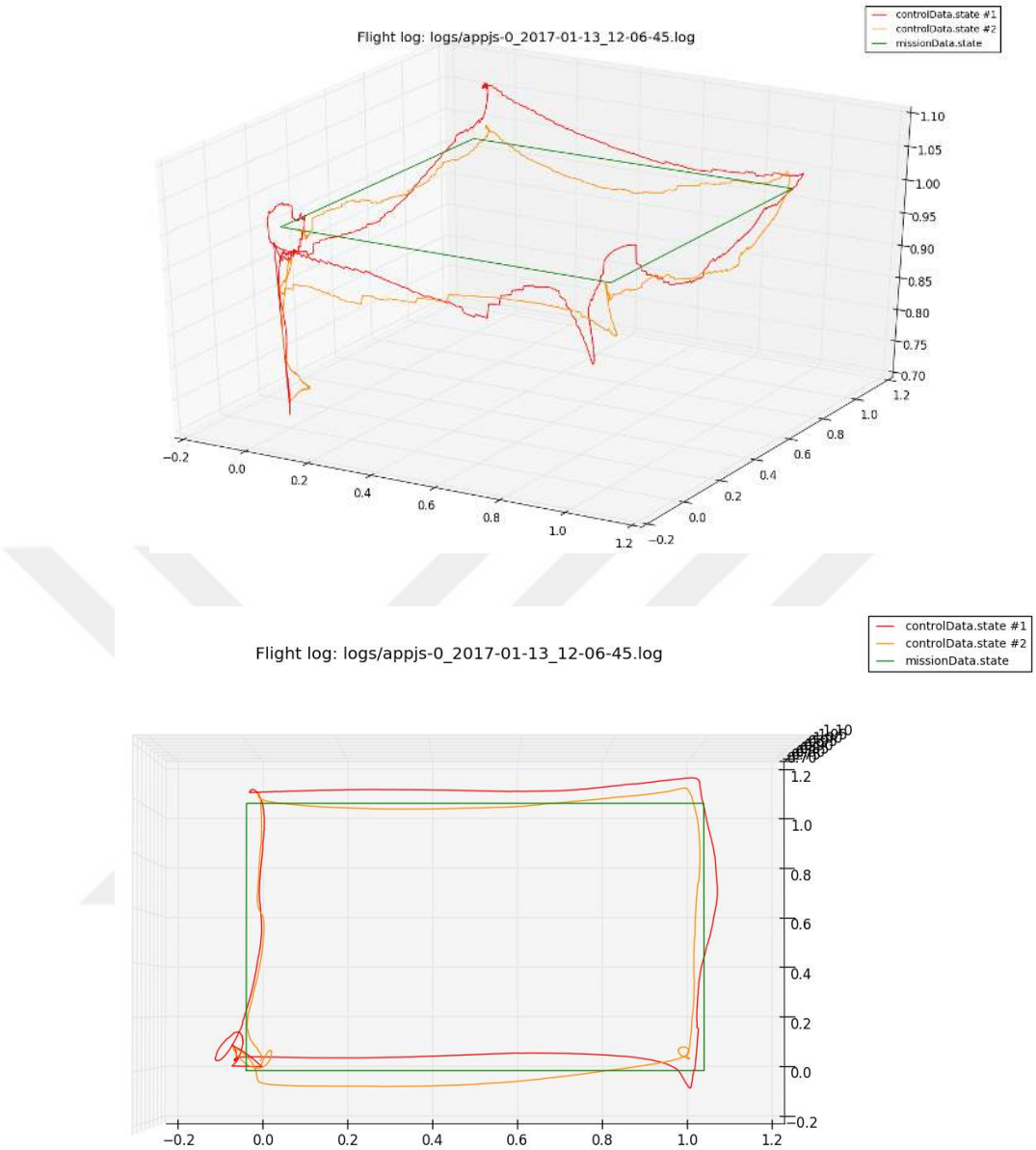


Figure 5.15 Experiment # 11 – 2 (two) drones are drawing a 1m x 1m square at 1m altitude with fixed heading

First experimental mission path at warehouse is to follow a right triangle as shown in Figure 5.16. The lattice map is defined as given in Equation 5.7.

$$Map_{P4b}\begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 1 & 1 & 1 \end{bmatrix} \tag{5.7}$$

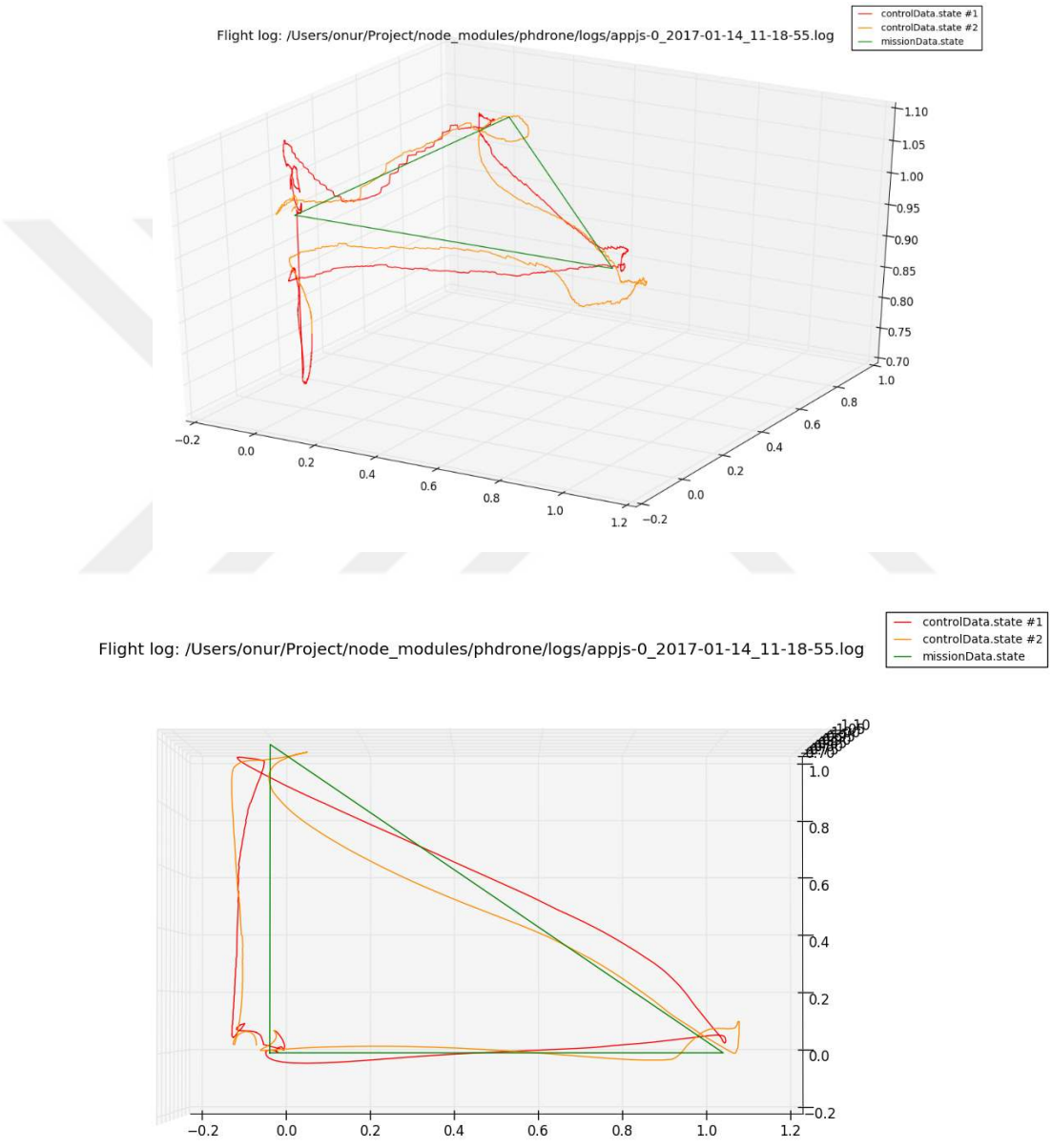


Figure 5.16 Experiment # 12 Two drones with altitude difference in flight, right triangle shaped path mission at a warehouse

In order to try the limits of our study, we flight a drone with more complicated "mountain" like path in the next experiment (Equation 5.8). In order to see effects of the concreate floor, we also cover some places with cardboards. This experiment shows that we still have larger errors while making cross movements (Figure 5.17). But, if the floor is not reflective and has no pattern, optical flow algorithm can correct the path during flight. This correction can be seen from P_1 (0.5, 1.5) to P_2 (1.5, 2.0) as seen in Figure 5.17.

$$Map_{P8}\begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} 0 & 0.5 & 1.5 & 2.5 & 0.5 & 0.5 & 0 \\ 0 & 1.5 & 2 & 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix} \tag{5.8}$$

