

DOKUZ EYLÜL UNIVERSITY
GRADUATE SCHOOL OF NATURAL AND APPLIED
SCIENCES

BALANCING AND ORIENTATION ANALYSIS
AND IMPLEMENTATION OF QUADROTOR
FLYING ROBOT

by
Oğuz GORA

January, 2013

İZMİR

BALANCING AND ORIENTATION ANALYSIS AND IMPLEMENTATION OF QUADROTOR FLYING ROBOT

**A Thesis Submitted to the
Graduate School of Natural and Applied Sciences of Dokuz Eylül University
In Partial Fulfillment of Requirements for the Degree of Master of Science
in Mechatronics Engineering, Mechatronics Engineering Program**

by

Oğuz GORA

January, 2013

İZMİR

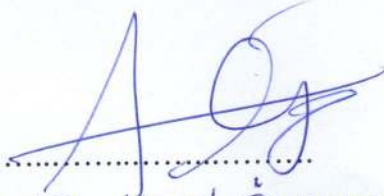
M.Sc THESIS EXAMINATION RESULT FORM

We have read the thesis entitled **“BALANCING AND ORIENTATION ANALYSIS AND IMPLEMENTATION OF QUADROTOR FLYING ROBOT”** completed by **OĞUZ GÖR** under supervision of **ASST. PROF. DR. TANER AKKAN** and we certify that in our opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Master of Science.



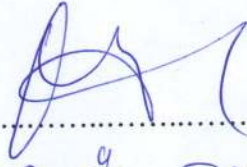
ASST. PROF. DR. TANER AKKAN

Supervisor



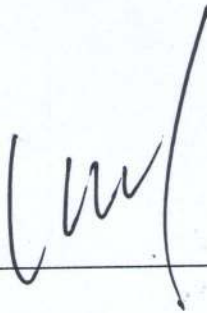
Asst. Prof. Dr. Ahmet ÖZKURT

(JuryMember)



Asst. Prof. Dr. Ögür TAMER

(JuryMember)



Prof. Dr. Mustafa SABUNCU

Director

Graduate School of Natural and Applied Sciences

ACKNOWLEDGEMENTS

I thank to my supervisor, Assistant Prof. Taner AKKAN, for giving me the opportunity to work in a very interesting area, guidance throughout my M.Sc. study. I would like to thank Ertuğrul ÖZKAN for helping me on different parts of study. I want to thank all the scientists who dedicate themselves to scientific studies. And finally, thanks to my family for their valuable support.

Oğuz GORA

BALANCING AND ORIENTATION ANALYSIS AND IMPLEMENTATION OF QUADROTOR FLYING ROBOT

ABSTRACT

The thesis reviews the control, balance and orientation angles of a quadrotor flying robot. A testbed is designed and used for the flying robot. The orientation data that is acquired by the orientation sensor which is externally mounted to the quadrotor flying robot is transferred through wireless communication to the computer. The transferred data is analyzed by Matlab and Python software platforms. Thus, Real Time behaviors of the remotely controlled system is analyzed. The hardware consists of 9-Degrees of Freedom; Orientation Sensor, Xbee Explorer Modules, Walkera Ufo 5#. In the study, the threshold control of quadrotor flying robot is successfully completed.

Keywords: Balance, orientation, quadrotor flying robot, Xbee, Matlab, Python.

DÖRT ROTORLU UÇAN ROBOT DENGE VE YÖNELİM ANALİZİ İLE UYGULAMASI

ÖZ

Bu çalışma, dört rotorlu uçan robotun kontrolünü, dengesini ve yönelim açılarını incelemektedir. Uçan robot için geliştirilmiş olan test platformu kullanılmıştır. Uçan robota harici takılan IMU yardımıyla alınan oryantasyon verileri, kablosuz olarak bilgisayara taşınmaktadır. Taşınan veriler Matlab ve Python programı ile analiz edilmektedir. Böylece, sistemin kumandadan aldığı hareket komutlarını çalıştırırken gösterdiği gerçek zamanlı davranışlar analiz edilmektedir. Uygulama için dokuz serbestlik derecesine sahip bir oryantasyon sensörü, Xbee Explorer modülleri, Walkera Ufo 5# dört rotorlu uçan robotu kullanılmıştır. Çalışmada dört rotorlu uçan robotun threshold kontrolü de yapılmıştır.

Anahtar kelimeler: Denge, yönelim, dört rotorlu uçan robot, Xbee, Matlab, Python.

CONTENTS

	Page
THESIS EXAMINATION RESULT FORM	ii
ACKNOWLEDGEMENTS	iii
ABSTRACT	iv
ÖZ	v
CHAPTER ONE – INTRODUCTION	1
1.1 Aim	1
1.2 Layout of The Thesis	1
1.3 Previous Studies	2
CHAPTER TWO – QUADROTOR FLYING ROBOT SYSTEMS	9
2.1 Quadrotor Flying Robots	9
2.1.1 Quadrotor Robot's Dynamics	12
2.1.2 Quadrotor Robot's Motions	15
2.2 Components of Quadrotor Robots	17
2.2.1 Frame	17
2.2.2 Motors	18
2.2.3 Main Control Units	18
2.2.4 Electronic Speed Controller	21
2.2.5 Li-Po Batteries	21
2.2.6 Propellers	22
2.2.7 Sensors	23
2.2.7.1 Inertial Measurement Unit (IMU)	23
2.2.7.2 Global Positioning System (GPS)	24
2.2.7.3 Ultrasonic Distance Measurement	25
2.2.7.4 Pressure Altimeter Sensor	27
2.2.7.5 Camera and Vision Sense	28

CHAPTER THREE – TESTBED DESIGN	31
3.1 Testbeds for Quadrotor Flying Robot	31
3.2 Our Designed Testbed	32
CHAPTER FOUR – MATLAB INTERFACE AND PYTHON SIMULATION.....	37
4.1 Matlab Interface	37
4.2 Python Simulation	38
4.2.1 Kalman Filter	41
4.2.2 Moving Average Filter in Python Simulation	43
CHAPTER FIVE – EXPERIMENTAL WORKS.....	46
5.1 Walkera UFO #5	46
5.2 Experiment Set-up Configuration	46
5.2.1 Xbee Modules.....	47
5.2.2 Configuring Xbee Modules	48
5.2.3 9-DOF Razor IMU.....	50
5.2.4 Programming Razor IMU	51
5.2.5 Realizing of Computer Controlled Remote Controller.....	51
CHAPTER SIX – CONCLUSIONS AND FUTURE WORKS.....	54
6.1 Conclusions	54
6.2 Results	55
6.3 Future Works	56
REFERENCES.....	58
APPENDIX – Razor Imu – Output File.....	62
APPENDIX – Razor Imu – AHRS Code	64
APPENDIX – Matlab – kaydet.m File	69
APPENDIX – Matlab – sifir.m File	71
APPENDIX – PC-Arduino File	72

APPENDIX – Python Simulation Codes 1.....	75
APPENDIX – Python Simulatin Codes 2.....	82

CHAPTER ONE

INTRODUCTION

1.1 Aim

Flying mechatronic systems are very complex systems. Especially, the helicopter is one of the most complex flying systems. Its complexity is sourced from its multi-functional structure and maneuver capability for performing the flying and orientation. The classical helicopter consists of a main rotor and a tail rotor. Additionally, other flying systems exist such as twin rotor, tandem rotor, quadrotor, and hexacopter.

In this study, we are interested in quadrotor types of air vehicles. The difference between conventional helicopters and quadrotor air vehicles is their enhanced maneuver ability. But, the control mechanism of quadrotors is harder than the conventional air vehicles.

Balancing a quadrotor flying prototype using orientation sensor is the main challenge of this thesis study. First, a testbed is designed and developed for studying in a safe environment. Second, the acceleration and orientation data are gathered by the Inertial Measurement Unit (IMU) sensor that externally attached on Walkera UFO 5# prototype. Third, the sensor data are transferred via Xbee modules wirelessly and analyzed in MATLAB and Python software platforms. Last, a threshold control algorithm is performed to balance the air vehicle mounted on a testbed.

1.2 Layout of The Thesis

Chapter 2 is generally about quadrotor flying robots. The chapter will provide information on every component of quadrotor robots and some flight parameters as well. Chapter 3 provides information about designed testbed. Chapter 4 explains the work about Matlab & Python and presents the codes. Chapter 5 and 6 summarize the results and experimental works.

1.3 Previous Studies

Quadrotor vehicles are not newly designed vehicles. In 1922, Dr. George de Bothezat and Ivan Jerome realized the first quadrotor vehicle. The quadrotor rotorcraft is shown in Figure 1.1. The structure of the vehicle is X-shaped and weighed as 1678 kg. It consists of six blades rotor which diameter is 8.1m. Two small propellers with variable pitch were used for thrusting and yaw control at the lateral ends (J.G.Leisman, 2006).



Figure 1.1 The quadrotor rotorcraft of Bothezat (Pedro, Rogelio, Alejandro, 2005).

This quadrotor configuration did not get popular until early 80's. After, many researchers have started to study quadrotors for unmanned air vehicles (UAV) because of their simple structures, attractive payload capacities. Also, their low cost construction is the main key for such remotely controlled vehicles.

The Draganflyer is commercially very popular radio controlled (RC) air vehicle (Draganfly, 2012). It has onboard attitude controller which eases the balanced operation for the movements and to control is relatively simpler than the conventional radio controlled helicopters. DraganFlyer is shown in Figure 1.2. In academical researches, balancing and managing air movements of such kind of quadrotor vehicles are a new research field. Among the multirotor vehicle designs,

quadrotors are popular. Three rotor designs are not stable as quadrotors. Six rotor designs are more complex. There are many other models developed in universities.



Figure 1.2 DraganflyerX-Pro.

Canberra, Australian National University's work on X-4 Flyer Mark II is thrust and stability studies (Pounds P., Mahony R., Corke P., 2006). X-4 Flyer Mark II has weight of 2 kg and is shown in Figure 1.3. Mark II as a successor of Mark I study has lighter sensor unit of IMU. The control board and a full six-axis IMU with magnetometer 'Eimu' were built by the CSIRO ICT Centre. Two HC-12 microprocessor controllers are used as control board. The quadrotor uses a simple proportional-integral-derivative control. The previous iteration used a slow, off-board control system connected to the flyer by a tether (Kivrak A.Ö, 2006).

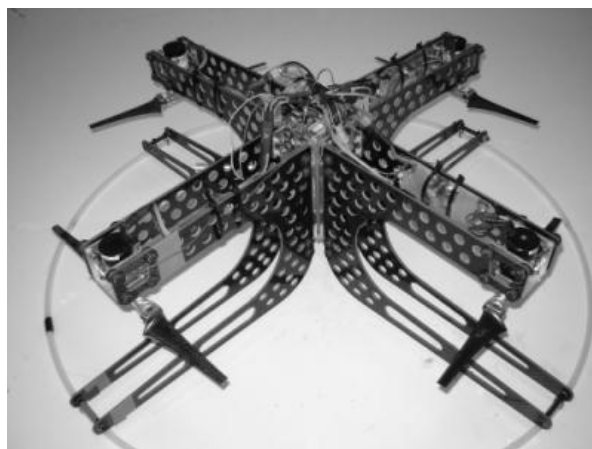


Figure 1.3 X-4 Flyer Mark II (Pounds P., Mahony R., Corke P., 2006).

Project of Stanford University STARMAC consists of four X4-flyer rotorcraft and has autonomous trajectory tracking system. This trajectory generated by reliable trajectory planning algorithms for multi agent systems and it is compatible for multiple-vehicle coordination. The open-loop system is unstable; but onboard controller makes the system easily commended by human operators. It tracks commands for the three angular rate and thrust. Lithium-polymer batteries are used to increase flight duration and payload capacity. The system is controlled by two Microchip 40 Mhz PIC microcontrollers programmed in C. Attitude stabilization were performed on board controller and can be changed from the base station on the ground. The communication is via a Bluetooth Class II device which range is approximately 150 ft. The frequency is 2.4 GHz with bandhopping, error correction and automatic retransmission specifications. STARMAC is shown in Figure 1.4.



Figure 1.4 Starmac (<http://www.flyingrobots.org/mavws/slides/08-Hoffmann.pdf>, 2012)

Swiss Federal Institute of Technology study is about the mechanical design, dynamic modeling, sensing, and control of an indoor Vertical Take-off and Landing (VTOL) autonomous robot OS43 (Figure 1.5). The main frame is cross shaped and made by with carbon rods. This configuration makes the body weight lighter about 240 grams. The test bench has four propulsion systems. Each includes a 29g motor with magnetic encoders, a 6g-gear box and 6g propeller. The Test bench is controlled by a RS232 port. The commands are sent to motor modules by system controller using I2C bus. The system controller is a PIC 16F876 microcontroller and has a PID (Proportional Integral and Derivative) regulator capable of open closed loop operation in position, speed or torque control. The MT9-B8 IMU9 is used for

estimating 3D orientation data and a Kalman filter is used to get stable acceleration and angular velocity data. A MATLAB simulation is used before the real system realization beforehand. Task of the controller was to stabilize the height with compensating the initial error of the roll, pitch and yaw angles. The real system has some problems sourced from undesired but unavoidable delays and the saturation of actuator. RS232 communication and actuator time constant are the main reason of the delays. MATLAB Simulink discrete-step delay blocks were used for simulate delays in the feedback loop and on the actuators (Kivrak A.Ö, 2006).

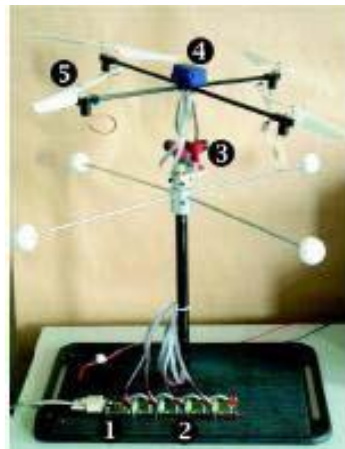


Figure 1.5 Quadrotor designed in Swiss Federal Institute of Technology.

In Pennsylvania State University two different studies had been done on quadrotors. First (Figure 1.6.) is a master thesis that had been done about a quadrotor test bench. The inertial measurement unit consists of three analog devices gyros ADXRS150EB, one accelerometer ADXL210EB. Attitude of the quadrotor is controlled with PI control law.

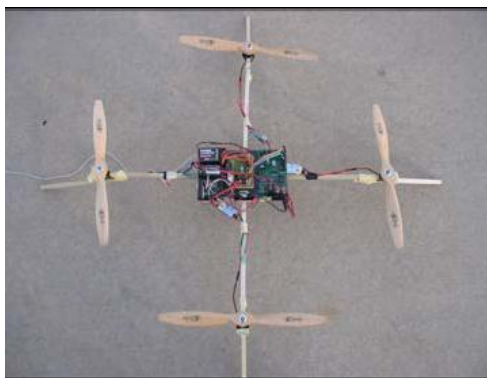


Figure 1.6 Quadrotor designed in Pennsylvania State University.