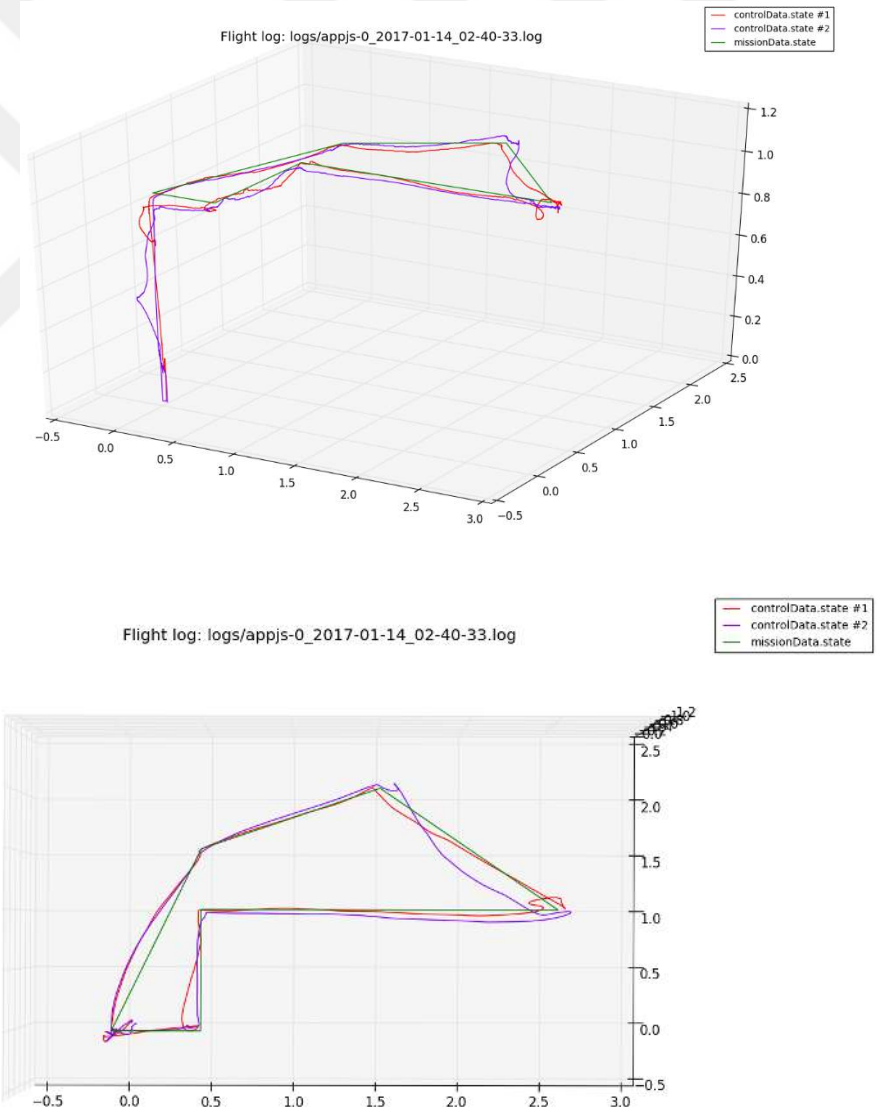


Figure 5.17 Experiment # 13 "mountain" like path at 1m altitude with fixed heading

5.4 Swarm Flight Missions

In order to keep formation with using lattice path we construct a world and body frame line given in Figure 5.18. All measurements are referred to the world frame (W), corresponding to axes W_x, W_y and W_z. The origin of the formation body frame (B) have axes B_x, B_y and B_z. Body frame was chosen to coincide with the centroid of the triangle formation shape.

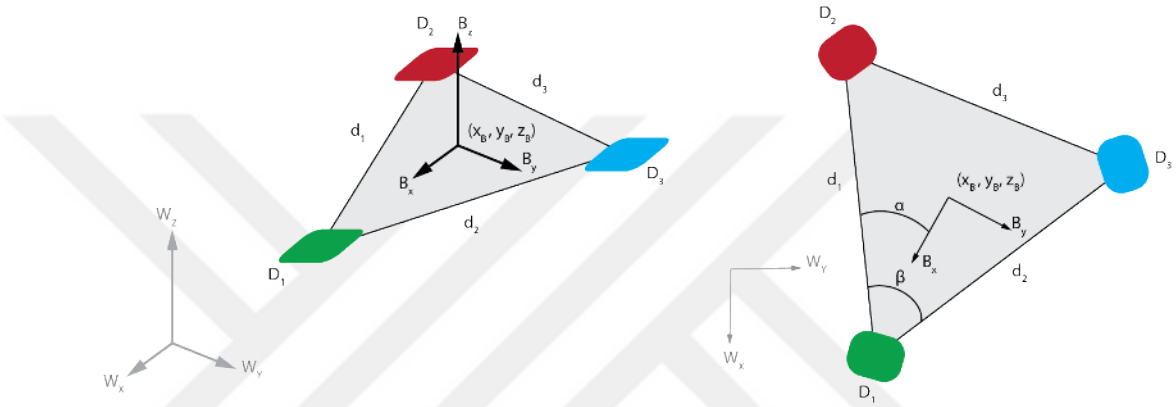


Figure 5.18 Perspective and top view of triangle formation, side by side

The centroid of formation triangle, P_B and shape matrix S_B is constructed with the following Equations.

$$P_B = \begin{bmatrix} x_B \\ y_B \\ z_B \end{bmatrix} = \begin{bmatrix} \dfrac{x_1 + x_2 + x_3}{3} \\ \dfrac{y_1 + y_2 + y_3}{3} \\ \dfrac{z_1 + z_2 + z_3}{3} \end{bmatrix} \tag{5.9}$$

$$S_B = \begin{bmatrix} d_1 \\ d_2 \\ \beta_B \end{bmatrix} = \begin{bmatrix} \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2 + (z_1 - z_2)^2} \\ \sqrt{(x_1 - x_3)^2 + (y_1 - y_3)^2 + (z_1 - z_3)^2} \\ \cos^{-1} \dfrac{d_1^2 + d_2^2 - d_3^2}{2d_1\, d_2} \end{bmatrix} \tag{5.10}$$

$$d_3 = \sqrt{(x_1 - x_3)^2 + (y_1 - y_3)^2 + (z_1 - z_3)^2} \tag{5.11}$$

$$B_x = D_1 - P_B = \begin{bmatrix} x_1 \\ y_1 \\ z_1 \end{bmatrix} - \begin{bmatrix} x_B \\ y_B \\ z_B \end{bmatrix} \tag{5.12}$$

$$B_z = (D_1 - D_2) \times (D_1 - D_3) \tag{5.13}$$

$$B_y = B_x \times B_z \tag{5.14}$$

$$Roll\ \phi_B = \tan^{-1} \frac{B_{x(y)}}{B_{y(x)}} \tag{5.15}$$

$$Pitch\ \theta_B = \sin^{-1} B_{x(z)} \tag{5.16}$$

$$Yaw\ \psi_B = \tan^{-1} \frac{B_{y(z)}}{B_{z(z)}} \tag{5.17}$$

For this experiment, assigned task is to perform a sequential flight in a triangular shape. Quadrotors are started the mission on each of the edges, when the i^{th} quadrotor moves i^{th} + 1 position, i^{th} +2 quadrotor moves i^{th} position (Figure 5.20 – 5.24). The mission is just like a puss-in-the corner game or like vultures draw circles in the air. In this experiment, we show that a quadrotor can trigger another quadrotor when it finishes its job as shown in Figure 5.19.