The second work done in University of Pennsylvania (Figure 1.7.) utilizes Draganflyer as a testbed. It has external pan-tilt ground and on-board cameras in addition to the three onboard gyroscopes. One camera placed on the ground captures the motion of five 2.5 cm colored markers present underneath the Draganflyer, to obtain pitch, roll and yaw angles and the position of the quadrotor by utilizing a tracking algorithm and a conversion routine. In other words, two-camera method has been introduced for estimating the full six degrees of freedom (DOF) pose of the helicopter. Algorithm routines ran in an off board computer. Due to the weight limitations GPS or other accelerometers could not be add on the system. The controller obtained the relative positions and velocities from the cameras only.

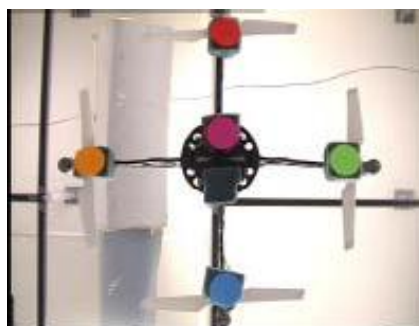


Figure 1.7 Quadrotor tracking with a camera (Erginer B., 2007).

Different kinds of air vehicles are subjected to academical researchs. For example tri-copter is one of them. The study done in Linkopings University interested in motions and dynamic equations of vehicle. Then, control algorithms are discussed

and controller is simulated. The model of Linkopings University is shown in Figure 1.8.

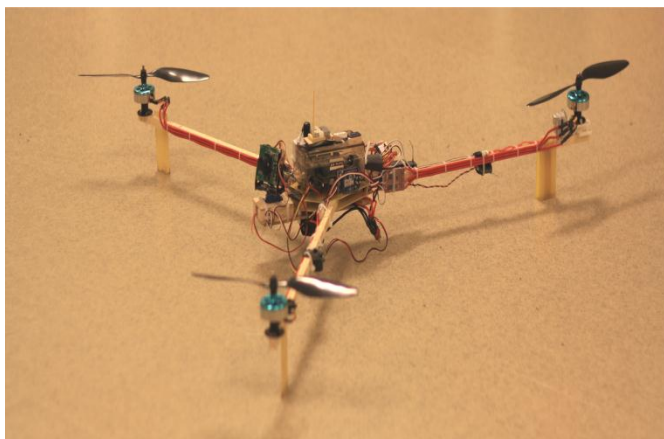


Figure 1.8 Linkopings University's Tri-copter.

In the study, a small model is used. There is a brushless motor at each arm. It is mounted to Y-shaped structure and the angle between two arms is 120. Each arm is 0.5 m. Atmega 2560 microprocessor is used for this study. And system also consist of an orientation sensor, barometer, sonar, global positioning sensor (Barsk K.J., 2012).

Hexacopters are also a different kind of air vehicle. Figure 1.9 and Figure 1.10 below show commercial hexacopter products.



Figure 1.9 A Commercial Hexacopter AH680-R
(<http://www.rckopter.com/index.php/hexacopter-ah680-r-1.html>, 2012).



Figure 1.10 A Commercial Hexacopter
(<http://www.aibotix.com/industry.html>, 2012)

Octocopters are also preferred and studied type of air vehicles. They have eight rotors and good flight stability but their maneuver capabilities are low compared to the ones that have fewer rotors. Figure 1.11 shows an octocopter design.



Figure 1.11 An Octocopter Design.
(http://gallery.mikrokoetter.de/main.php/v/Nachbau/Okto1_003.JPG.html, 2012).

CHAPTER TWO

QUADROTOR FLYING ROBOT SYSTEMS

2.1 Quadrotor Flying Robots

A Quadrotor Flying Robot is a multicopter air vehicle and uses four rotors and propellers. Such a configuration is very popular among the other rotor configuration such as three, six and eight rotors structures. A quadrotor UAV has high maneuverability, and the potential to hover and to take off, fly, and land in small areas, and can have simple control mechanisms. However, damping is very low, so for stable flight action a stability system is needed (Altug E., Ostrowski, J.P., Mahony R., 2002). In Figure 2.1, some quadrotor models are given.



Figure 2.1. Quadrotor models

The actuator system of quadrotor flying robot consists of four electric motors driving four rotors placed in the four corners of a planar square, those ones placed oppositely rotate in the same direction, while the perpendicular ones rotate reversely. A revolution or a torque control can be realized in the ensemble of a motor and the rotor mounted on it to ensure simple and efficient actuation, which can practically be realized within a local control loop.

Generally two general categories are fixed wing and rotary wing. Rotary winged types are superior when compared to fixed types in terms of achieving higher degree of freedom, low speed flying, more stationary flights, and for indoor operations.

The Quadrotors are preferred over manned aircrafts with some criterias:

Surveillance: Surveillance for military and civilian applications requires some automatic processes. Here, a quadrotor vehicle has visual and radar sensors to detect activities on a viewable area. A high resolution camera which is mounted on a quadrotor vehicle especially is operated on low light conditions with thermal imaging features. Also, some radar devices can be used for this purpose. This surveillance information can be processed automatically on quadrotor's own microcontrollers or transferred to an air-base for further evaluation.

Health Risk Conditions: Monitoring the environment which has chemical or nuclear contamination require unmanned air operations and sensor informations transferred wirelessly to operation center without having a health risk.

Dangerous Flying Conditions: Extreme weather conditions make harder to flying safely in both military and civilian purposes. A Quadrotor reduces to danger in such conditions and relatively low cost air flying hardware can be discarded.

Researches: Quadrotor Flying Robots are used in research and development for the aviation field. A miniature scaled of manned air vehicles can be tested in air without risking a human life. Especially, self- balancing in extreme weather conditions such as air gaps and wind anomalies can be analyzed and improved.

Environmentally Critical Roles: Quadrotor Flying Robots are usually smaller than manned aircrafts. They consume less power and produce lower emission and noise. Also Quadrotor Flying Robots induce less environmental disturbance and pollution.

Economic Reasons: UAVs (Unmanned Air Vehicle) and Quadrotor Flying Robots have lower costs than manned aircrafts. First cost, operation cost, maintenance costs, hangarage cost are all less. Labour costs of operators and insurance are also cheaper (Austin R., 2010).

With an aim to achieve autonomous flight and long flight time, mass and power distribution is important for a Quadrotor Robot. Figure 2.2 shows mass and power distribution of a Quadrotor Robot.

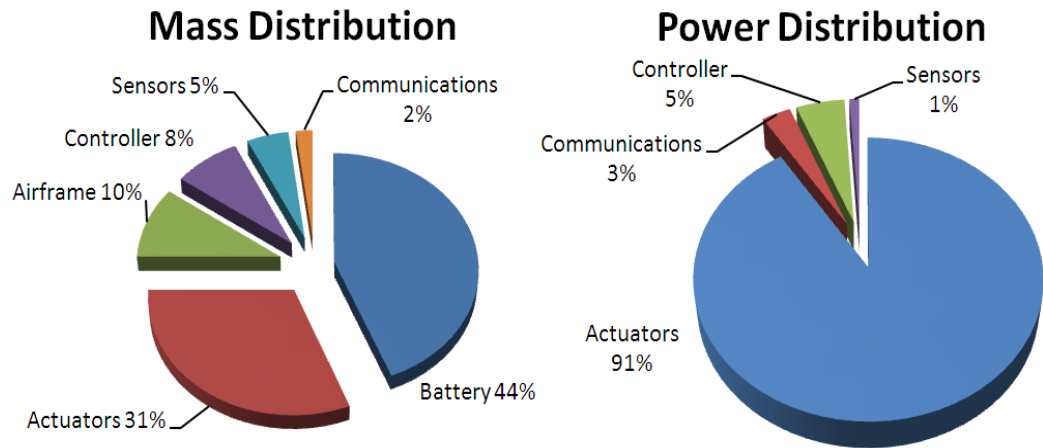


Figure 2.2 Example mass and power distributions of a quadrotor (Raza S.A., 2010).

Battery and actuators sum weight about %75 of the system's mass and % 90 power of the system is consumed by actuators. The electronic systems are light weight and consume low energy when compared to the mechanical parts.

The figure below shows the general logical structure of quadrotor robots. As seen below, the most important unit is the main control unit. All remote commands from Wireless Communication Receiver evaluated and the result output signals transmitted to the motors by ESC (Electronic Speed Controller) units. Control unit receives all sensor data and converts them to comprehensible forms, commands the ESCs to drive motors, communicates with stations. Power Supply control unit distributes the power to the control board, sensors, motor controllers and motors. Also, Global Positioning module can be added. Logical schematic of general-purpose quadrotor robots is shown in Figure 2.3.

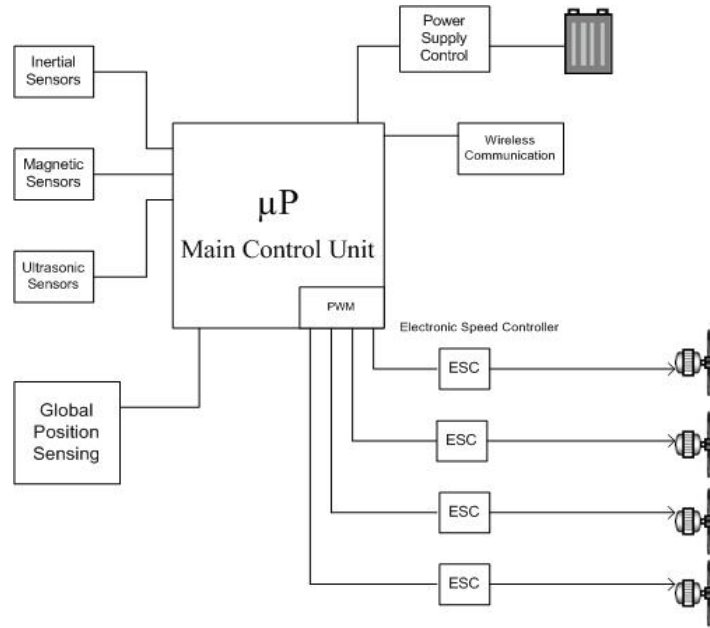


Figure 2.3 Logical structure of quadrotor robots.

2.1.1 Quadrotor Robot's Dynamics

A mathematical model has to be derived to obtain a quadrotor system's behaviors. Quadrotor robot's behaviors are defined by velocities of each motors and state of relational velocities. Therefore, position and orientation data can be obtained according to input values. Later, any control algorithm can be applied to the motors to get desired motions. Some physical definitions of a quadrotor system are shown in Figure 2.4.

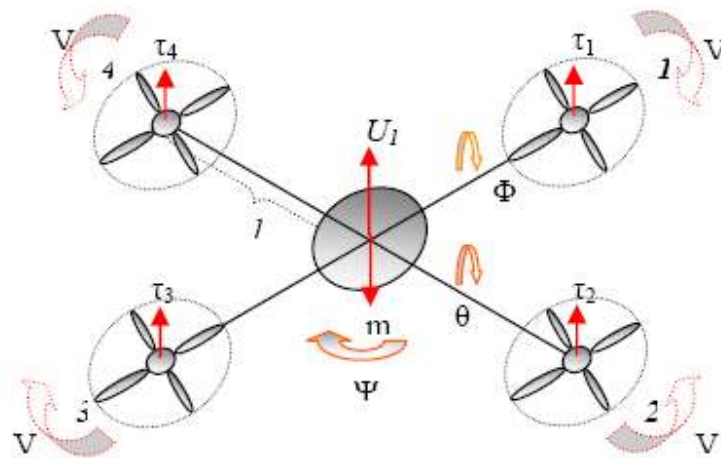


Figure 2.4 Quadrotor (Dikmen İ.C., Arısoy A.,& Temeltaş H., 2010).

By change of propeller's revolution, torques and moments are produced. Let V_i ; is i'th propeller's velocity. τ_i is thrust. So total thrust expression effect the system:

$$U_1 = \tau_1 + \tau_2 + \tau_3 + \tau_4$$

Roll moment is produced by the change of 2 and 4 number propeller's revolution:

$$U_2 = l(\tau_4 - \tau_2)\tau_4$$

Pitch moment is produced by the change of 1 and 3 number propeller's revolution:

$$U_3 = l(\tau_3 - \tau_1)\tau_4$$

Yaw moment is produced by the change of odd and even number propeller's revolution:

$$U_4 = (\tau_1 - \tau_2 + \tau_3 - \tau_4)$$

Also:

$$\begin{aligned} T_i &= b\Omega_i^2 \\ D_i &= d\Omega_i^2 \end{aligned} \quad b, d \text{ constant}$$

So, all moments affected the system are:

$$U_1 = l(\Omega_4^2 - \Omega_2^2)$$

$$U_2 = l(\Omega_3^2 - \Omega_1^2)$$

$$U_3 = (\Omega_1^2 - \Omega_2^2 + \Omega_3^2 + \Omega_4^2)$$

$$\Omega_r = \Omega_1 - \Omega_2 + \Omega_3 - \Omega_4$$

$U_{1,2,3}$ are control signals (Dikmen İ.C., Arisoy A., & Temeltaş H., 2010). Dynamic model formed through Euler-Lagrange approach with following assumptions:

- Structure is rigid.

- Structure is symmetrical.
- Mass center and body center are in same axis.
- Propellers are rigid (Dikmen İ.C., Arısoy A., & Temeltaş H., 2010).

Let us write Lagrangian derivatived movement equation:

$$L = T - V$$

$$\Gamma_i = \frac{d}{dt} \left(\frac{\delta L}{\delta \dot{q}_i} \right) - \frac{\delta L}{\delta q_i}$$

Here $\dot{q}_i$ are generalized coordinates and Γ_i are generalized forces. So movement equation becomes:

$$I_{xx} \ddot{\phi} = \dot{\theta} \dot{\phi} (I_{yy} - I_{zz}) + J_R \dot{\theta} \Omega_r + U_1$$

$$I_{yy} \ddot{\phi} = \dot{\theta} \dot{\phi} (I_{zz} - I_{xx}) + J_R \dot{\phi} \Omega_r + U_2$$

$$I_{zz} \ddot{\phi} = \dot{\phi} \dot{\theta} (I_{xx} - I_{yy}) + U_3$$

Table 2.1 shows the descriptions of the symbols written in above equations:

Table 2.1 Symbols and descriptions

Symbol	Description
Φ	Roll Angle
θ	Pitch Angle
ψ	Yaw Angle
τ	Thrust
Ω	Propeller Rotation
m	Mass
L	Arm Length
b, d	Constants

2.1.2 Quadrotor Robot's Motions

Basic motions of quadrotor robot are shown in the Figure 2.5.below.

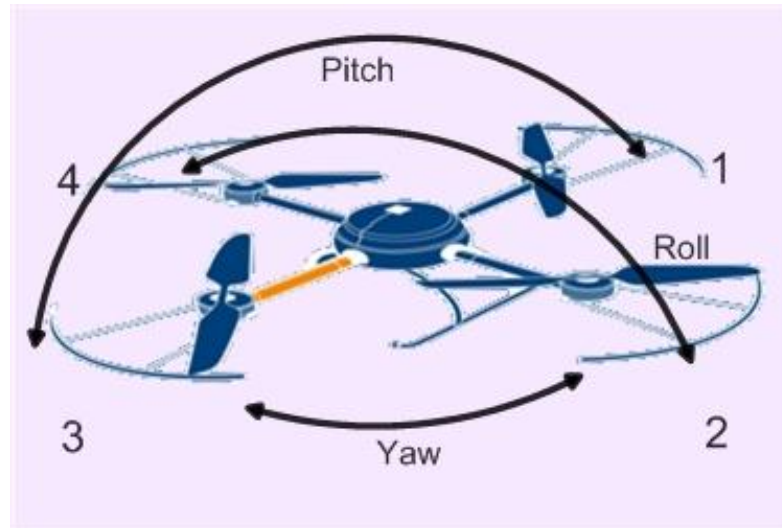


Figure 2.5 Quadrotor Motions.