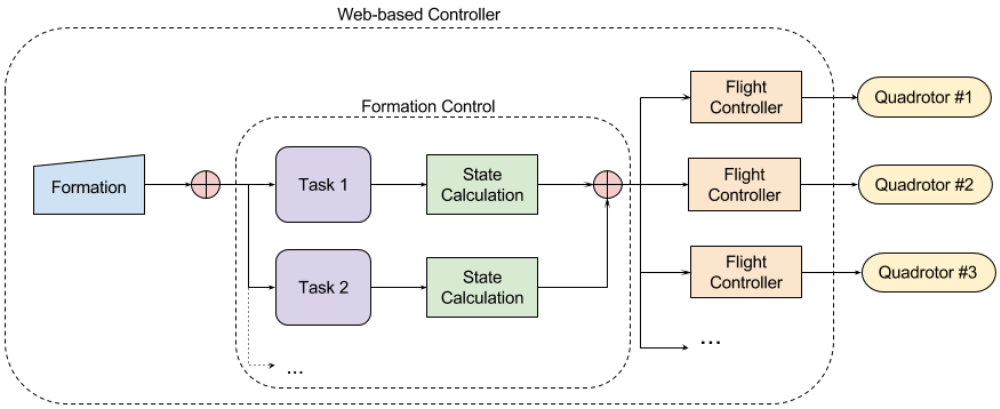


Figure 5.19 Control loop of a group flight

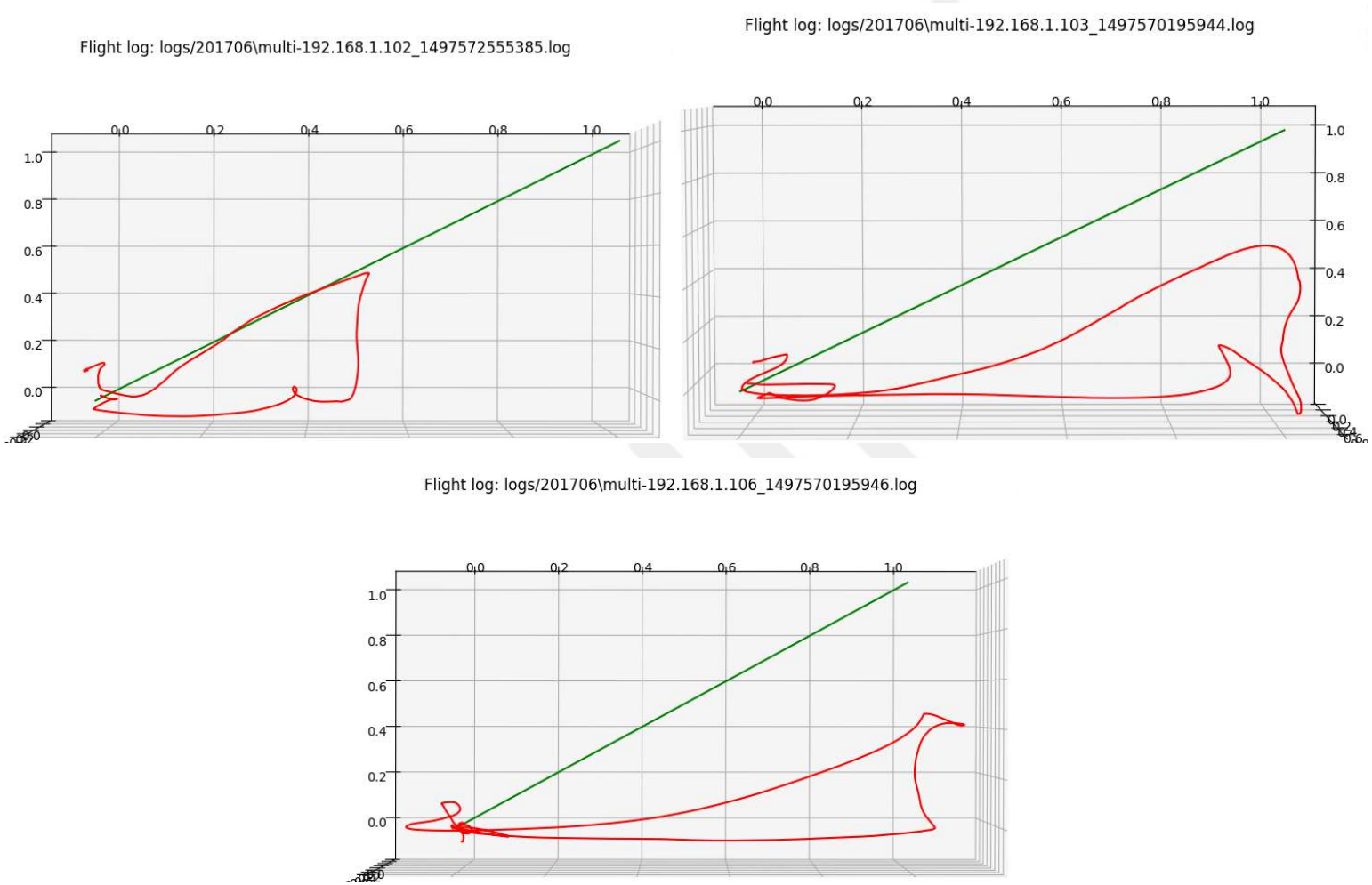


Figure 5.20 Experiment # 13(a) Three quadrotors are flocking in an indoor environment

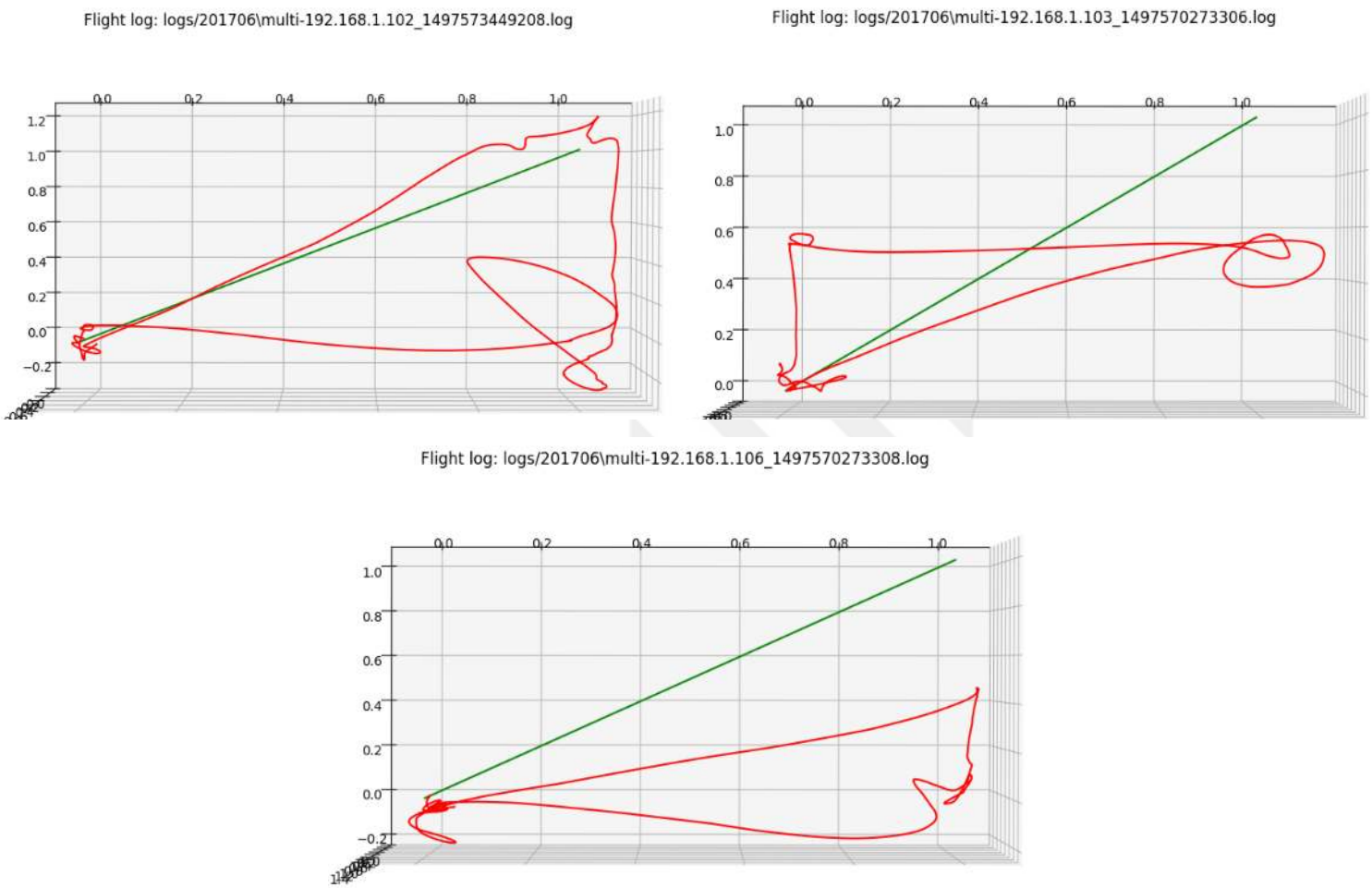


Figure 5.21 Experiment # 13(b) Three quadrotors are flocking in an indoor environment

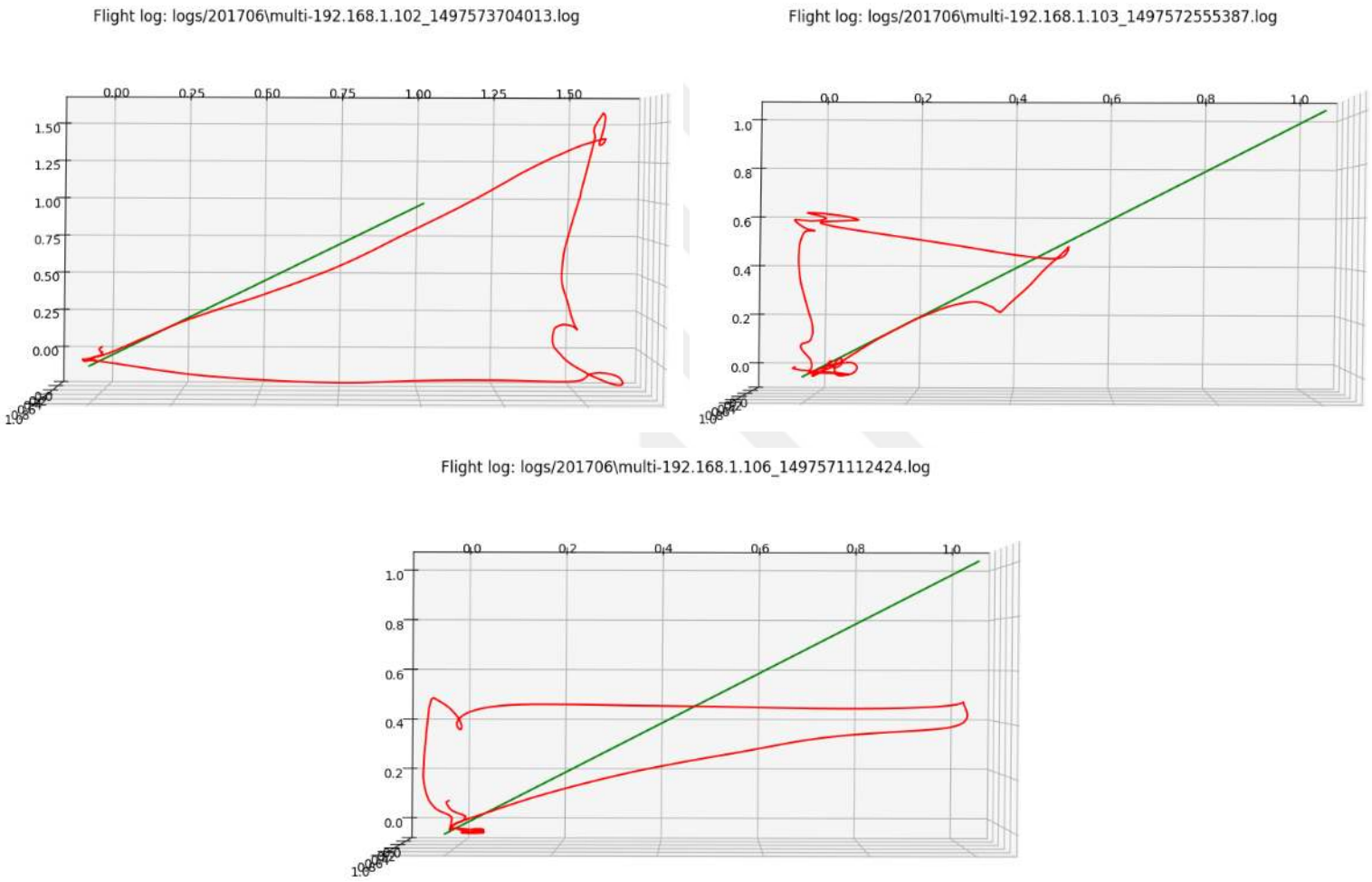


Figure 5.22 Experiment # 13(c) Three quadrotors are flocking in an indoor environment

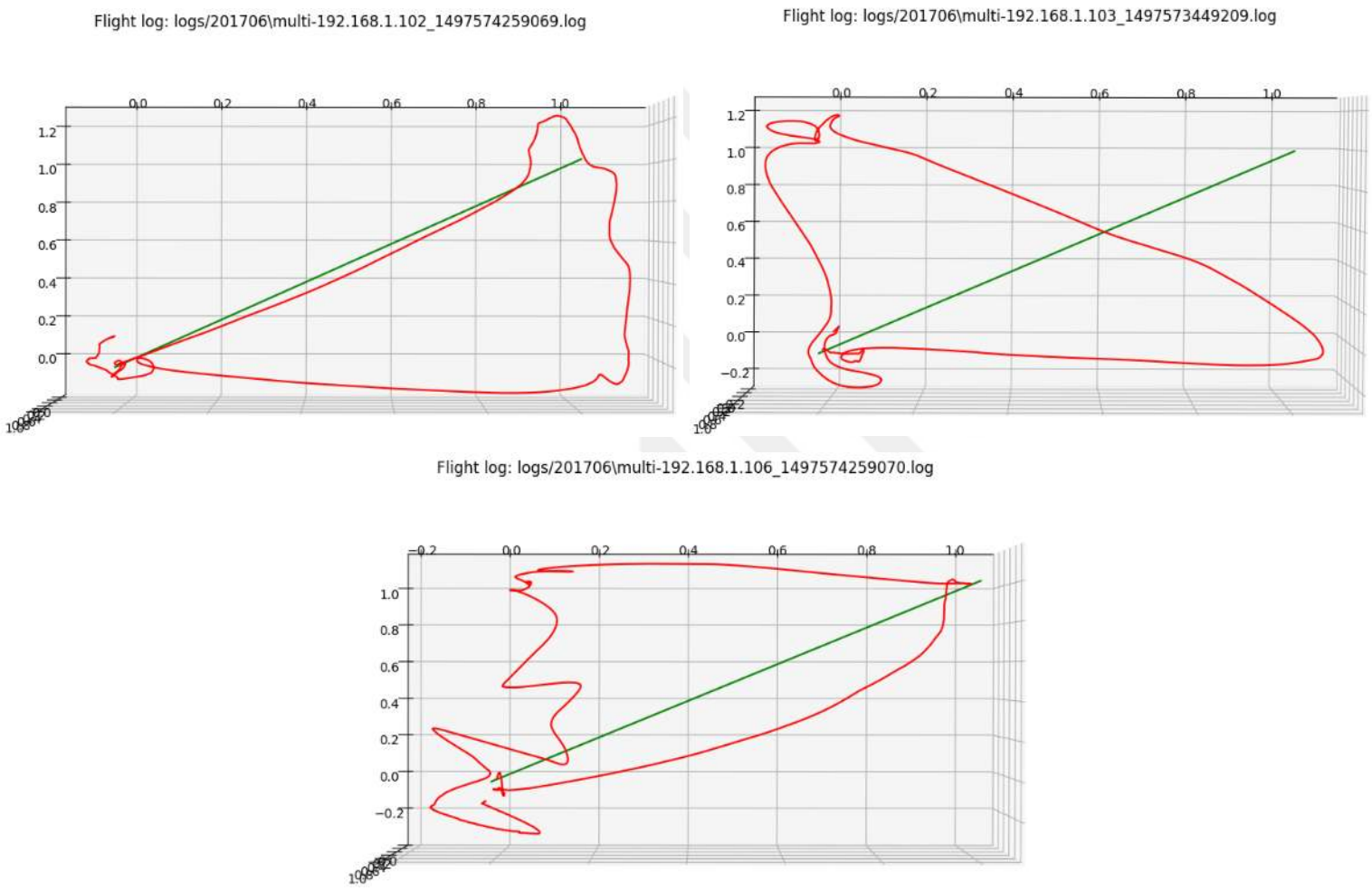


Figure 5.23 Experiment # 13(d) Three quadrotors are flocking in an indoor environment

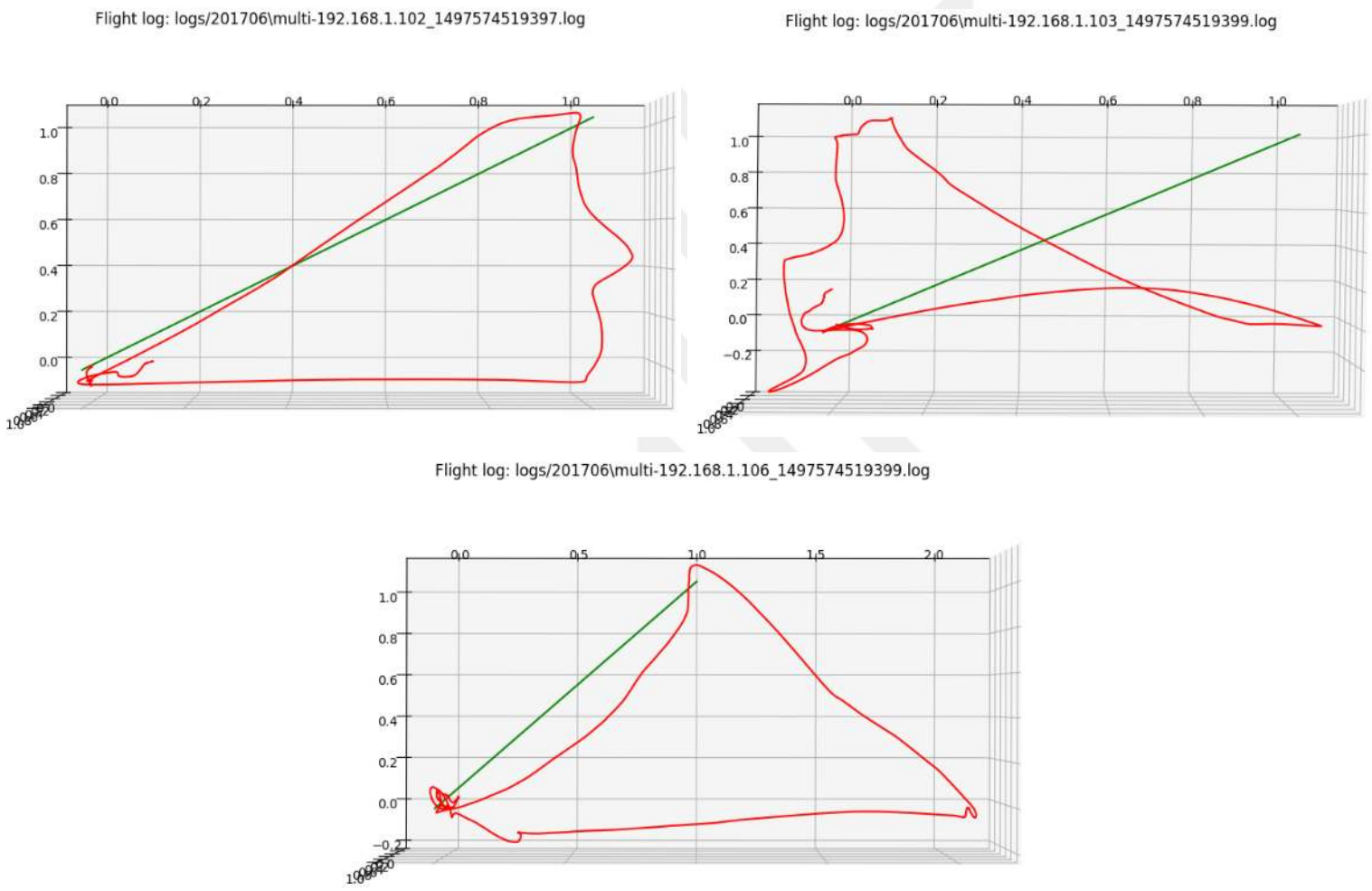
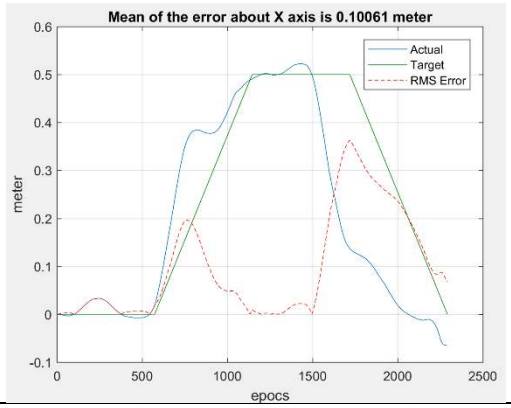
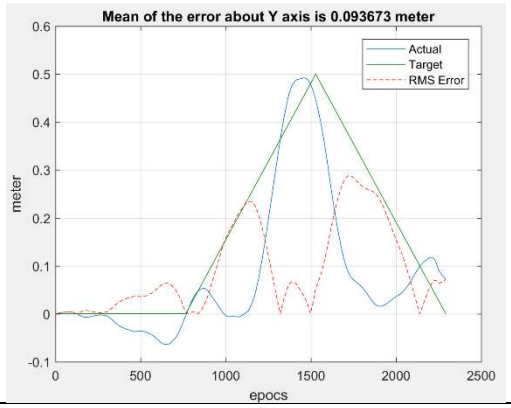