Here the rotor number 1 is accepted as front rotor and rotor number 3 as tail rotor. Motions can be defined as follows:

Pitch motion: It is the nose up and down movement of the air vehicle. Pitch angle symbolized as θ . Pitch motion is shown in Figure 2.6.

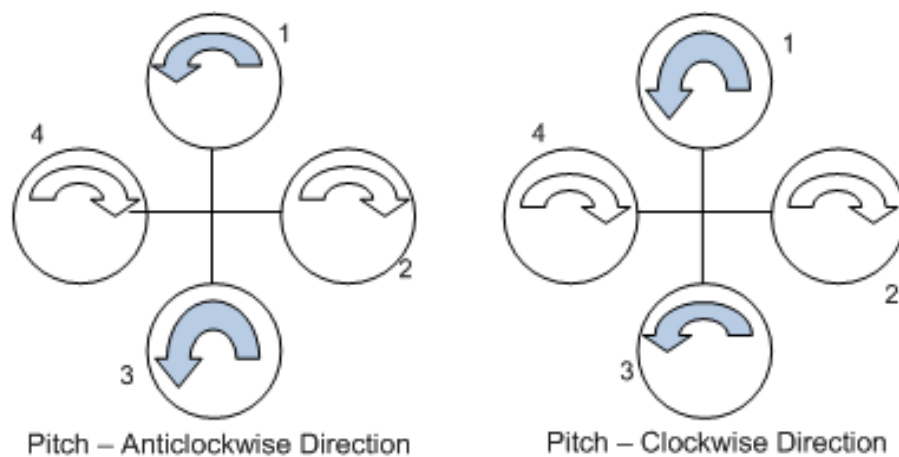


Figure 2.6 Pitch.

While the rotor set (2,4) turn clockwise direction, speed difference between anti-clockwise rotating rotor set (1,3) determines pitch up or down movement.

Roll motion: It is the roll left or roll right movement of the air vehicle. Roll angle symbolized as Φ . While the rotor set (1,3) turn anti-clockwise direction, speed difference between clockwise rotating rotor set (2,4) determines roll left or right movement. Roll motion is shown in Figure 2.7.

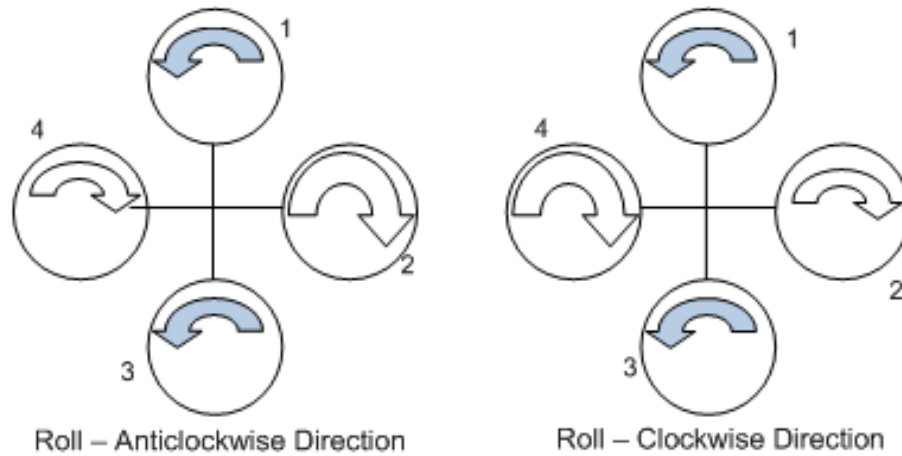


Figure 2.7 Roll.

Yaw motion: Yaw movement is simply deviation from route of air vehicle. Yaw angle symbolized as ψ . While the rotor set (1,3) turn anti-clockwise direction and rotor set (2,4) turn clockwise direction, increasing or decreasing the speed of rotor set (1,3) determines yaw movement. Yaw motion is shown in Figure 2.8.

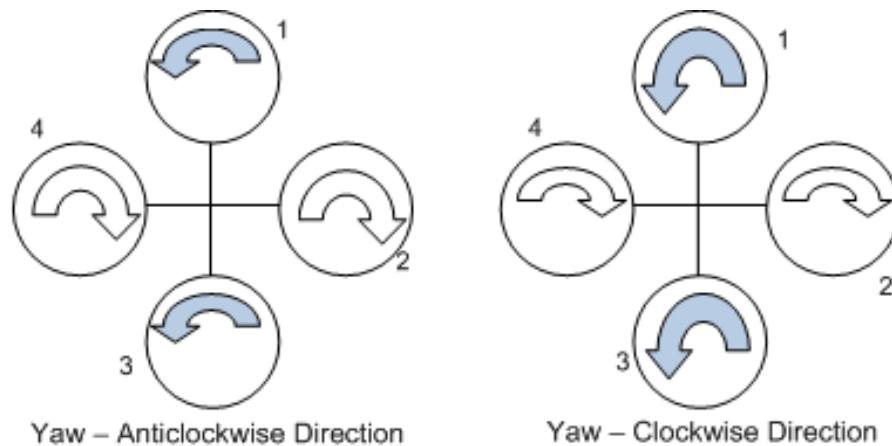


Figure 2.8 Yaw.

Take-off and Land motions: Quadrotor flying robots take off and land vertical. So, total thrust must be perpendicular to gravity force. While the rotor set

(1,3) turn anti-clockwise direction and rotor set (2,4) turn clockwise direction, take off or land can be managed. Take-off and Land motions are shown in Figure 2.9.

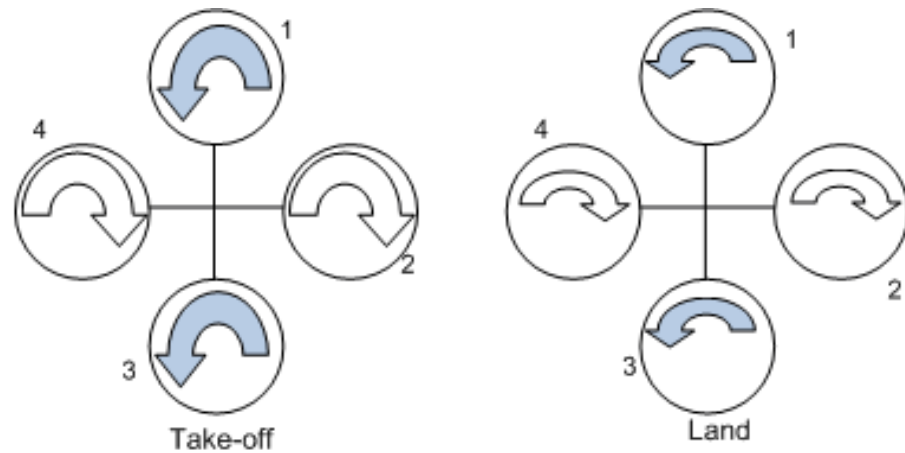


Figure 2.9 Take-off and Land.

2.2 Components of Quadrotor Robots

2.2.1 Frame

Quadrotor Robot's frame design has an importance. With a poor design, robot can't get motor's efficiency. The robustness, weight and cost are critical design factors.

Aluminum, historically, has been the material of choice for RC helicopters. Aluminum is light and strong, dissipates heat well, and is relatively inexpensive in comparison to some of the other possibilities. The negative for aluminum is that it tends to be too heavy for small aircraft models. Also, aluminum can develop cracks over time from vibrations. Plastic absorbs vibration much better the previously mentioned aluminum. Also, it is fairly durable and will return to its original shape if bent. Plastic is also very inexpensive, light and can be machined very easily. Various types of plastics were explored, including Nylon, Polypropylene, Delrin, Ultem, polyethylene. The loads placed on these parts are well within the yield strength of the materials, so ultimately the decision came down to price, and raw Nylon was the cheapest plastic available.

Carbon Fiber is currently the best material available for RC helicopters. It is stronger and lighter than aluminum and absorbs vibration better than plastic. It can be molded to be super stiff in one direction and flexible in the other. But, it is also much more expensive than other materials. Also, it is difficult to machine so it would require an outside source to manufacture the required parts.

2.2.2 Motors

Quadrotor Robots have 4 motors that each one has light-weight and high power. Usually two kinds of motors are preferred: Brushed Direct Current (DC) motors and Brushless DC motors.

Brushless DC motors compared to brushed motors show better torque characteristic, high efficiency, long operating life, superior features such as quiet operation. Brushless motors provide high torque at low speeds due to the properties do not need gear mechanism. Brushless and brushed DC motors are shown in Figure 2.10.



Figure 2.10 Brushless DC motor (left) and Brushed DC motor (right).

2.2.3 Main Control Units

Main control units are simply electronic control circuits. Each component of Quadrotor Robot is connected to main control unit. Sensors send the information about related measurement quantity. For example, Global Positioning Sensor sends location values, ultrasonic distance sensor records altitude data. Wireless communication unit also transfers and receives control orders. Towards these orders, main control unit drives the actuators via electronic speed controller.

Camera data is also processed in main control unit. The Quadrotor Robots with integrated camera or trajectory follower Quadrotor Robots need additional memory for recordings.

Generally this unit is realized through printed circuit boards (PCB). Some appliers prefer commercial ready to use controllers and some other appliers prefer to design their own controllers.

Figure 2.11 shows an ArduPilot Mega – UAV controller.

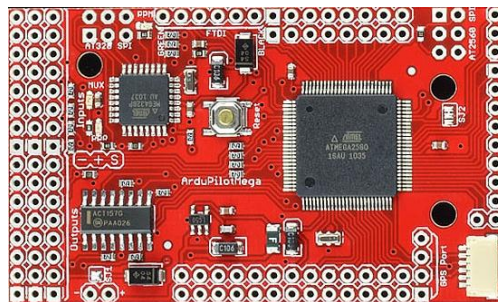


Figure 2.11 GPS-10294 ArduPilot Mega UAV Controller

(<https://www.sparkfun.com/products/10294>)

ZeroUAV YS-X6 autopilot is also a flight controller system. It supports Android/IOS and PC systems, provides auto-navigation, target lock, self-leveling and position/altitude holding. It can be used in professional and hobby applications. Components consist of main controller, inertial measurement unit, global positioning Sensor (GPS), WiFi, damper pad and cables. All components weight 212 grams. Operation voltage is between 3.7 V and 22.2 V. Figure 2.12 shows ZeroUAV YS-X6.



Figure 2.12 ZeroUAV YS-X6

(www.hobbyking.com/hobbyking/store/__25925__ZeroUAV_YS_X6_Autopilot_GPS_Flight_Control_System.html, 2012).

HobbyKing KK2.0 Control Board is another commercial flight controller with LCD. It allows total for 8 motors to be controlled. It weights 21 grams. It has 3-axis gyro and accelerometer. Operation voltage is between 4.8 V and 6 V. The system uses Atmage324PA microcontroller. KK2.0 is shown in Figure 2.13.



Figure 2.13 Hobbyking KK2.0 Multi-rotor LCD Flight Control Board

(http://www.hobbyking.com/hobbyking/store/__24723__Hobbyking_KK2_0_Multi_rotor_LCD_Flight_Control_Board.html).

2.2.4 Electronic Speed Controller

An electronic speed controller is a circuit that increase or decrease electric motor's speed. Also controller changes the direction of motors and can be used for dynamic brake. Electronic speed controllers (ESCs) are used on radio controlled vehicles.

The ESCs use input signals from the control board to provide power required to control motors. Each one has a microprocessor and own driver software. And also to realize this task, electronic switching circuits are needed.

ESCs have to suffice enough amperage to drive motors. For ESC choice, fast refresh ratings are important. Flashing ESCs that optimized for Quadrotors are also available in markets. Figure 2.14 shows commercial ESC examples.

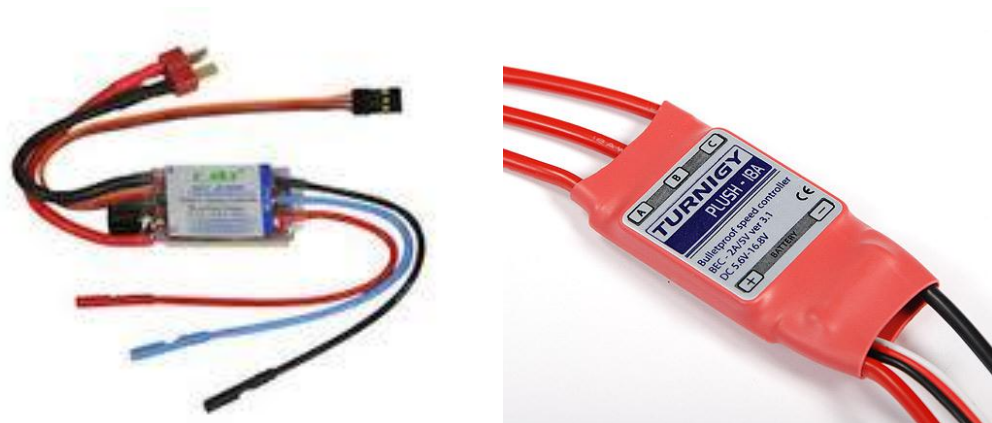


Figure 2.14 Electronic Speed Controllers (www.skyhighhobby.com/rc/all-about-speed-controls-esc, 2012).

2.2.5 Li-Po Batteries

There are different battery technologies available including lead acid (Pb), nickel-metal hydride (NiMH), lithium ion (Li-ion) and lithium polymer (LiPo). LiPo batteries are smaller than most of the battery types. A disadvantage of LiPo batteries is the requirement of the additional electronics for monitoring and charging. Multi-pack of the LiPo batteries is needed to get higher voltages but it is required to have load-balancing circuits for charging purpose. (Dubois E., Gray P., & Nigay L., 2010).

2.2.6 Propellers

The propeller is the thrust producing component on the quadrotor. Designs of the airfoil for the propeller are highly studied and coveted engineering secrets and have been for more than a century. Figure 2.15 shows an example propeller.



Figure 2.15 Quadrotor Robot Propellers.

Over the years many classes of airfoils have been developed. Each design created for different applications, producing different flight specifications. For the RC quadrotor application there is only one class, the small-scale fixed-pitch rotor blade. The miniature quadrotor most commonly uses a twin-blade propeller for each of its motors. The twin-blade design is light and provides excellent mission endurance. Assuming compressibility effects are negligible the best airfoils have a high-aspect ratio, which increases lift. In aerodynamics however, maximizing lift is only as important as minimizing drag. A high lift-to-drag ratio is desirable. There are two types of drag, known as profile drag and separation drag that must be minimized. Profile drag is reduced by decreasing the camber and reducing the cross-section. To reduce separation drag the trailing edge of the airfoil must be sharp, which creates a thin, highly acute angle. An additional consideration is the moment force on the airfoil. A high moment force can increase wear, create unwanted vibrations, and increase drag. To reduce the moment, the center of pressure must be moved rearward. One can accomplish this by shifting the center of thickness, however this decreases lift. Consequently, there is a trade-off between minimizing the moment force and maximizing lift. Another way to reduce moments is by increasing the mechanical stiffness, which reduces angular deflection (Dupuis M., Gibbons J., Hobson-Dupont M., Knight A., Lepilov A., Monfreda M., & Mungai G., 2008).

2.2.7 Sensors

As well as in many control systems, in Quadrotor Robot applications sensors have great importance. They provide fast and accurate reactions against external environment conditions.

Various sensors are used in Quadrotor applications. In in-door applications Quadrotor needs following items:

Inertial sensors, Micro-Electromechanical System (MEMS)-based accelerometers, angular rate sensors (providing measurements 6 degrees of freedom), ultrasound or infrared-based altitude measurement device, an indoor positioning system, velocity and orientation measurements.

In out-door applications Quadrotor needs following items:

Inertial sensors, MEMS-based accelerometers, angular rate sensors (providing measurements 6 degrees of freedom), barometric pressure or laser-based altitude measurement device, magnetometer-based orientation measurement system, global positioning sensor (GPS) and velocity measurements (Soumelidis A., Gaspar P., Bauer P., Lantos B., & Prohaszka Z.).

2.2.7.1 Inertial Measurement Unit (IMU)

First in 1949 C.S.Draper applied IMU. Afterwards, IMUs have become common navigational component of ships and aircrafts. Historically an IMU worked on itself and need no references to provide estimation. However the term has changed, now IMU term is also used for external referenced systems.

IMUs have two basic kinds; gimbaled systems and strap-down systems. To provide a stable platform for measurements, gimbaled IMUs are mounted within complex gimbal structures. Gyroscopes are used for confirming initial reference frame. Strap-down IMUs connected to vehicle directly. There is no need any transformation.

In either case estimating the motion requires integrating information from the sensors within the IMU (accelerometers, gyroscopes, etc.) in real time. This was a time consuming calculation in the past, prior to 1970s. Because of this gimbaled IMU was more popular. Today, such a calculation is no need more effort and cost.

A standard IMU has six degree of freedom (DOF): Position (x, y, z) and orientation (roll, pitch, yaw). IMU-like systems also provide some estimations such as attitude and heading reference systems (HRS). They operate like IMU but provide less information about vehicle's state.

Commercial IMUs also maintain estimates of velocity and acceleration (Castillo P., Lozano R., & Dzul A.E., 2005).

IMUs are very sensitive to measurement errors that usually sourced from gyroscope and accelerometer errors. Repetitively collected gyroscope data drifts in time and as a result estimation of the orientation data can be false interpreted. Also, for getting the position information twice integrated accelerometer data leads quadratic errors. Thus, drift errors are very basic for any IMU, because the gravity vector effects can't be eliminated. For that reason, some extra information is needed to correct these drift errors. For example, GPSs can be used for correction of this long period of measurement operations.

2.2.7.2 Global Positioning System (GPS)

GPS was invented in United States first. It named as NAVSTAR. Until 1982 GPS was only for military usage. After 1982 it was available for commercial and private use.

Selected four satellites of nominal 24 satellites transmit signals to the receiver GPS. This receiver calculates the location using signals.

The satellites have an atomic clock. They propagate radio signals. Propagated signals contain the time at the start of the signal. Then receiver GPS calculates the range from each satellite through arrival time. Radio frequencies vary between 1.1 GHz and 1.6 GHz in L band.

GPS provides two capability; Standard Positioning System (SPS) and Precise Positioning Service (PPS). SPS is designed for civil usage and PPS is for military usage. These two services provided by every satellite.

The SPS works with unencrypted codes at L1 frequency and gives horizontal position in 10 meters accuracy.

The Differential GPS (DGPS) improves GPS services using fixed network. These earth-based stations broadcast the difference between known fixed location and transmitted location by satellites. Then the receivers use these differences to correct their data. DGPS reduces the errors about 20 cm per 100 km. The air vehicles can use their base station as a reference station.

A GPS example module is shown in Figure 2.16.



Figure 2.16 GPS Receiver UP-501
(<https://www.sparkfun.com/products/10702>, 2012).

GPS also helps in a ‘dead reckoning system’ to derive a component of smoothing to the raw GPS. This combination needs a filter such as a Kalman filter. The filter also gives an element of modeling of measurement errors (Austin R., 2010).

GPS’s are used in quadrotor robot applications and other outdoor robot applications. They provide reliable information about robot’s location.

2.2.7.3 Ultrasonic Distance Measurement

Ultrasound is an acoustic wave which its frequency is higher than audible range of human ear, 20 kHz. Some animals can hear ultrasounds like bats.

Sir Francis Galton made first known experiments in the years around 1883. Many years later people applied ultrasound technology into engineering, medicine and other daily life activities.

The most known technique of ultrasound is *time of flight technique* (Pedersen O.T., Karlsson N., 1999).

The pulse-echo method is an example. In the pulse-echo method, a ultrasonic sound transmitted to an object. When the pulse reaches an object, it is totally or partially reflected, and the elapsed time from starting time is measured. This time depends on velocity of the sound and the distance between objects. The velocity of the sound is c , the time difference is t . So the distance is:

$$d = \frac{ct}{2}$$

The figure 2.17 shown below is a basic pulse echo system. The transmitter and the receiver are may be included in a device; they are here below separated for simplicity.

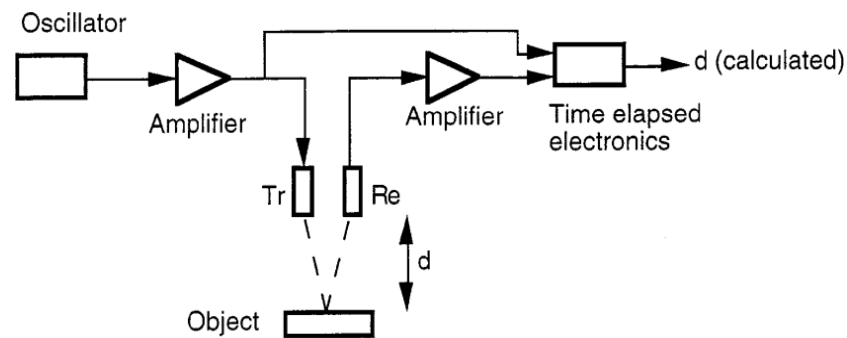


Figure 2.17 Simple Pulse-echo system (Pedersen O.T, Karlsson N., 1999).

Also in Quadrotor Robot applications, ultrasonic distance measurement is needed. Especially during the flight, ultrasonic distance measurement is important to obtain altitude data. In Figure 2.18 an ultrasonic sensor is shown.



Figure 2.18 Ultrasonic Sensor
(Parallax Inc. *PING*
#28015, 2009).

2.2.7.4 Pressure Altimeter Sensors

Pressure altimeters use the atmospheric pressure and height relation. Atmospheric pressure is the weight of the air surrounding the earth. And it changes by the location's height. There is an inverse proportion between height and atmospheric pressure in a specified linear area.

Pressure altimeters are used in many fields such as avionics, hiking, climbing, off-road vehicles. In quadrotor Robot systems, pressure altimeters work sometimes with ultrasonic sensors. These two sensors determine the height of quadrotor robot.

With the development of sensor technology, also MEMS-based pressure altimeters are preferred.

A commercial pressure altimeter sensor is shown in Figure 2.19.

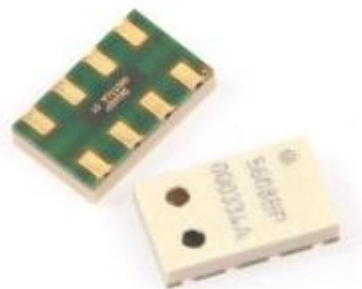


Figure 2.19 Micro Altimeter
Pressure Sensor MS5607
(<http://www.jameco.com/Jameco/Products/ProdDS/2150230.pdf>, 2012)

2.2.7.5 Camera and Vision Sense

The use of camera in Quadrotor Robots is very common. Implementation of camera to Quadrotor Robots brings with possible problems. For getting correct and stable camera view, you have to stabilize the camera or use proper software. To make this stabilization by mechanical ways, 2 servo motors are needed. A Hexacopter is shown in Figure 2.20.



Figure 2.20 Hexacopter Draganflyer with camera.

The camera in Quadrotor robots is used for recording, self-stabilization and also take-off and landing objectives. For example; In Traffic flow control, save & rescue actions, fire alarm systems, mapping or tracking an object etc. cameras are useful.

Video cameras are used as vision sensors. Generally, these video cameras include imaging optics, electronics and integrated signal processing hardware and software. They can store images, send images using serial, USB and wireless interfaces. They are used in industrial production control, manufacturing, fault detection of products and lately in aircraft applications.

Monochrome or color sensor cameras can be used for vision applications. Monochrome vision sensors get the image in grayscale or black and white. Color sensing vision sensors can give red, green and blue (RGB) colors which combine a spectrum of colors. Also, color cameras have grayscale, and black and white type converting software to make image processing recognitions. There are also multiple chip color in vision sensors. Each one captures a part of color image for example only one color. Then results are united to obtain full image.

Important specifications for vision sensors are frame rate and storage capacity. Here, the frame rate is measured of still images captured in one second and determines the inspection rate of the objects per second. Horizontal resolution, maximum frame rate, shutter speed, sensitivity, and signal-to-noise ratio are other important specifications.

Applications of camera usage are color recognition, image direction, object positioning, object detection, edge detection.

Charge Coupled Devices (CCD) and Complementary Metal Oxide Semiconductor (CMOS) type image sensors are very popular in digital cameras. CCD types detect electrons with a sensitive material and integrated circuits obtain the digital image. CMOS sensors principle is the same as in CCDs, but work at low voltages than as in CCDs. Therefore, the power consumption of CMOS type is very low.

The other important parameters for vision sensors are dimensions, lens types, shutter control mechanisms, sensor specifications, performance and operating voltages, and operating humidity.

Besides cameras, *video cameras* or web cameras are also used in machine vision, quality monitoring, security, and remote monitoring for industrial and commercial operations.

Video camera specifications are basically video resolution, maximum frame rate, shutter speed, sensitivity, and signal-to-noise ratio. Video resolution defines horizontal and vertical pixel counts. For example, 640×480 image size has 640 vertical and 480 horizontal pixels. Here, the ratio of vertical and horizontal resolutions called as aspect ratio and the value is 4:3. Another popular aspect ratio for today is 16:9. Usually, 320x240 or 640x480 image size is sufficient for image processing application and also reduces the processing payload. Maximum frame rate is measured as the maximum number of captured frames per second. Shutter time is a typical exposure time for getting the image light information that reflected from the object. A wide range of selection can be made for this shutter time; more exposure time means more capture information but it is much affected from camera shakes. Sensitivity is about the environment light level and it is required for getting more

detailed and brighter images. Last, signal-to-noise ratio (SNR) parameter is important to have more clear images filtered from the unwanted pixel noises.

Analog video camera formats are NTSC, PAL, SECAM, RS170, RS330 and CCIR. Digital output interfaces are RS232, RS422, RS485, parallel interfaces, Ethernet, DeviceNet, ARCNET, PROFIBUS, CANbus, Firewire, TTL, USB, and radio or wireless (Castillo P., Lozano R., & Dzul A.E., 2005).

CHAPTER THREE

TESTBED DESIGN

3.1 Testbeds for Quadrotor Flying Robot

A quadrotor testbed is intended to protect the whole system from beginner's control. Most quadrotors are especially lightweight and has powerful motors; therefore they are easy to accelerate on each direction. These unwanted direction controls can damage first rotors, motors, motor gears, ESC units, frame and finally controller board and sensors. Another purpose of using testbeds is to test the system parts individually or together. Last, many stabilization mechanical or software adjustments can be done safely.

In previous quadrotor studies, many different testbed designs are done. They have different strategies and structures for different purposes. Usually, these designs allow quadrotors to move in three dimensional spaces, but in some applications only one dimension intended to be tested.

In one of these works, Abhishek Bhargava (2008) bond their Quadrotor Robot to the fixed place in the ceiling. Also, each rotor is bond to the base for safety reasons Testbed of Abhishek Bhargava is shown in Figure 3.1.

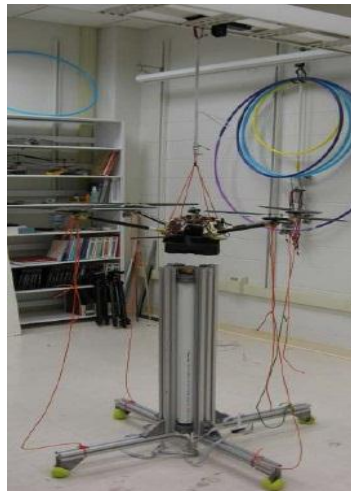


Figure 3.1 Testbed 1 (Bhargava A., 2008).

In another work, testbed is just the frame of light weight, high strength carbon fiber tubes joined by plastic L and cross joints. The resulting testbed is very light, under the payload capacity of original vehicle, but very flexible (Patel C.A, 2002). Figure 3.2 shows Patel's testbed design.

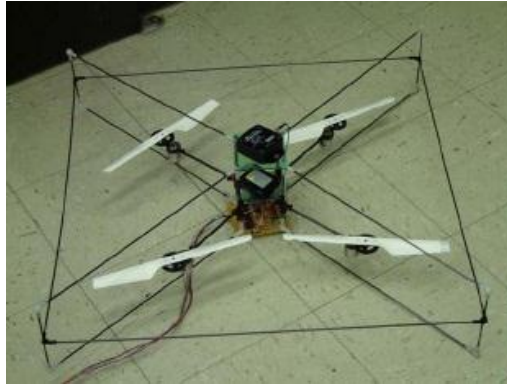


Figure 3.2 Testbed 2 (Patel C.A, 2002).

In another design, different rotor blades were tested for maximum thrust production versus power input. This testbed allows only one dimensional movement. The test was conducted for each blade at four fixed pulse width values which were 1.4, 1.5, 1.6 and 1.7 ms. Figure 3.3 shows this testbed design (DiCesare A., Gustafson K., Lindenfelzer P., 2009)



Figure 3.3 Testbed 3 (DiCesare A., Gustafson K, Lindenfelzer P., 2009).

3.2 Our Designed Testbed

In our project, it is aimed that to have a simple and low cost testbed for training the quadrotor operator. Therefore, possible damages to the important parts such as

blades, motors, and frame are prevented. Three dimensional movements are possible with this design and also suitable for the motion and stabilization tests.

First the testbed mechanics is drawn and tested in Solidworks design software. Figure 3.4 shows the testbed drawings.