Figure 5.24 Experiment # 13(e) Three quadrotors are flocking in an indoor environment

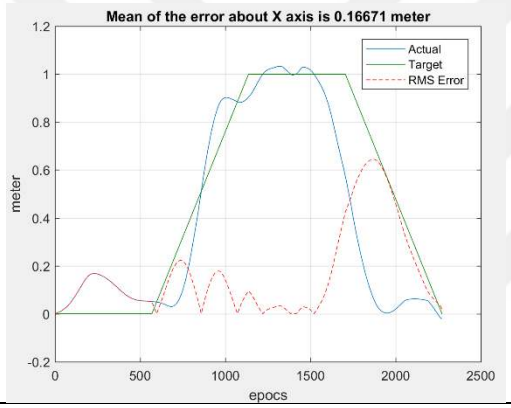
Table 5.1 Group flight experiments' accuracy analysis of Experiment # 13(a)

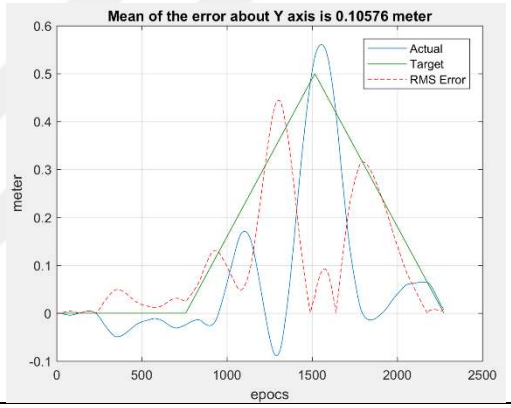
AR Drone with 192.168.1.102 IP address



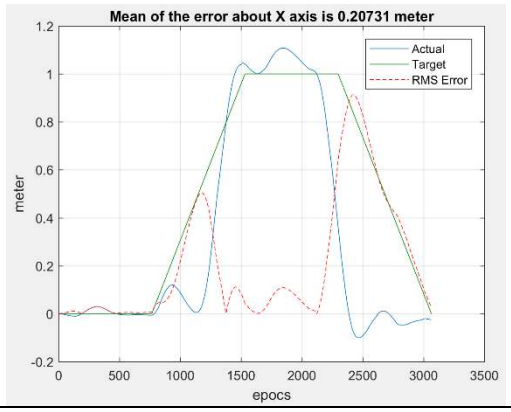


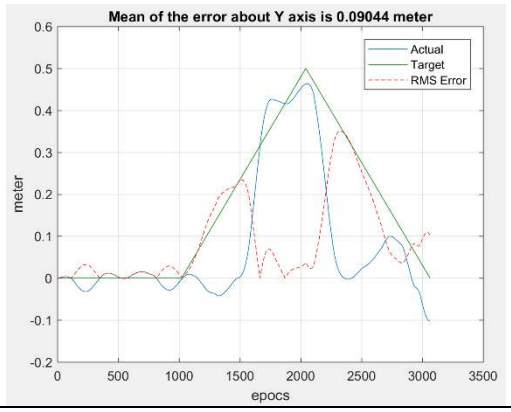
AR Drone with 192.168.1.103 IP address





AR Drone with 192.168.1.106 IP address





As seen in Table 5.1, the mean error of x and y axis is between 9 to 20 cm. For a GPS-denied, vision and essential IMU based flying platforms this error can be acceptable when considering GPS accuracy which is about 4.6 m (Diggelen, 2015).

CHAPTER SIX
CONCLUSIONS

In this thesis, a web based multiple quadrotor aerial vehicle control platform called PhDrone is developed for further multi robot systems. Previous experience of building remote controlled quadrotor, real-time web application programming are the key components of successfully achieving this study.

In the beginning of the study, unmanned aerial vehicles are examined deeply. Multi rotor designs have generally basic mechanical and aerodynamical structure. Beside these advantages, controlling an aerial vehicle with fixed rotor is a nonlinear problem. In order to learn and study further, a custom four rotor UAV is designed and build. This UAV has basic features, but it helps to understand flight dynamics, responses to the controller and overall performance. Then for a group flight, a ready-to-flight quadrotor platform is chosen. Because, that quadrotors already have vision systems, 9 DoF IMU, proximity and pressure sensors for altitude and more importantly a Linux kernel OS for custom programming and more. Additionally, price of the group is going to be lower than a custom build quadrotor with same specifications.

PhDrone software platform is designed to operate any computer platform whether they do support Node.Js or not. If the platform does not support, system could easily serve from remote client in order to control local quadrotors. This flexibility comes from Node.js itself. It has a web-first approach and could serve any type of content with specific publisher code layout.

Quadrotor controller system is composed of several components including quadrotor hardware, application server, bidirectional event based real-time communicator, single and multiple robot controller and web framework. Although the quadrotor hardware is a read-to-fly one, quadrotor's mechanical structures are also analyzed for validating and possible spare part needs. Quadrotor's official SDK is used to understand its control methods and strategies and wrapped in Node.js. For custom computer vision algorithms and self-healing secure wireless network, hardware's Linux kernel is also modified.

In this study, well-known PID and EKF control methods are applied in order to control single or multiple quadrotor from web service in real-time. First single quadrotor flight is studied for controllable and observable flights. Then multiple quadrotors are studied with separate flight missions. At last flocking flight missions are assigned to multiple quadrotors in order to perform. As a result of these studies, an appropriate flight control platform is created for both single and multiple flight control of quadrotor aerial vehicles.

As a future work, the quadrotor hardware may be upgraded into a more battery capacity model. Because the most critical bottle neck is its flight time, this time span is about 10-15 minutes. There are some multi rotors with more flight time, but the optimal solution is to use higher Ah/g battery chemistry. If micro solar photovoltaic cells are become much more efficient than current state, they can also help to extend flight time. Developing an on-air charging stations with supercapacitor on-board will help to extend flight time with some time delays. Wi-Fi connection range can be increased by range extenders or using different protocols like LoRa or LTE. LoRa can help to extent telemetry data, it cannot be used for live video streaming or any other data intense operations. But with a rapid growth in mobile communication speed and data range, LTE or even 5th generation mobile broadband communication standard (5G) can help to transfer both telemetry and live streams in real time to any client in the world without any practical range limit.

In the web interface, user customized features can also be added in order to generate automated flight reports. With real time tracking of each group members' flight that are both fly and landed. When the web interface is served under a public domain and data API is synchronized with other UAV, it can easily operate various kind of robots.

REFERENCES

Anderson, D., & Thomson, D. (2009). Improving rotorcraft survivability to RPG attack using inverse methods. *Proceedings of the SPIE – The International Society for Optical Engineering*, ISSN 0277-786X, *483*, 74830M.

Araos, D. (2013). *WPA2 support for AR Drone 2.0*. Retrieved July 25, 2017, from https://github.com/daraosn/ardrone-wpa2

Arnon R. (2011). *Simple Kalman filter for tracking using OpenCV 2.2*. Retrieved July 25, 2017, from http://www.morethantechnical.com/2011/06/17/simple-kalman-filter-for-tracking-using-opencv-2-2-w-code

Bakke, R. (2015) *KKmulticontroller multi-rotor control board.* Retrieved from March 14, 2015, from http://wiki.seeedstudio.com/wiki/KKmulticontroller

Banderier, C., & Flajolet, P. (2002) Basic analytic combinatorics of directed lattice paths. *Theoretical Computer Science*, *28,* 37–80.

Betz, A. (1966) *Introduction to the theory of flow machines*. Oxford: Pergamon

Brandao, A.S., Rampinelli, V.T.L., Martins, F.N., Sarcinelli-Filho, M., & Carelli, R. (2015) The multilayer control scheme: A strategy to guide n-robots formations with obstacle avoidance. *Journal of Control, Automation and Electrical Systems, 26*, 201–214.