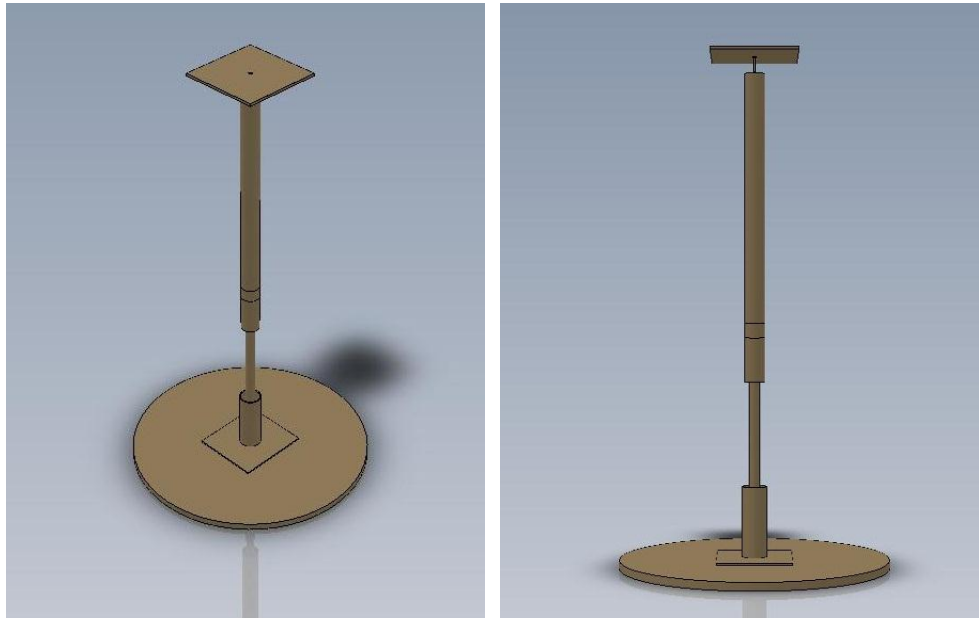


Figure 3.4 Testbed Solidworks Drawings.

The testbed was constructed using cylindrical lightweight air pump and a holder base. This designed testbed stands on a circular plate. Quadrotor Robot is fixed on a cylinder that mounted perpendicular to the plate. So, Quadrotor Robot can move easily. In Figure 3.5, lateral view of the constructed testbed is shown. Figure 3.6 shows the above view of the testbed.



Figure 3.5 Designed testbed lateral view.

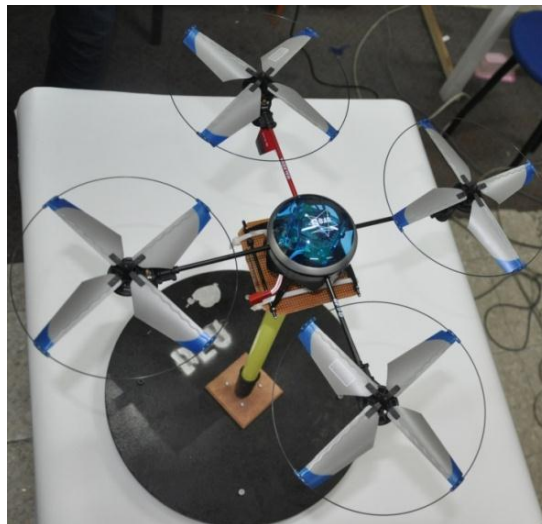


Figure 3.6 Designed testbed above view

The air pump air outlet is wide open to air to eliminate air compression resistance. But, using a different type of air pump, air outlet gap can be adjusted with a screw to change the air pump resistance. This can be useful to adjust the takeoff weight of quadrotor to simulate a payload like photo camera or video camera. Total weight of the testbed is about 1 kilogram. Cylinder upward movement length is 25 cm. Cylinder minimum lifting force is 0.4 kgf.

Because of the stabilization problems when quadrotor tries to lift testbed, metal supports are implemented to the testbed. Last version of the testbed is shown in Figure 3.7.



Figure 3.7 Last design of the testbed.

This designed testbed can move right or left side and up or down easily. Testbed can perform rolling right and left motion with almost 60 degrees. It provides enough degree of freedom for project's simulations. A testbed have to also perform up and down motions. The Figure 3.8 and 3.9 show right, left tilts and up direction motion of the testbed.

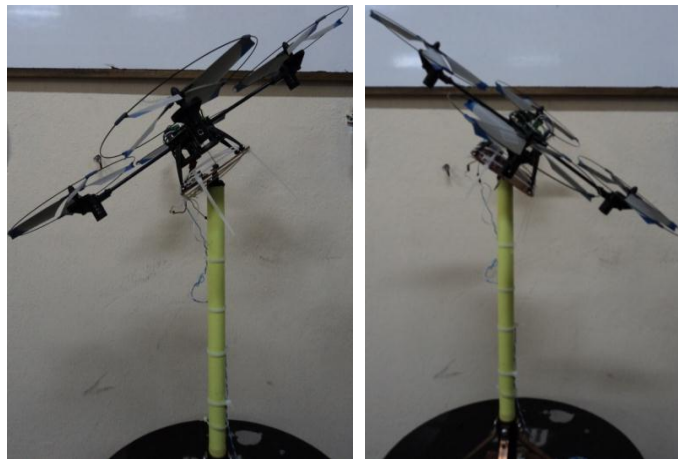


Figure 3.8 Left and right tilts of the testbed.



Figure 3.9 Testbed's up direction motion.

CHAPTER FOUR

MATLAB INTERFACE AND PYTHON SIMULATION

The most important part of this project is probably program interfaces. Because, all related works come together and visualize orientation and gyro angles. And also, filter outputs are observable in graphics.

4.1 Matlab Interface

As known, Matlab software is used various fields of signal processing and other applications.

In this part, the values sent by IMU are represented in software. Because of the swiftness of serial channel processing is not very well, the work is ended in Matlab. To work better and faster, Python software is preferred. Related Matlab codes are shown in appendices. Figure 4.1 is a flow chart of Matlab software.

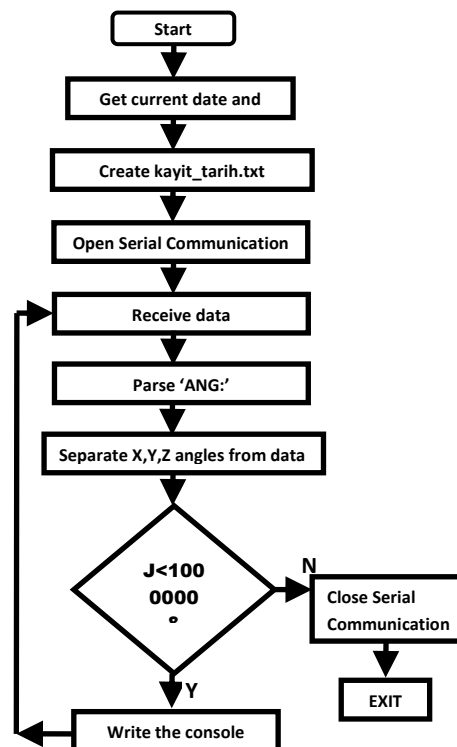


Figure 4.1 Matlab Flow Chart.

4.2 Python Simulation

Python can be used for general goals. It's a high level programming language and it has high code readability. Also Python's syntax is known as expressive, clear and simple.

Python provides multiple programming methods primarily but it is not limited object-oriented or similar to that of wide range of non-scripting contexts. Using its tools, Python code can be packaged into standalone executable programs.

In our project, in the programming side, the motions of Quadrotor robot are simulated and filtered the accelerometer values by using Python. Also manual and automatic control of stabilization is implemented. Figure 4.2 is Python software's flow chart.

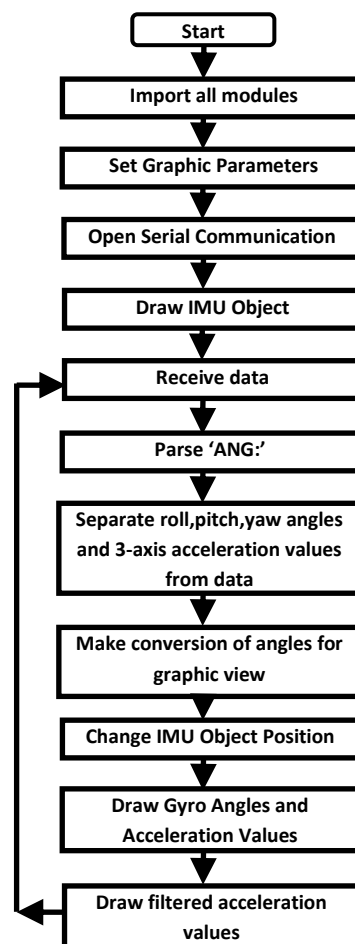


Figure 4.2 Python Flow Chart.

Figure 4.3 shows the second Python software. It explains control structure of the software.

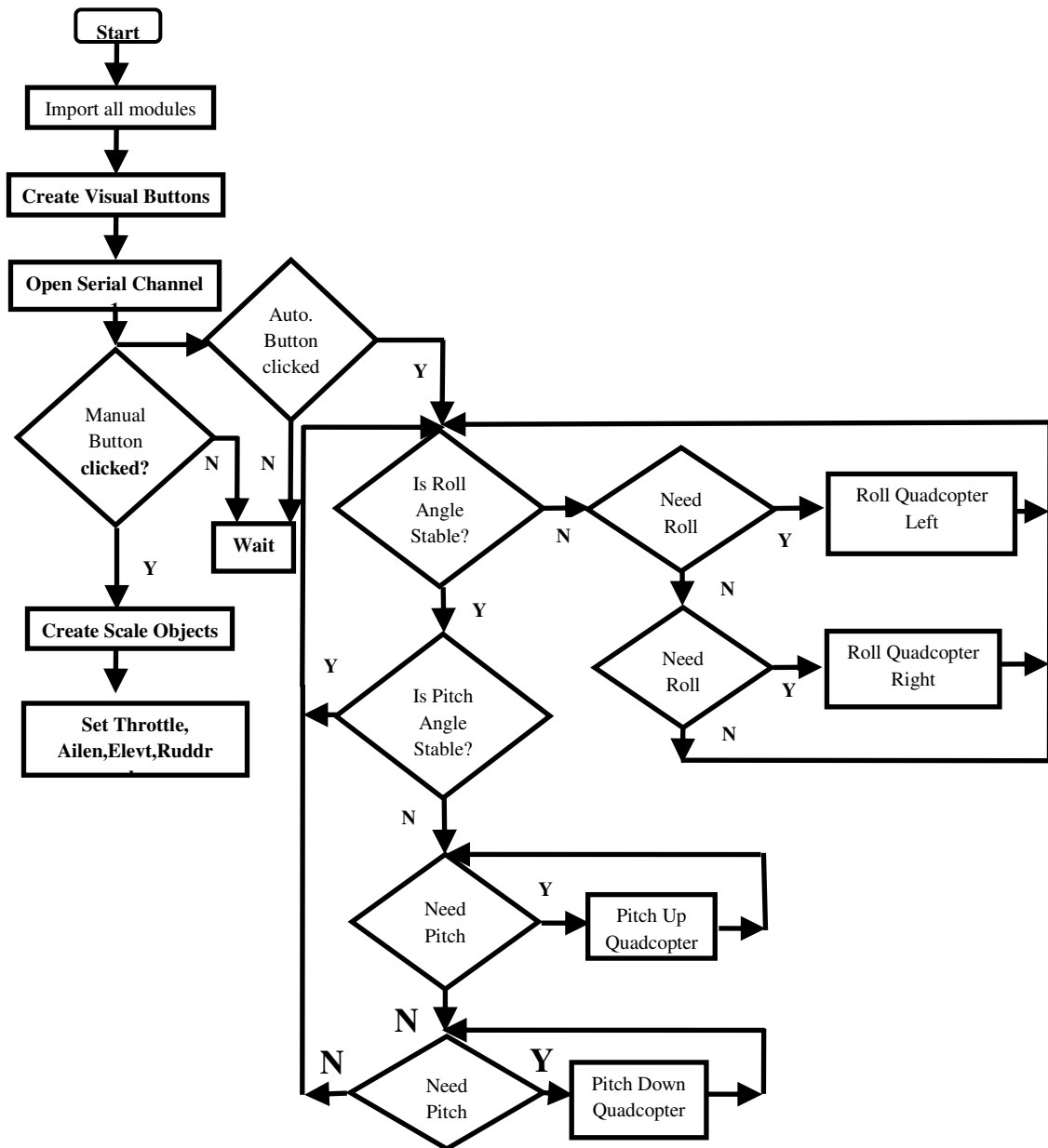


Figure 4.3 Python Flow Chart.

As seen above, there are two control type in Python simulation; Manual and Automatic. The Figure 4.4 shows the interface for manual and automatic stabilization in Python simulation.

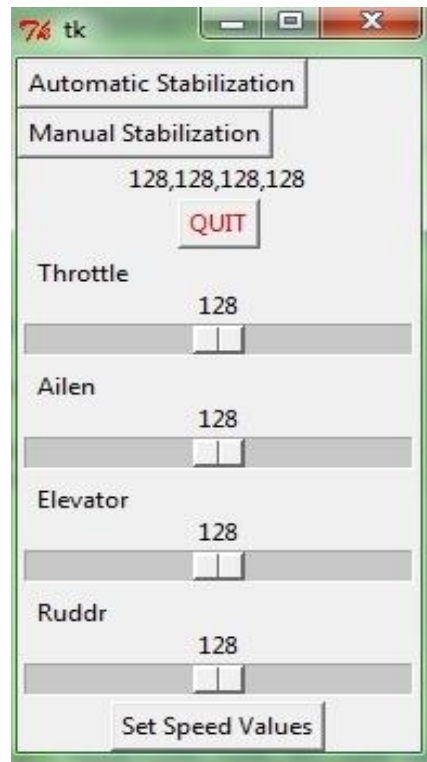


Figure 4.4 Python Interface.

The Figure 4.5 below shows the Python Simulation of Quadrotor Robot. This view represents orientation and position of Quadrotor Robot.

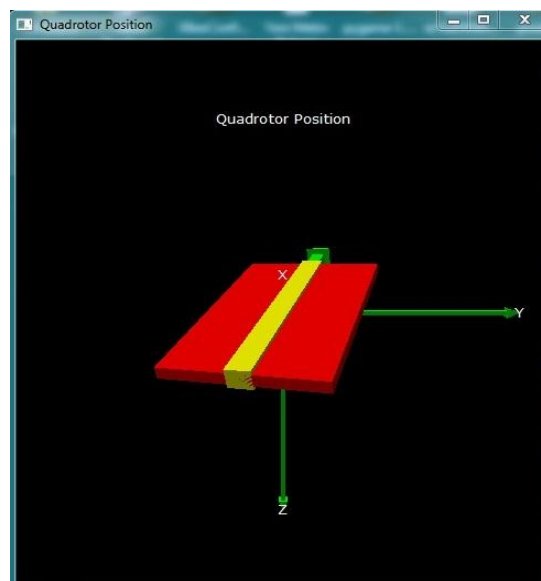


Figure 4.5 Quadrotor Position on Python.

The simulation also interests roll, pitch and yaw angles. The angular equivalents of these angles are shown and the differences are represented by graphical arrows and lines. Figure 4.6 shows these values that provided by the IMU.



Figure 4.6 IMU – Gyro Angles.

As known, the IMU provides also accelerometer values. These values are needed to be viewed in graph. Therefore, 3 axis are drawn in a graphic. Figure 4.7 shows 3-axis accelerometer values.