Chen, J., Sun, D., Yang, J., & Chen, H. (2010) Leader-follower formation control of multiple non-holonomic mobile robots incorporating a receding-horizon scheme. *The International Journal of Robotics Research*, *29*(6), 727–747.

Consolini, L., Morbidi, F., Prattichizzo, D., & Tosques, M. (2008) Leader-follower formation control of nonholonomic mobile robots with input constraints. *Automatica 40*(5)*,* 1343– 1349.

Diggelen, F., & Enge, P. (2015) The World's first GPS MOOC and Worldwide Laboratory using Smartphones. *Proceedings of the 28th International Technical Meeting of The Satellite Division of the Institute of Navigation (ION GNSS+ 2015)*, 361 – 369.

Geisendörfer, F. (2014) *node-ar-drone.* Retrieved from July 25, 2017, from https://github.com/felixge/node-ar-drone

Johnston, C. (2013). *Nikola Tesla's remote-control boat, and other unpopular inventions*. Retrieved from July 25, 2017, from https://arstechnica.com/science/2013/07/nikola-teslas-remote-control-boat-and-other-unpopular-inventions/

Keskin, O. & Kıral, Z. (2013) Controlling quadrotor with image processing. *16th National Machine Theory Symposium (UMTS 2013)*, 132-142.

Lerman, K. & Galstyan, A. (2002). Mathematical model of foraging in a group of robots. Effect of interference. *Autonomous Robots, 13*, 127–141.

Liang, O. (2014) *PID tuning and quadcopter flight behavior.* Retrieved from July 25, 2017, https://oscarliang.com/understanding-pid-for-quadcopter-rc-flight/

Little, E. (2016) *Socket.io.* Retrieved from July 25, 2017, from https://github.com/socketio

Malinen, J. (2007) *wpa_cli(8) - Linux man page*. Retrieved July 25, 2017, from https://linux.die.net/man/8/wpa_cli

Malinen, J. (2007) *wpa_passphrase(8) - Linux man page*. Retrieved July 25, 2017, from https://linux.die.net/man/8/wpa_passphrase

Malinen, J. (2013) *Linux WPA/WPA2/IEEE 802.1X Supplicant*. Retrieved July 25, 2017, from https://w1.fi/wpa_supplicant/

Marcolino, L. S. & Chaimowicz, L. (2009). Traffic control for a swarm of robots: Avoiding target congestion. *In Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, 1955–1961.

Orsag, M. & Bogdan, S. (2012). *Influence of Forward and Descent Flight on Quadrotor Dynamics, Recent Advances in Aircraft Technology*, Dr. Ramesh Agarwal (Ed.), ISBN: 978-953-51-0150-5, InTech, Retrieved July 28, 2017 from http://www.intechopen.com/books/recent-advances-in-aircraft-technology/influence-of-forwardand-descent-flight-on-quadrotor-dynamics

Piskorski, S. Brulez, N., Eline, P., D'Haeyer F. (2010). *AR.Drone Developer Guide*

Piskorski, S. (2012) *Parrot Inc. AR.Drone 2.0 Linux Kernel Source*. Retrieved July 28, 2017, from https://www.devzone.parrot.com/projects/show/oss-ardrone2

Ragheb, M. (2010) *Theory of Wind Machines, Betz Equation*. Retrieved May 14, 2011 from http://mragheb.com/NPRE%20475%20Wind%20Power%20Systems/Wind%20Energy%20Conversion%20Theory%20Betz%20Equation..pdf

Sahin, E. (2004). Swarm robotics: From sources of inspiration to domains of application. *In SAB 2014 International Workshop on Swarm Robotics —Revised Selected Papers, Lecture Notes in Computer Science*, *3342,* 10–20.

Sahin, E., Girgin, S., Bayindir, L., & Turgut, A. E. (2008). Swarm robotics. *In Swarm Intelligence, Natural Computing Series,* 87–100.

Scorp2kk (2016) *Tools to handle parrot firmware files (.plf).* Retrieved July 25, 2017, from https://github.com/scorp2kk/ardrone-tool

Simons, M. (1994). *Model aircraft dynamics*. Herts: Argus Books

Zhang, Y., Mehrjerdi, H. (2013) A survey on multiple unmanned vehicles formation control and coordination: normal and fault situations. *International Conference on Unmanned Aircraft Systems (ICUAS)*, 1087–1096.