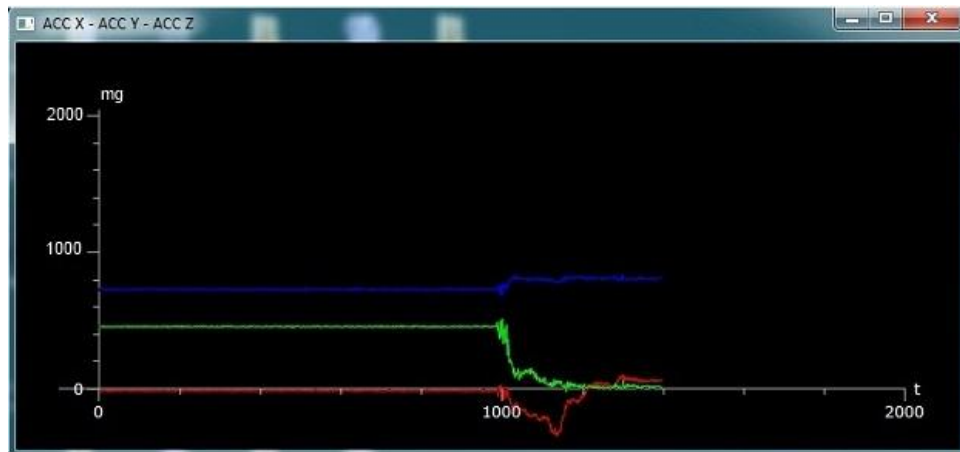


Figure 4.7 Accelerometer 3-axis values.

4.2.1 Kalman Filter

The Kalman Filter is a good estimator for a large class of systems with uncertainty and a very effective estimator for an even larger class. It is one of the most well-

known and often-used tools for so called stochastic state estimation from noisy sensor measurements. Under certain assumptions, the Kalman Filter is an optimal, recursive data processing or filter algorithm.

The Kalman filter is optimal, because it can be shown that, under certain assumptions, the Kalman Filter is optimal with respect to any criterion. For example; the mean squared error. Because of the Kalman filter uses all possible information collected, optimal performance can be obtained.

The Kalman filter is recursive, which brings the useful property that not all data needs to be kept in storage and re-processed every time when for example a new measurement arrives. Information gained in successive steps is all incorporated into the latest result.

The Kalman filter is a data process algorithm. It is useful because there is no need any other information about the system estimated. Only output and input values are enough for processing the algorithm. Filter tries to obtain the best estimate.

The Kalman filter is a feedback type algorithm. The filter estimates a signal state at a defined time and then finds feedback form of noisy measurement. The Kalman filter equations are divided into two types as time update equations and measurement update equations. The time update equations derive the priori estimates using error covariance for the next time interval. Therefore, these equations can be considered as predictor equations. The measurement update equations find a feedback for *a posteriori* estimate. Thus these equations can be considered as corrector equations (Negenborn R, 2003)

The Figure 4.8 shows deviations and Kalman estimate of an example.

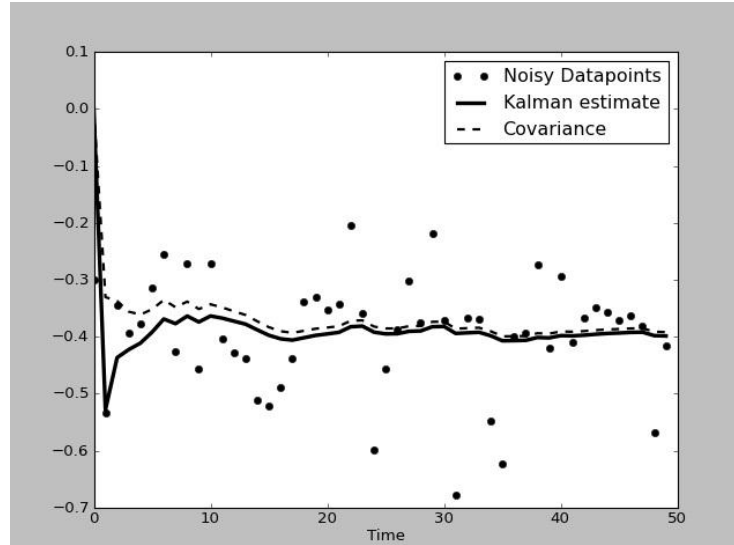


Figure 4.8 A Kalman Filter example.

4.2.2 Moving Average Filter in Python Simulation

The moving average filter is a very simple and effective filter type to eliminate noises in many digital signal processing applications. Usually it reduces high frequency random type noises on a slowly changing signals and this operation smoothes the signal and filters the unwanted high frequency changes for getting sharp step responses. This feature is usually preferred for time domain signals (Smith S.W, 2003).

The moving average filter is known also as running average filter and operates by averaging a number of points from the input noisy signal to generate each point in the output signal. In equation form, this is written:

$$y[i] = \frac{1}{M} \sum_{j=0}^{M-1} x[i + j]$$

where $x[]$ is the input signal, $y[]$ is the output signal and M is the number of points in the average.

For example, in a 5-point moving average filter configuration, point 50 in the signal is given by:

$$y[50] = \frac{x[50] + x[51] + x[52] + x[53] + x[54]}{5}$$

As an alternative, the group of points from the input signal can be chosen *symmetrically* around the output point:

$$y[50] = \frac{x[48] + x[49] + x[50] + x[51] + x[52]}{5}$$

An example noisy signal is shown in Figure 4.9. Then Figure 4.10 shows two filtered output signals. One of them is the output of 11 points moving average filter; the other one is the output of 51 points moving average filter.

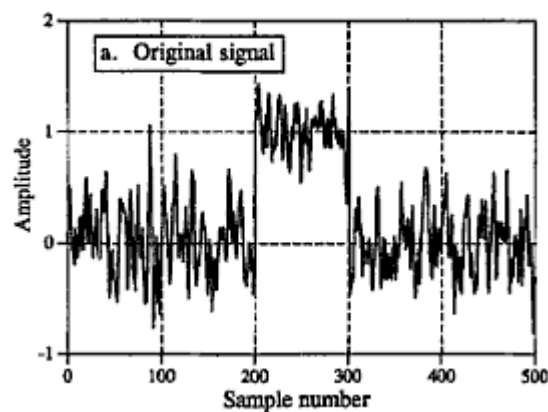


Figure 4.9 A Noisy Signal (Smith S.W., 2003).

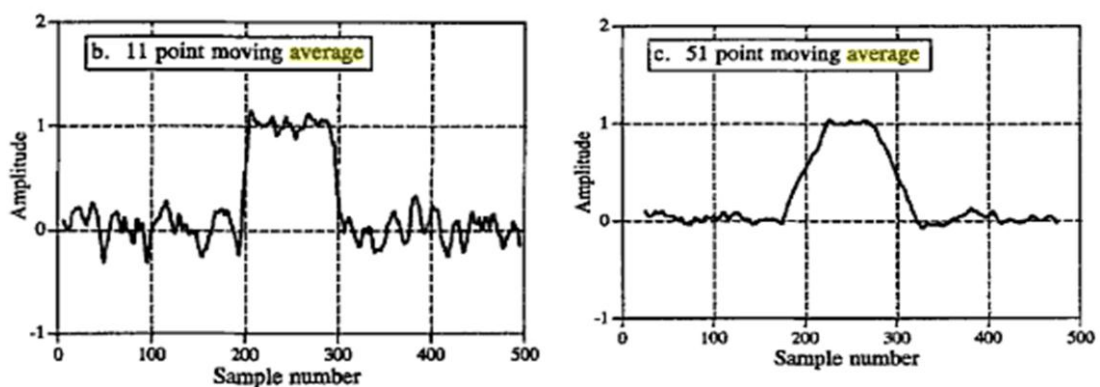


Figure 4.10 11 point moving average filter and 51 point moving average filter (Smith S.W., 2003)

When other filters are considered, the moving average filter produces the lowest noise for a given edge sharpness. The amount of noise reduction is equal to the square-root of the number of points in the average. For example, a 100-point moving average filter reduces the noise by a factor of 10 (Smith S.W.,2003).

The Figure 4.11 shows a sample Python simulation. In this figure, 3-axis acceleration values and the filtered output of these values are represented.

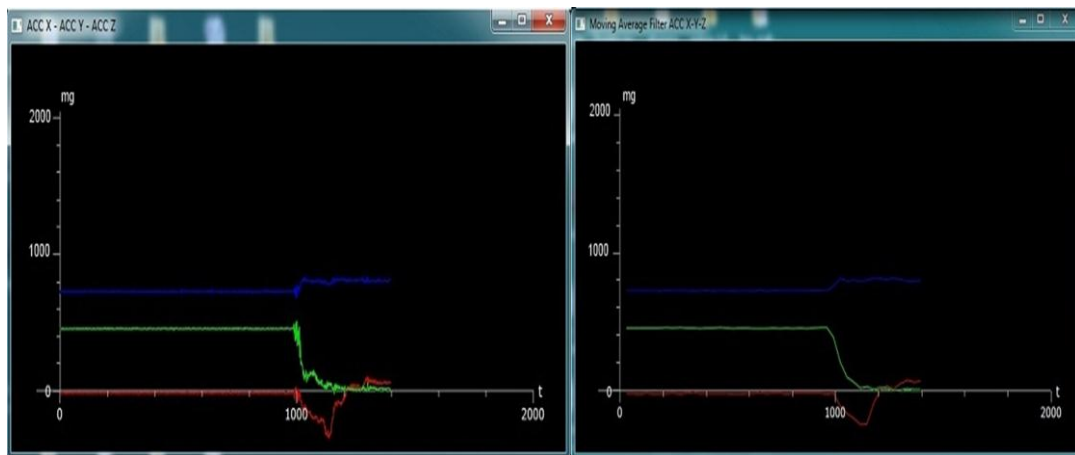


Figure 4.11 Acceleration values and moving average filtered acceleration values.

CHAPTER FIVE

EXPERIMENTAL WORKS

5.1 Walkera UFO #5

In our project Walkera Ufo #5 Quadrotor Robot is chosen. This robot flies via 2.4 Ghz Remote controller's directives. Higher models of Ufo #5 include a camera to record video. Figure 5.1 shows the Quadrotor Robot which is mounted on top of the testbed.



Figure 5.1 Walkera Ufo #5 mounted on top of the testbed.

Walkera Ufo #5 has 3-D gyro on it. It tries to handle the stabilization of system. But in this project, the own gyroscope of the Quadrotor Robot is disabled. Because, the external IMU mounted on the testbed, provides the gyro angles.

5.2 Experiment Set-up Configuration

The experiment set-up organized schematics is in Figure 5.2 below. Quadrotor Robot that is held on controlled testbed, performs through the signals come from remote controller. Also the stability angles and acceleration values are sent to computer side by the IMU and Xbee which are also placed on testbed.

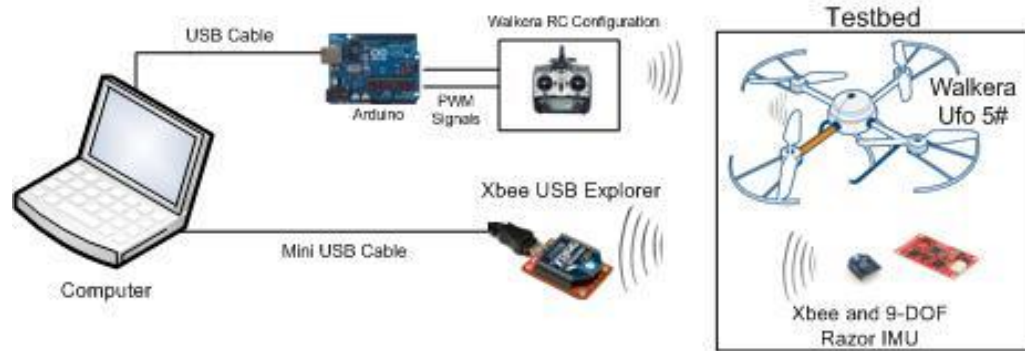


Figure 5.2 Experiment Set-up.

On the computer side, the sent data are received by Xbee module that mounted on Xbee Usb Explorer. Then that data are saved in Matlab and simulated in Python. Thus, the observable data are obtained in Python simulation.

5.2.1 Xbee Modules

Xbee is the one of the radio modules that Digi International produced. First Xbee modules named MaxStream in 2005 and they have 802.15.4 standard point to point and point to multipoint communications at over the baudrates of 250 kbit/s.

There are two models of Xbee. One of them is low power 1mW Xbee and the other one is higher power 100 mW Xbee-Pro model. Since these initial introduction models, a number of new XBee radios have been introduced. All Xbee modules can be used with at least four connections: Power, Ground, Serial In, Serial Out and additionally; reset and sleep. Additionally, most XBee families have some other flow control, I/O, A/D and indicator lines built in (Xbee RF modules datasheet, 2011). The Figure 5.3 shows a Xbee module.



Figure 5.3 Xbee Module.

Features are listed below:

- 3.3V - 50mA
- 250kbps Max data rate
- 1mW output
- Works at 100m range.
- Built-in antenna
- 8 digital IO pins
- 128-bit encryption
- Advanced Networking
- Low Power Consumption

5.2.2 *Configuring Xbee Modules*

We have to configure Xbee modules proper to current application. To configure X-CTU software is needed.

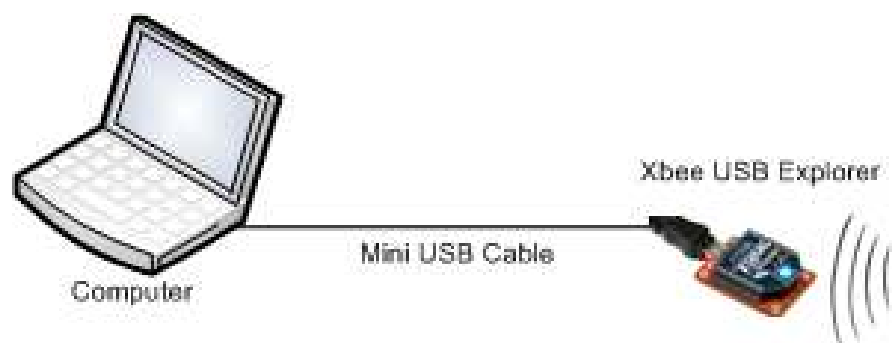


Figure 5.4 Basic Configuration.

After the configuration is set above, the parameters are arranged in X-CTU software. The Figure 5.5 shows the programming interface of X-CTU program.

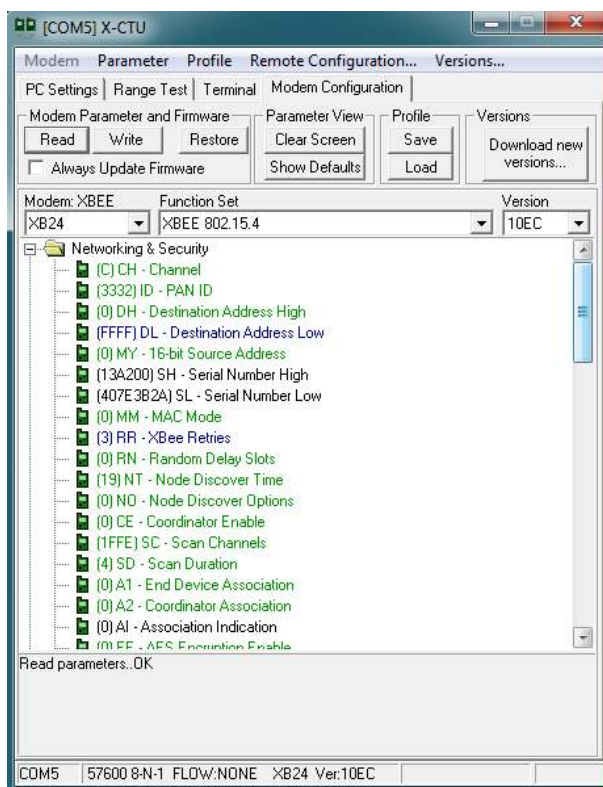


Figure 5.5 X-CTU program.

XBee parameters table is shown in Table 5.1

Table 5.1 Both of Xbee's Parameters.

XBEE - PC	
Version:	10E6
Poh ID:	3332
My ID:	0
DL ID:	FFFF
DH ID:	0
BD:	6 (means 57600 kbit baudrate)
D3:	3
IC:	8
RR:	3
R0:	10

XBEE - XB24	
Version:	10E6
Poh ID:	3332
My ID:	1
DL ID:	0
DH ID:	0
BD:	6 (means 57600 kbit baudrate)
D3:	5
IU:	0
IA:	FFFF
R0:	10

5.2.3 9-DOF Razor IMU

The 9-DOF Razor IMU works with an LY530AL (single-axis gyro), LPR530AL (two-axis gyro), ADXL345 (three-axis accelerometer), and HMC5843 (three-axis magnetometer) to realize nine degrees of inertial measurement.

The results of all sensors are processed by an on-board ATmega328. With variety of works, the 9-DOF Razor IMU can become a reference system for attitude measurement. So, 9-DOF Razor IMU is a good sensor enough for unmanned air vehicles and other stabilization systems.

The board comes programmed with Arduino bootloader and an example firmware. With a 3.3 V FTDI Basic breakout and Arduino IDE software, Razor can be programmed. For a wireless solution, it can be connected to Xbee Explorer or Bluetooth Mate.

The 9-DOF works at 3.3VDC; different voltage values are regulated down to the 3.3V. LiPo batteries are good choices (<https://www.sparkfun.com/products/9623>, 2012). The Figure 5.6 shows the Razor IMU.



Figure 5.6 Razor IMU SEN-09623
(<https://www.sparkfun.com/products/9623>, 2012).

Specifications:

- Nine-axis on a board:

- LY530ALH - 300°/s one-axis gyro
- LPR530ALH - 300°/s dual-axis gyro
- ADXL345 - 13-bit resolution, $\pm 16g$, triple-axis accelerometer
- HMC5843 - three-axis, digital magnetometer
- Atmega328 pairs up sensor outputs and send them via serial channel.
- There are autorun feature and an example firmware.
- Output pins are compatible with FTDI Basic Breakout and Xbee Explorer
- Input voltages are between 3.5 V and 16VDC.
- On-Off and Reset Switches.
- Dimensions: 49.53 x 27.94 mm.

5.2.4 *Programming Razor IMU*

A new Razor IMU works with original firmware. We have to change this firmware to get intended data. That for original AHRS and output codes are achieved and changes are made. Then for programming the IMU, the configuration is set below in Figure 5.7.

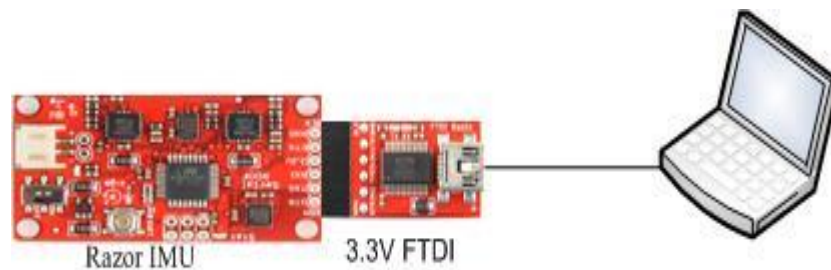


Figure 5.7 Programming IMU.

After the configuration is set, the IMU modified output codes and AHRS codes are sent by using Arduino software.

5.2.5 *Realizing of Computer Controlled Remote Controller*

As you know, Quadrotor robots are controlled by remote controllers. Remote controllers have two sticks. Stick controls and the movements are shown in Figure 5.8.

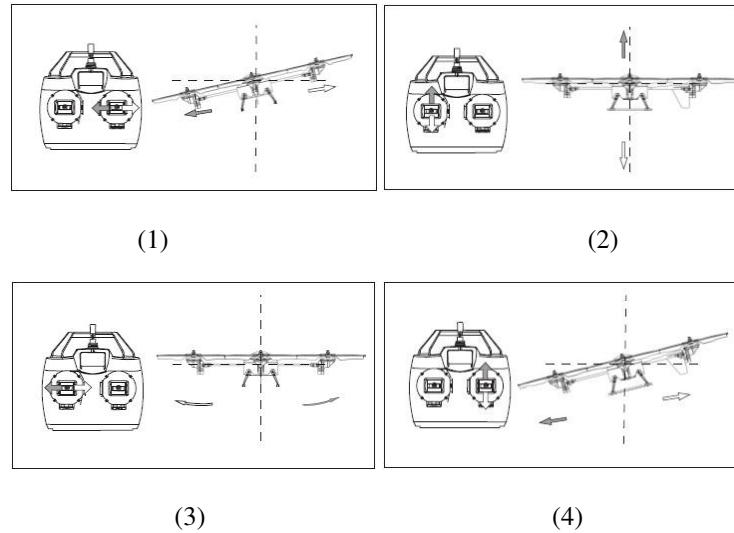


Figure 5.8 Stick controls.

1. When moving the aileron stick left or right, the Quadrotor accordingly flies left or right.
2. When moving the throttle stick up or down, the Quadrotor accordingly flies up or down.
3. When moving the rudder stick left or right, the head of the Quadrotor accordingly flies left or right.
4. When moving elevator stick up or down, the Quadrotor accordingly flies forward or backward (Walkera Ufo #5, 2010).

As part of our project, the remote controller's sticks are disabled. Instead of them our own PWM signals are produced and passed through low pass filter. The directives for producing PWM signals come from computer via serial channel. The Figure 5.9 shows computer control of the system.

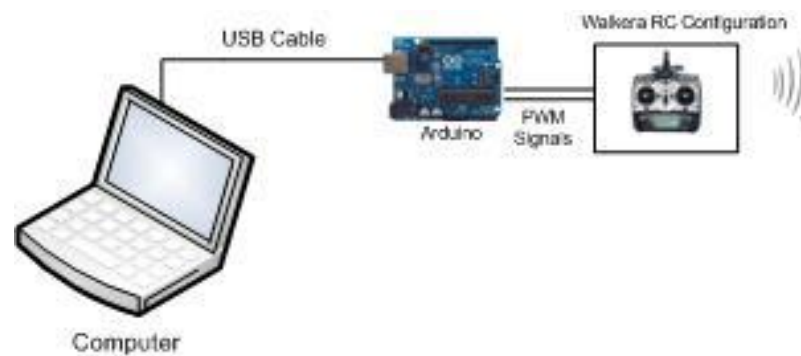


Figure 5.9 Computer Control.

The Figure 5.10 shows the new modified controller. It contains an Arduino in the case. Through the serial channel on Arduino, PWM reference values are received. Arduino software is controlled by each speed reference value like: “hiz: xxx,yyy,zzz,ttt “. Sending values must be between 0 and 255.

Arduino produces PWM signals and then the filtered signals are sent to the remote controller.



Figure 5.10 Modified Remote Controller.

CHAPTER SIX

CONCLUSIONS AND FUTURE WORKS

6.1 Conclusions

In this project, firstly the historical development of Quadrotor flying robots, applications and possible future developments were investigated. The basic elements of a quadrotor were explained individually. These components are needed to build a functional quadrotor system, and different experimental or commercial products are available in different quality and price range.

Especially, the main control board is very important for a good selection, because it has balancing and navigational sensors on it. These sensors are getting more and more advanced with the technological developments, therefore they must be updated in time.

The second important part is the ESC controller, and its' performance is very important to drive motors effectively and seamlessly. Also, the motors must be powerful enough to carry the whole frame and battery efficient for long flights. Usually, DC brushless servo motors are good selection for long service life, because they don't contain brushes that mechanically contacted. Also, these brushless motors usually don't require extra gearboxes for power increase. Every additional moving mechanical part in that kind of flying robots can be problematic in use time.

Finally, frame and propellers are needed to be built flexible as well as rigid and powerful. Here, a wide range of selections is available from plastic to fiber. Also metal frames are available, but they are not preferred so much because of their weights. Plastic ones are inexpensive, but can't be so flexible and durable. Here, the wise selection is fiber frames provide flexibility and toughness at the same time.

The other universities studies were mentioned to evaluate the preference of quadrotors and their possible study areas.

6.2 Results

Unmanned Air Vehicles are very hard to operate and requires experience to control them stable with no crash risks. Stable operation is related to self-balancing with the aid of orientation sensors and powerful control algorithm. Testbeds are ideal to learn how to fly them and balancing system development with convenient sensors and software. For that reasons, firstly a testbed designed to work safely with the Quadrotor. Quadrotor mounted on the designed testbed. And then simulation and analysis, being another stage at research, has started. Primarily, IMU sensor, assigned to send information to the Quadrotor, programmed. The wireless communication module which is cooperating with sensor was configured. Thus, the opportunity to obtain data in desired form from the sensor has been reached.

Data transmitted to computer were first acquired and inspected with Matlab. Then, to achieve terms of visual quality and speed, data migrated to Python software. Movement data from sensors are visualized with Python on a platform representing the Quadrotor robot. Also according to incoming data, 3-axis acceleration data has shown on the graph. These data are displayed in the same way in a state of running average of the filtrate.

Another phase of the project was the realization of the control of Quadrotor robot through the computer. This has achieved by an intervention to the original control of robot. On another stage of the developing Python software, Quadrotor robots manual and automatic control has performed. With the help of scales placed, desired values entered manually, however, on the Automatic control, any control algorithm have not applied but threshold control has done.

As a result, the intended project testbed design and analysis, control objectives were successfully completed. The project was the forerunner for the future work and completely new designs. More sophisticated control algorithms and designs can be evaluated.

6.3 Future Works

Even so Quadrotor robots history is not that old, its ongoing development process with UAV's are promising. A widespread usage is estimated to be in the future. Nowadays, more and more commercial products are available for different purposes from toys entertaining to defense. A popular application is for the movie industry is stable camera video capture from air without need expensive and time consuming installed ground apparatus. Another popular application is monitoring or spying an area generally with hexacopters due to their good stable abilities from high above in the sky for defense purposes.

Some other goals can be subjected to academical researches such as:

Trajectory path planning: Precision positioning is important to carry some parts from one location to another. Also, a planned camera caption is possible with certain specific locations. Orientation sensors, GPS sensors, barometric altitude sensors, GSM (global system for mobile communications) positioning systems have to be combined with the quadrotors for trajectory path planning.

Team objectives: Multiple air robots with different type of apparatus can be operated together to accomplish a hard task on the air. Approaching from air to operation area must be quite smart concept in some applications that ground robots cannot be operated.

Completely new mechanical frame and electronic board: More lightweight frame without compromising with flexible and durable structure can be research areas such as material sciences. Electronic boards can be built with more powerful CPUs and more reliable and extended range wireless communication including remote control and video transmission. Also, the board weight can be reduced to save battery time and increase the payload capacity.

Powerful control software: Sensory data collection and their precise signal processing are very important issues for well balanced and safe flying action. Here, some control algorithms can be developed and applied to present control software. Usually PID (Proportional Integral and Derivative) control is used for motor drive,

and some filters for sensor data smoothing for flawless operation. Also, some neural network, fuzzy systems or genetic algorithms can be applied especially for long term system stability or navigational purposes. Addition to this some statistical algorithms can be developed for the same trajectory planning tasks.

REFERENCES

- Altug, E., Ostrowski, J.P, Mahony, R. (2002). *Control of a Quadroter Helicopter Using Visual Feedback*. Proceedings of the IEEE International Conference on Robotics and Automation, Washington, D.C. 72–77.
- ArduPilot Mega UAV Controller*. (n.d.). Retrieved November 20, 2012, from <https://www.sparkfun.com/products/10294>
- Austin, R. (2010). *Unmanned Aircraft Systems. Uavs Design, Development and Deployment*. Wiley Publication. 6-7
- Austin, R. (2010). *Unmanned Aircraft Systems. Uavs Design, Development and Deployment*. Wiley Publication. 170
- Barsk, K.J. (2012). *Model Predictive Control of a Tricopter*. Institutionen för Systemteknik, Linköpings University.77
- Bhargava, A. (2008). *Development of a Quadroter Testbed for Control and Sensor Development*. 50-52
- Castillo, P., Lozano, R., & Dzul, A.E. (2005). *Modeling and Control of Mini-Flying Machines, Advances in Industrial Control*, 187
- Castillo, P., Lozano, R., & Dzul, A.E. (2005). *Modeling and Control of Mini-Flying Machines, Advances in Industrial Control*, 219
- DiCesare, A., Gustafson, K., & Lindenfelzer, P. (2009). *Design Optimization of a Quad-Rotor Capable of Autonomous Flight*. 31-36

Dikmen, İ.C., Arısoy, A., & Temeltaş, H. (2010). Dikey İniş-Kalkış Yapabilen Dört Rotorlu Hava Aracının (Quadrotor) Uçuş Kontrolü. *Havacılık ve Uzay Teknolojileri Dergisi*. 4 (3), 33-40

Draganfly. (n.d.). Innovative UAV Aircraft & Aerial Video Systems. Retrieved December 1, 2012, from <http://www.draganfly.com/industrial/products.php>

Dubois, E., Gray, P., & Nigay, L. (2010). *The Engineering of Mixed Reality Systems*. Springer-Verlag London. 221

Dupuis, M., Gibbons, J., Hobson-Dupont, M., Knight, A., Lepilov, A., Monfreda, M., & Mungai, G. (2008). *Design Optimization of a Quad-rotor Capable of Autonomous Flight*. Aerospace Engineering, Worcester Polytechnic Institute. 29-30

Electronic Speed Controller (ESC). (n.d.). Retrieved December 2, 2012, from <http://www.skyhighhobby.com/rc/all-about-speed-controls-esc>

Erginer, B. (2007). *Quadrotor Vtol Aracının Modellenmesi ve Kontrolü*. 12-13

GPS Receiver UP-501. (n.d.). Retrived December 2, 2012, from <https://www.sparkfun.com/products/10702>

Hexacopter AH680-R. (n.d.). Retrieved December 2, 2012, from <http://www.rckopter.com/index.php/hexacopter-ah680-r-1.html>.