APPENDICES

Appendix-1: Quadcopter simulation code in MATLAB

```
% define a controller
function c = control(name, Kd, Kp, Ki)
    % Default parameters
    if nargin == 1
        Kd = 4;
        Kp = 3;
        Ki = 5.5;
    elseif nargin == 2 || nargin > 4
        error('Incorrect number of parameters. (name, Kd, Kp, Ki)');
    end
    if strcmpi(name, 'pd')
        c = @(state, thetadot) pd_controller(state, thetadot, Kd,
Kp);
    elseif strcmpi(name, 'pid')
        c = @(state, thetadot) pid_controller(state, thetadot, Kd,
Kp, Ki);
    else
        error(sprintf('Unknown controller ?! "%s"', name));
    end
end
% PD controller
function [input, state] = pd_controller(state, thetadot, Kd, Kp)
    % no integral controller so zero
    if ~isfield(state, 'integral')
        state.integral = zeros(3, 1);
    end
    % total thrust
    total = state.m * state.g / state.k / ...
        (cos(state.integral(1)) * cos(state.integral(2)));
    % PD error and inputs
    err = Kd * thetadot + Kp * state.integral;
    input = err2inputs(state, err, total);
    % update controller state
    state.integral = state.integral + state.dt .* thetadot;
end
```

```
% PID controller
function [input, state] = pid_controller(state, thetadot, Kd, Kp,
Ki)
    % integral controller to zero when not
    if ~isfield(state, 'integral')
        state.integral = zeros(3, 1);
        state.integral2 = zeros(3, 1);
    end
    % stop wind-up
    if max(abs(state.integral2)) > 0.01
        state.integral2(:) = 0;
    end
    % total thrust
    total = state.m * state.g / state.k / (cos(state.integral(1)) *


cos(state.integral(2)));
    % error and inputs.
    err = Kd * thetadot + Kp * state.integral - Ki *
state.integral2;
    input = err2inputs(state, err, total);
    % update controller state
    state.integral = state.integral + state.dt .* thetadot;
    state.integral2 = state.integral2 + state.dt .* state.integral;
end
% inputs; desired torques, desired total thrust, and physical
parameters
function inputs = err2inputs(state, err, total)
    e1 = err(1);
    e2 = err(2);
    e3 = err(3);
    Ix = state.I(1, 1);
    Iy = state.I(2, 2);
    Iz = state.I(3, 3);
    k = state.k;
    L = state.L;
    b = state.b;

    inputs = zeros(4, 1);
    inputs(1) = total/4 -(2 * b * e1 * Ix + e3 * Iz * k * L)/(4 * b
* k * L);
    inputs(2) = total/4 + e3 * Iz/(4 * b) - (e2 * Iy)/(2 * k * L);
    inputs(3) = total/4 -(-2 * b * e1 * Ix + e3 * Iz * k * L)/(4 * b
* k * L);
    inputs(4) = total/4 + e3 * Iz/(4 * b) + (e2 * Iy)/(2 * k * L);
end
```

```
% compute rotation matrix for angles
function R = rotation(angles)
    phi = angles(3);
    theta = angles(2);
    psi = angles(1);
    R = zeros(3);
    R(:, 1) = [
        cos(phi) * cos(theta)
        cos(theta) * sin(phi)
        - sin(theta)
    ];
    R(:, 2) = [
        cos(phi) * sin(theta) * sin(psi) - cos(psi) * sin(phi)
        cos(phi) * cos(psi) + sin(phi) * sin(theta) * sin(psi)
        cos(theta) * sin(psi)
    ];
    R(:, 3) = [
        sin(phi) * sin(psi) + cos(phi) * cos(psi) * sin(theta)
        cos(psi) * sin(phi) * sin(theta) - cos(phi) * sin(psi)
        cos(theta) * cos(psi)
    ];
End

% begin the simulation
function result = run(control, tstart, tend, dt)
    % general constants
    g = 9.81;
    m = 0.5;
    L = 0.25;
    k = 3e-6;
    b = 1e-7;
    I = diag([5e-3, 5e-3, 10e-3]);
    kd = 0.25;
    if nargin < 4
        tstart = 0;
        tend = 4;
        dt = 0.005;
    end
    ts = tstart:dt:tend;
    N = numel(ts);
    xout = zeros(3, N);
    xdotout = zeros(3, N);
    thetaout = zeros(3, N);
    thetadotout = zeros(3, N);
    inputout = zeros(4, N);
    controller_params = struct('dt', dt, 'I', I, 'k', k, 'L', L,
'b', b, 'm', m, 'g', g);
    x = [0; 0; 10];
    xdot = zeros(3, 1);
    theta = zeros(3, 1);
```

```
    if nargin == 0
        thetadot = zeros(3, 1);
    else
        deviation = 300;
        thetadot = deg2rad(2 * deviation * rand(3, 1) - deviation);
    end
    ind = 0;
    for t = ts
        ind = ind + 1;
        if nargin == 0
            i = input(t);
        else
            [i, controller_params] = controller(controller_params,
thetadot);
        end
        % calculate forces, torques, and accelerations
        omega = thetadot2omega(thetadot, theta);
        a = acceleration(i, theta, xdot, m, g, k, kd);
        omegadot = angular_acceleration(i, omega, I, L, b, k);
        % system state
        omega = omega + dt * omegadot;
        thetadot = omega2thetadot(omega, theta);
        theta = theta + dt * thetadot;
        xdot = xdot + dt * a;
        x = x + dt * xdot;
        % store simulation state
        xout(:, ind) = x;
        xdotout(:, ind) = xdot;
        thetaout(:, ind) = theta;
        thetadotout(:, ind) = thetadot;
        inputout(:, ind) = i;
    end
    result = struct('x', xout, 'theta', thetaout, 'vel', xdotout,
...
                    'angvel', thetadotout, 't', ts, 'dt', dt,
'input', inputout);
end
% test input
function in = input(t)
    in = zeros(4, 1);
    in(:) = 700;
    in(1) = in(1) + 150;
    in(3) = in(3) + 150;
    in = in .^ 2;
end
% calculate thrust given current inputs and thrust coefficient
function T = thrust(inputs, k)
    T = [0; 0; k * sum(inputs)];
End
```

```
% calculate torques, given current inputs, length, drag coefficient,
and thrust coefficient
function tau = torques(inputs, L, b, k)
    tau = [
        L * k * (inputs(1) - inputs(3))
        L * k * (inputs(2) - inputs(4))
        b * (inputs(1) - inputs(2) + inputs(3) - inputs(4))
    ];
end
% calculate acceleration in inertial reference frame
function a = acceleration(inputs, angles, vels, m, g, k, kd)
    gravity = [0; 0; -g];
    R = rotation(angles);
    T = R * thrust(inputs, k);
    Fd = -kd * vels;
    a = gravity + 1 / m * T + Fd;
end
% calculate angular acceleration in body frame
function omegad = angular_acceleration(inputs, omega, I, L, b, k)
    tau = torques(inputs, L, b, k);
    omegad = inv(I) * (tau - cross(omega, I * omega));
end

% calculate derivatives of roll, pitch, yaw to omega.
function omega = thetadot2omega(thetadot, angles)
    phi = angles(1);
    theta = angles(2);
    psi = angles(3);
    W = [
        1, 0, -sin(theta)
        0, cos(phi), cos(theta)*sin(phi)
        0, -sin(phi), cos(theta)*cos(phi)
    ];
    omega = W * thetadot;
end
% calculate omega to roll, pitch, yaw derivatives
function thetadot = omega2thetadot(omega, angles)
    phi = angles(1);
    theta = angles(2);
    psi = angles(3);
    W = [
        1, 0, -sin(theta)
        0, cos(phi), cos(theta)*sin(phi)
        0, -sin(phi), cos(theta)*cos(phi)
    ];
    thetadot = inv(W) * omega;
end
```

```
% visualize the quadrocopter simulation in 3D
function h = show_test(data)
    figure; plots = [subplot(3, 2, 1:4), subplot(3, 2, 5),
subplot(3, 2, 6)];
    subplot(plots(1));
    pause;
    [t thrusts] = quadrocopter;
    axis([-10 30 -20 20 5 15]);
    zlabel('Height');
    title('3D quadrocopter Flight Simulation');
    animate(data, t, thrusts, plots);
end

% animate a quadrocopter in flight
function animate(data, model, thrusts, plots)
    for t = 1:2:length(data.t)
        subplot(plots(1));
        dx = data.x(:, t);
        move = makehgtform('translate', dx);
        angles = data.theta(:, t);
        rotate = rotation(angles);
        rotate = [rotate zeros(3, 1); zeros(1, 3) 1];
        set(model,'Matrix', move * rotate);
        scales = exp(data.input(:, t) / min(abs(data.input(:, t))) +
5) - exp(6) +  1.5;
        for i = 1:4
            s = scales(i);
            if s < 0
                scalez = makehgtform('yrotate', pi)  *
makehgtform('scale', [1, 1, abs(s)]);
            elseif s > 0
                scalez = makehgtform('scale', [1, 1, s]);
            end
            set(thrusts(i), 'Matrix', move * rotate * scalez);
        end
        xmin = data.x(1,t)-20;
        xmax = data.x(1,t)+20;
        ymin = data.x(2,t)-20;
        ymax = data.x(2,t)+20;
        zmin = data.x(3,t)-5;
        zmax = data.x(3,t)+5;
        axis([xmin xmax ymin ymax zmin zmax]);
        drawnow;
        subplot(plots(2));
        multiplot(data, data.angvel, t);
        xlabel('Time (s)');
        ylabel('Angular Velocity (rad/s)');
        % f, q, y are coded as red, green, and blue
        title('Angular Velocity | phi; red, tetha; green, psi;
blue');
```

```
        grid on
        legend({'phi','tetha', 'psi'}, 'Position', [0,0,0.1,0.1])
        subplot(plots(3));
        multiplot(data, data.theta, t);
        xlabel('Time (s)');
        ylabel('Angular Displacement (rad)');
        title('Angular Displacement');
    end
end
function multiplot(data, values, ind)
    times = data.t(:, 1:ind);
    values = values(:, 1:ind);
    plot(times, values(1, :), 'r-', times, values(2, :), 'g.',
times, values(3, :), 'b-.');
    xmin = min(data.t);
    xmax = max(data.t);
    ymin = 1.1 * min(min(values));
    ymax = 1.1 * max(max(values));
    axis([xmin xmax ymin ymax]);
end
% draw a quadrocopter
function [h thrusts] = quadrocopter ()
    % draw rots.
    h(1) = prism(-5, -0.25, -0.25, 10, 0.5, 0.5);
    h(2) = prism(-0.25, -5, -0.25, 0.5, 10, 0.5);
    % draw motors
    [x y z] = sphere;
    x = 0.5 * x;
    y = 0.5 * y;
    z = 0.5 * z;
    h(3) = surf(x - 5, y, z, 'EdgeColor', 'none', 'FaceColor', 'c');
    h(4) = surf(x + 5, y, z, 'EdgeColor', 'none', 'FaceColor', 'c');
    h(5) = surf(x, y - 5, z, 'EdgeColor', 'none', 'FaceColor', 'c');
    h(6) = surf(x, y + 5, z, 'EdgeColor', 'none', 'FaceColor', 'c');
    % draw thrust
    [x y z] = cylinder(0.1, 7);
    thrusts(1) = surf(x, y + 5, z, 'EdgeColor', 'none', 'FaceColor',
'g');
    thrusts(2) = surf(x + 5, y, z, 'EdgeColor', 'none', 'FaceColor',
'r');
    thrusts(3) = surf(x, y - 5, z, 'EdgeColor', 'none', 'FaceColor',
'g');
    thrusts(4) = surf(x - 5, y, z, 'EdgeColor', 'none', 'FaceColor',
'r');
    for i = 1:4
        x = hgtransform;
        set(thrusts(i), 'Parent', x);
        thrusts(i) = x;
    end
```

```
    % group all quadrocopter parts into one object
    t = hgtransform;
    set(h, 'Parent', t);
    h = t;
end
function h = prism(x, y, z, w, l, h)
    [X Y Z] = prism_faces(x, y, z, w, l, h);
    faces(1, :) = [4 2 1 3];
    faces(2, :) = [4 2 1 3] + 4;
    faces(3, :) = [4 2 6 8];
    faces(4, :) = [4 2 6 8] - 1;
    faces(5, :) = [1 2 6 5];
    faces(6, :) = [1 2 6 5] + 2;
    for i = 1:size(faces, 1)
        h(i) = fill3(X(faces(i, :)), Y(faces(i, :)), Z(faces(i, :)),
'b'); hold on;
    end
    t = hgtransform;
    set(h, 'Parent', t);
    h = t;
end
function [X Y Z] = prism_faces(x, y, z, w, l, h)
    X = [x x x x x+w x+w x+w x+w];
    Y = [y y y+l y+l y y y+l y+l];
    Z = [z z+h z z+h z z+h z z+h];
end
```

Appendix-2: Quadcopter flight log display in python

```
1. #!/usr/bin/env python
2. # -*- coding: utf-8 -*-
3.
4. from mpl_toolkits.mplot3d import Axes3D
5. from matplotlib.pyplot import cm
6. import matplotlib.pyplot as plt
7. import matplotlib as mpl
8. import numpy as np
9. import csv
10.
11.def getColumn(filename, column):
12.    results = csv.reader(open(filename), delimiter=",")
13.    return [(result[column]) for result in results]
14.
15.def missionPath(path):
16.    if path == 'P1':
17.        # # mission path P1 aka square
18.        mx = [0, 0, 1, 1, 0]
19.        my = [0, 1, 1, 0, 0]
20.        mz = [1, 1, 1, 1, 1]
21.    elif path == 'P1a':
22.        # # mission path P1a
23.        mx = [0.00, 0.00, 1.00, 1.00, 0.00]
24.        my = [0.00, -1.00, -1.00, 0.0, 0.0]
25.        mz = [1.00, 1.00, 1.00, 1.00, 1.00]
26.    elif path == 'P2':
27.        # mission path P2 aka home
28.        mx = [0.00, 0.00, 0.50, 0.50, 1.00, 1.00, 0.00]
29.        my = [0.00, 1.00, 1.25, 0.50, 0.50, 0.00, 0.00]
30.        mz = [1.00, 1.00, 1.00, 1.00, 1.00, 1.00, 1.00]
31.    elif path == 'P3':
32.        # mission path P3 aka play
33.        mx = [0.00, 0.00, 1.00, 0.00]
34.        my = [0.00, 1.00, 0.50, 0.00]
35.        mz = [1.00, 1.00, 1.00, 1.00]
36.    elif path == 'P4':
37.        # mission path P4 aka pisagor
38.        mx = [0.00, 0.00, 1.50, 0.00]
39.        my = [0.00, 1.00, 1.00, 0.00]
40.        mz = [1.00, 1.00, 1.00, 1.00]
41.    elif path == 'P4a':
42.        # mission path P4a aka reverse pisagor
43.        mx = [0.00, 1.00, 1.00, 0.00]
44.        my = [0.00, 1.50, 0.00, 0.00]
45.        mz = [1.00, 1.00, 1.00, 1.00]
46.    elif path == 'P4b':
47.        # mission path P4b aka big pisagor
48.        mx = [0.00, 0.00, 1.00, 0.00]
49.        my = [0.00, 1.00, 0.00, 0.00]
50.        mz = [1.00, 1.00, 1.00, 1.00]
```

```
51.     elif path == 'P5':
52.         # mission path P5 aka slash
53.         mx = [0.00, 1.00]
54.         my = [0.00, 1.00]
55.         mz = [1.00, 1.00]
56.     elif path == 'P6':
57.         # mission path P6 aka gun
58.         mx = [0.00, 0.00, 2.50, 2.50, 0.50, 0.50, 0.00]
59.         my = [0.00, 2.00, 2.00, 1.00, 1.00, 0.00, 0.00]
60.         mz = [1.00, 1.00, 1.00, 1.00, 1.00, 1.00, 1.00]
61.     elif path == 'P7':
62.         # mission path P7 aka sail
63.         mx = [0.00, 0.00, 1.00, 1.50, 2.50, 0.50, 0.50, 0.00]
64.         my = [0.00, 1.00, 2.00, 2.00, 1.00, 1.00, 0.00, 0.00]
65.         mz = [1.00, 1.00, 1.00, 1.00, 1.00, 1.00, 1.00, 1.00]
66.     elif path == 'P8':
67.         # mission path P8 aka mountain
68.         mx = [0.00, 0.50, 1.50, 2.50, 0.50, 0.50, 0.00]
69.         my = [0.00, 1.50, 2.00, 1.00, 1.00, 0.00, 0.00]
70.         mz = [1.00, 1.00, 1.00, 1.00, 1.00, 1.00, 1.00]
71.     elif path == 'P9':
72.         # mission path P9 aka diamond
73.         mx = [0.00, -0.50, 0.75, 1.25, 0.0]
74.         my = [0.00, 1.00, 1.25, 0.25, 0.00]
75.         mz = [1.00, 1.00, 1.00, 1.00, 1.00]
76.     elif path == 'P10':
77.         # mission path P10 aka flag
78.         mx = [0.00, 0.00, 2.00, 1.00, 1.00, 0.00]
79.         my = [0.00, 4.00, 4.00, 4.00, 0.00, 0.00]
80.         mz = [1.00, 1.00, 1.00, 1.00, 1.00, 1.00]
81.     else:
82.         mx = [0, 0, 1, 1, 0]
83.         my = [0, 1, 1, 0, 0]
84.         mz = [1, 1, 1, 1, 1]
85.     return  {'x':mx, 'y':my ,'z':mz }
86.
87.# mission path P1
88.# log_file = "logs/mission-s2-2017-01-13_11-01-02.txt"
89.
90.fig = plt.figure()
91.ax = fig.gca(projection='3d')
92.title = "Flight log: " + log_file
93.mpl.rcParams['legend.fontsize'] = 10
94.ax.plot(mp['x'], mp['y'], mp['z'], label='missionData.state', co
   lor='green')
95.
```

```
if 'divider' in globals():
    x1 = getColumn(log_file,0)[1:divider]
    y1 = getColumn(log_file,1)[1:divider]
    z1 = getColumn(log_file,2)[1:divider]
  else:
      x1 = getColumn(log_file,0)[1:]
      y1 = getColumn(log_file,1)[1:]
      z1 = getColumn(log_file,2)[1:]

  px1 = []
  for xx in x1:
      try:
          px1.append(float(xx))
      except Exception as e:
          continue

  py1 = []
  for yy in y1:
      try:
          py1.append(float(yy))
      except Exception as e:
          continue

  pz1 = []
  for zz in z1:
      try:
          pz1.append(float(zz))
      except Exception as e:
          continue

  ax.plot(px1, py1, pz1, label = 'controlData.state #1', col
or = 'red')

  if 'divider' in globals():
      start = divider
      color = iter(['#1f78b4','#fdbf6f','#ff7f00','#cab2d6',
'#6a3d9a','#ffff99','#b15928', '#ffed6f', '#8dd3c7'])

      for pn in range(2, fnum):
          x2 = getColumn(log_file,0)[start:divider*pn]
          y2 = getColumn(log_file,1)[start:divider*pn]
          z2 = getColumn(log_file,2)[start:divider*pn]

          px2 = []
          for xx in x2:
              try:
                  px2.append(float(xx))
              except Exception as e:
                  continue

          py2 = []
          for yy in y2:
              try:
                  py2.append(float(yy))
```

```
                except Exception as e:
                    continue

            pz2 = []
            for zz in z2:
                try:
                    pz2.append(float(zz))
                except Exception as e:
                    continue

            ax.plot(px2, py2, pz2, label = 'controlData.state #%d' % pn, color = next(color))

            start = divider*pn

    plt.title(title)
    ax.legend()
    plt.show()
```

Appendix-3: Group flight experiments' accuracy analysis code in MATLAB

```
clc

clear

data = importdata('multi-192.168.1.102_1497572555385.txt');

first = 0;
Xlast = 1;
Ylast = 0.5;
Xdivisor = 4;
Ydivisor = 3;



Xraw = data(:,1);
Yraw = data(:,2);
Zraw = data(:,3);



zeros = find(Zraw);
total = zeros(end);


Xlimit = ceil(total / Xdivisor);
Ylimit = ceil(total / Ydivisor);
```

```
Xstep = Xlast / Xlimit;
Ystep = Ylast / Ylimit;



if Xdivisor == 3
    for n = 1:Xlimit
        Xmap(n,:) = first;
    end

    for n = (Xlimit+1):(2*Xlimit)
        Xmap(n,:) = first + Xstep;
        first = Xmap(n,:);
    end

    for n = ((2*Xlimit)+1):(3*Xlimit)
        Xmap(n,:) = Xlast - Xstep;
        Xlast = Xmap(n,:);
    End

elseif Xdivisor == 4
    for n = 1:Xlimit
        Xmap(n,:) = first;
    end



    for n = (Xlimit+1):(2*Xlimit)
        Xmap(n,:) = first + Xstep;
        first = Xmap(n,:);
    end



    for n = ((2*Xlimit)+1):(3*Xlimit)
        Xmap(n,:) = Xlast;
    end



    for n = ((3*Xlimit)+1):(4*Xlimit)
        Xmap(n,:) = Xlast - Xstep;
        Xlast = Xmap(n,:);
    end
end

first = 0;
```

```
if Ydivisor == 3
    for n = 1:Ylimit
        Ymap(n,:) = first;
    end


    for n = (Ylimit+1):(2*Ylimit)
        Ymap(n,:) = first + Ystep;
        first = Ymap(n,:);
    end




    for n = ((2*Ylimit)+1):(3*Ylimit)
        Ymap(n,:) = Ylast - Ystep;
        Ylast = Ymap(n,:);
    end

elseif Ydivisor == 4


    for n = 1:Ylimit
        Ymap(n,:) = first;
    end



    for n = (Ylimit+1):(2*Ylimit)
        Ymap(n,:) = first + Ystep;
        first = Ymap(n,:);
    end



    for n = ((2*Ylimit)+1):(3*Ylimit)
        Ymap(n,:) = Ylast;
    end



    for n = ((3*Ylimit)+1):(4*Ylimit)
        Ymap(n,:) = Ylast - Ystep;
        Ylast = Ymap(n,:);
    end
end

for n = 1:total
    Xrmse(n,:) = sqrt(mean((Xmap(n,:) - Xraw(n,:)).^2));
end
```

```
for n = 1:total
     Yrmse(n,:) = sqrt(mean((Ymap(n,:) - Yraw(n,:)).^2));
end

figure('Name','X axis with target, actual and
error','NumberTitle','off')

plot(Xraw(1:total), 'DisplayName', 'Actual');

hold on;


plot(Xmap, 'DisplayName', 'Target', 'Color',[0 0.5 0]);

plot(Xrmse, '--', 'DisplayName', 'RMS Error', 'Color', [1 0
0]);

xlabel({'epocs'});

ylabel({'meter'});

title(['Mean of the error about X axis is '
num2str(mean(Xrmse)) ' meter']);

legend('show');

grid on;

figure('Name','Y axis with target, actual and
error','NumberTitle','off')

plot(Yraw(1:total), 'DisplayName', 'Actual');

hold on;

plot(Ymap, 'DisplayName', 'Target', 'Color', [0 0.5 0]);

plot(Yrmse, '--', 'DisplayName', 'RMS Error', 'Color', [1 0
0]);

xlabel({'epocs'});

ylabel({'meter'});

title(['Mean of the error about Y axis is '
num2str(mean(Yrmse)) ' meter']);

legend('show');

grid on;
```