Hexacopter, Aibotix Inc. (n.d.). Retrieved November 10, 2012, from <http://www.aibotix.com/industry.html>

Kıvrak, A.Ö. (2006). *Design of Control Systems for a Quadrotor Flight Vehicle Equipped With Inertial Sensors*. Mechatronics Engineering, Atılım University. 14-15

Kıvrak, A.Ö. (2006). *Design of Control Systems for a Quadrotor Flight Vehicle Equipped With Inertial Sensors*. Mechatronics Engineering, Atılım University. 10-11

Micro Altimeter Pressure Sensor MS5607. (n.d.). Retrieved November 4, 2012, from <http://www.jameco.com/Jameco/Products/ProdDS/2150230.pdf>

Negenborn, R. (2003). *Robot Localization and Kalman Filters On Finding Your Position in a Noisy World*. Denmark. 33-37

Octocopter. (n.d.). Retrieved December 10, 2012, from http://gallery.mikrokoetter.de/main.php/v/Nachbau/Okto1_003.JPG.html

Parallax Inc. PING #28015. (2009). Ultrasonic Distance Sensor Datasheet.

Patel, C. A. (2002). *Building a Testbed for Mini Quadrotor Unmanned Aerial Vehicle With Protective Shroud*. Sardar Patel University. 31-33

Pedersen, O.T., Karlsson, N. (1999). Time-of-Flight Ultrasonic Displacement Sensors. *The Measurement Instrumentation and Sensors Handbook*, CRC Press, 92 – 96

Pounds, P., Mahony, R., Corke, P. (2006). *Modeling and Control of a Quad-rotor Robot*. Australian National University, Canberra, Australia.

Raza, S.A. (2010). *Design and Control of a Quadrotor Unmanned Aerial Vehicle*. Electrical and Computer Engineering, School of Information Technology and Engineering, University of Ottawa, 6.

Razor IMU, *SEN-09623.9 Degrees of Freedom IMU*. (n.d.). Retrieved August 20, 2012, from <https://www.sparkfun.com/products/9623>

Smith, S.W. (2003). *Digital Signal Processing: A Practical Guide for Engineers and Scientists*. Newnes Press. 277-282

Soumelidis, A., Gaspar, P., Bauer, P., Lantos, B., & Prohaszka, Z. (2007). *Design of an Embedded Microcomputer Based Mini Quadrotor UAV*.

Starmac. (n.d.). Retrieved November 10, 2012 from <http://www.flyingrobots.org/mavws/slides/08-Hoffmann.pdf>

Walkera Ufo #5. User Handbook. (n.d.). Retrieved May 2, 2010, from <http://www.walkera.com>

Xbee RF Modules Datasheet. (2011). IEEE 802.15.4 RF Modules by Digi International.

ZeroUAV YS-X6. (n.d.). Retrieved December 3, 2012, from http://www.hobbyking.com/hobbyking/store/__25925__ZeroUAV_YS_X6_Auto_pilot_GPS_Flight_Control_System.html

APPENDICES

Razor IMU - Output File

```

void printdata(void)
{
  Serial.print("!");
  #if PRINT_EU_AN == 1
  Serial.print("ANA:");
  Serial.print(ToDeg(roll));
  Serial.print(",");
  Serial.print(ToDeg(pitch));
  Serial.print(",");
  Serial.print(ToDeg(yaw));
  Serial.print(",");
  Serial.print(ACC[0]);
  Serial.print(",");
  Serial.print(ACC[1]);
  Serial.print(",");
  Serial.print(ACC[2]);
  #endif
  #if PRINT_EULER == 1
  Serial.print("ANG:");
  Serial.print(ToDeg(roll));
  Serial.print(",");
  Serial.print(ToDeg(pitch));
  Serial.print(",");
  Serial.print(ToDeg(yaw));
  #endif
  #if PRINT_ANALOGS==1
  Serial.print(",AN:");
  Serial.print(AN[sensors[0]]); //(int)read_adc(0)
  Serial.print(",");
  Serial.print(AN[sensors[1]]);
  Serial.print(",");
  Serial.print(AN[sensors[2]]);
  Serial.print(",");
  Serial.print(ACC[0]);
  Serial.print(",");
  Serial.print(ACC[1]);
  Serial.print(",");

```

```

Serial.print(ACC[2]);
Serial.print(",");
Serial.print(magnetom_x);
    Serial.print (" ");
Serial.print(magnetom_y);
    Serial.print (" ");
Serial.print(magnetom_z);
    #endif
    /*#if PRINT_DCM == 1
    Serial.print ("DCM:");
Serial.print(convert_to_dec(DCM_Matrix[0][0]));
    Serial.print (" ");
Serial.print(convert_to_dec(DCM_Matrix[0][1]));
    Serial.print (" ");
Serial.print(convert_to_dec(DCM_Matrix[0][2]));
    Serial.print (" ");
Serial.print(convert_to_dec(DCM_Matrix[1][0]));
    Serial.print (" ");
Serial.print(convert_to_dec(DCM_Matrix[1][1]));
    Serial.print (" ");
Serial.print(convert_to_dec(DCM_Matrix[1][2]));
    Serial.print (" ");
Serial.print(convert_to_dec(DCM_Matrix[2][0]));
    Serial.print (" ");
Serial.print(convert_to_dec(DCM_Matrix[2][1]));
    Serial.print (" ");
Serial.print(convert_to_dec(DCM_Matrix[2][2]));
    #endif*/
Serial.println();
}
long convert_to_dec(float x)
{
return x*10000000;
}

```


Razor IMU – AHRS Code

```
// Sparkfun 9DOF Razor IMU AHRS
// 9 Degree of Measurement Attitude and Heading Reference System
// Axis definition:
// X axis pointing forward (to the FTDI connector)
// Y axis pointing to the right
// and Z axis pointing down.
// Positive pitch : nose up
// Positive roll : right wing down
// Positive yaw : clockwise
/* Hardware version - v13
    ATmega328@3.3V w/ external 8MHz resonator
    High Fuse DA
    Low Fuse FF
    ADXL345: Accelerometer
    HMC5843: Magnetometer
    LY530:      Yaw Gyro
    LPR530:     Pitch and Roll Gyro
Programmer : 3.3v FTDI
    Arduino IDE : Select board "Arduino Duemilanove w/ATmega328"
*/
#include <Wire.h>
// ADXL345 Sensitivity(from datasheet) => 4mg/LSB 1G => 1000mg/4mg = 256
steps
// Tested value : 248
#define GRAVITY 248 //this equivalent to 1G in the raw data coming from the
accelerometer
#define Accel_Scale(x) x*(GRAVITY/9.81)//Scaling the raw data of the accel to
actual acceleration in meters for seconds square

#define ToRad(x) (x*0.01745329252) // *pi/180
#define ToDeg(x) (x*57.2957795131) // *180/pi

// LPR530 & LY530 Sensitivity (from datasheet) => (3.3mv at 3v)at 3.3v: 3mV/°/s,
3.22mV/ADC step => 0.93
// Tested values : 0.92
#define Gyro_Gain_X 0.92 //X axis Gyro gain
#define Gyro_Gain_Y 0.92 //Y axis Gyro gain
#define Gyro_Gain_Z 0.92 //Z axis Gyro gain
#define Gyro_Scaled_X(x) x*ToRad(Gyro_Gain_X) //Return the scaled ADC raw
data of the gyro in radians for second
```

```

#define Gyro_Scaled_Y(x) x*ToRad(Gyro_Gain_Y) //Return the scaled ADC raw
data of the gyro in radians for second
#define Gyro_Scaled_Z(x) x*ToRad(Gyro_Gain_Z) //Return the scaled ADC raw
data of the gyro in radians for second

#define Kp_ROLLPITCH 0.02
#define Ki_ROLLPITCH 0.00002
#define Kp_YAW 1.2
#define Ki_YAW 0.00002

/*For debugging purposes*/
//OUTPUTMODE=1 will print the corrected data,
//OUTPUTMODE=0 will print uncorrected data of the gyros (with drift)
#define OUTPUTMODE 1

//define PRINT_DCM 0 //Will print the whole direction cosine matrix
#define PRINT_ANALOGS 0 //Will print the analog raw data
#define PRINT_EULER 0 //Will print the Euler angles Roll, Pitch and Yaw
#define PRINT_EU_AN 1 // Euler acilari ve Acc

#define ADC_WARM_CYCLES 50
#define STATUS_LED 13

int8_t sensors[3] = {1,2,0}; // Map the ADC channels gyro_x, gyro_y, gyro_z
int SENSOR_SIGN[9] = {-1,1,-1,1,1,1,-1,-1,-1}; //Correct directions x,y,z - gyros,
accelerometers, magnetometer

float G_Dt=0.02; // Integration time (DCM algorithm) We will run the integration
loop at 50Hz if possible

long timer=0; //general purpose timer
long timer_old;
long timer24=0; //Second timer used to print values
int AN[6]; //array that store the 3 ADC filtered data (gyros)
int AN_OFFSET[6]={0,0,0,0,0,0}; //Array that stores the Offset of the sensors
int ACC[3]; //array that store the accelerometers data

int accel_x;
int accel_y;
int accel_z;
int magnetom_x;
int magnetom_y;

```

```

int magnetom_z;
float MAG_Heading;

float Accel_Vector[3]= {0,0,0}; //Store the acceleration in a vector
float Gyro_Vector[3]= {0,0,0}; //Store the gyros turn rate in a vector
float Omega_Vector[3]= {0,0,0}; //Corrected Gyro_Vector data
float Omega_P[3]= {0,0,0}; //Omega Proportional correction
float Omega_I[3]= {0,0,0}; //Omega Integrator
float Omega[3]= {0,0,0};

// Euler angles
float roll;
float pitch;
float yaw;
float errorRollPitch[3]= {0,0,0};
float errorYaw[3]= {0,0,0};
unsigned int counter=0;
byte gyro_sat=0;
float DCM_Matrix[3][3]= { {1,0,0} , {0,1,0} , {0,0,1} };
float Update_Matrix[3][3]={ {0,1,2},{3,4,5},{6,7,8} }; //Gyros here
float Temporary_Matrix[3][3]={ {0,0,0} , {0,0,0} , {0,0,0} };
//ADC variables
volatile uint8_t MuxSel=0;
volatile uint8_t analog_reference;
volatile uint16_t analog_buffer[8];
volatile uint8_t analog_count[8];
void setup()
{
  Serial.begin(57600);
  pinMode (STATUS_LED,OUTPUT); // Status LED
  Analog_Reference(DEFAULT);
  Analog_Init();
  I2C_Init();
  Accel_Init();
  Read_Accel();

  Serial.println("Sparkfun 9DOF Razor AHRS");

  digitalWrite(STATUS_LED,LOW);
  delay(1500);

  // Magnetometer initialization

```

```

Compass_Init();

// Initialize ADC readings and buffers
Read_adc_raw();
delay(20);
for(int i=0;i<32;i++) // We take some readings...
{
    Read_adc_raw();
    Read_Accel();
    for(int y=0; y<6; y++) // Cumulate values
        AN_OFFSET[y] += AN[y];
    delay(20);
}
for(int y=0; y<6; y++)
    AN_OFFSET[y] = AN_OFFSET[y]/32;

AN_OFFSET[5]=GRAVITY*SENSOR_SIGN[5];
//Serial.println("Offset:");
for(int y=0; y<6; y++)
    Serial.println(AN_OFFSET[y]);
delay(2000);
digitalWrite(STATUS_LED,HIGH);
    Read_adc_raw(); // ADC initialization
timer=millis();
delay(20);
counter=0;
}
void loop() //Main Loop
{
    if((millis()-timer)>=20) // Main loop runs at 50Hz
    {
        counter++;
        timer_old = timer;
        timer=millis();
        if (timer>timer_old)
            G_Dt = (timer-timer_old)/1000.0; // Real time of loop run. We use this on the
            DCM algorithm (gyro integration time)
        else
            G_Dt = 0;

        // *** DCM algorithm
        // Data adquisition

```

```

    Read_adc_raw(); // This read gyro data
    Read_Accel();   // Read I2C accelerometer
    if (counter > 5) // Read compass data at 10Hz... (5 loop runs)
    {
        counter=0;
        Read_Compass(); // Read I2C magnetometer
        Compass_Heading(); // Calculate magnetic heading
    }

    Matrix_update();
    Normalize();
    Drift_correction();
    Euler_angles();
    // ***
    printdata();
    //Turn off the LED when you saturate any of the gyros.

    if((abs(Gyro_Vector[0])>=ToRad(300))||(abs(Gyro_Vector[1])>=ToRad(300))||(abs(
    Gyro_Vector[2])>=ToRad(300)))
    {
        if (gyro_sat<50)
            gyro_sat+=10;
    }
    else
    {
        if (gyro_sat>0)
            gyro_sat--;
    }

    if (gyro_sat>0)
        digitalWrite(STATUS_LED,LOW);
    else
        digitalWrite(STATUS_LED,HIGH);
    }
}

```

MATLAB Codes – kaydet.m file

```

tarih = datestr(now);
tarih1 = datestr(now, 'dd_mm_yy_HH_MM_SS')
cikis=0;
time=0;isik=0;temp1=0;temp2=0;temp3=0;
str="";
%sen=0;
j=1;
x=0;
g_uz=20; % grafik x eksen uzunlugu

fname=['kayit_' tarih1 '.txt'];
fid = fopen(fname, 'wt');
fprintf('%s\n',tarih); fprintf(fid,'%s\n',tarih);
fig1=figure(1);

uicontrol('Style', 'pushbutton', 'String', 'Quit',...
    'Position', [10 10 50 20],...
    'Callback', 'cikis=cikis+1;'); % Pushbutton string callback
                                % that calls a MATLAB function

s1=serial('COM5','Baudrate',57600);
fopen(s1);

str=fscanf(s1); %!ANA:0.08,0.08,-156.14
uzun=length(str);
nerde=strfind(str,'!ANA:');
yeni=str(:,(nerde+5):uzun);
    %disp(str);
sen=str2num(yeni);
    ang_x=sen(1);
    ang_y=sen(2);
    ang_z=sen(3);
fprintf(fid, '%4.2f %4.2f %4.2f\n', ang_x,ang_y,ang_z);
disp([num2str(ang_x,'%4.2f') ' ' num2str(ang_y,'%4.2f') ' ' num2str(ang_z,'%4.2f')]);

while(j<1000000 & cikis~=3)
str=fscanf(s1); %!ANA:0.08,0.08,-156.14
uzun=length(str);
nerde=strfind(str,'!ANA:');
yeni=str(:,nerde+5:uzun);

```

```

        %disp(str);
sen=str2num(yeni);
    ang_x=sen(1);
    ang_y=sen(2);
    ang_z=sen(3);
fprintf(fid, '%4.2f %4.2f %4.2f\n', ang_x,ang_y,ang_z);
disp([num2str(ang_x,'%4.2f') ' ' num2str(ang_y,'%4.2f') ' ' num2str(ang_z,'%4.2f')]);

if(0)
x(j)=j;

% if (mod((j/2),1)==0)
if(j>g_uz)
x1=x(j-g_uz:j);
size(ang_x)
    temp_ang_x=ang_x((j-g_uz):j);
    temp_ang_y=ang_y((j-g_uz):j);
    temp_ang_z=ang_z((j-g_uz):j);
xmin=j-g_uz;
xmax=j;
else
x1=x;
    temp_ang_x=ang_x;
    temp_ang_y=ang_y;
    temp_ang_z=ang_z;

xmin=0;
xmax=g_uz;
end

plot(x1,temp_ang_x,x1,temp_ang_y,x1,temp_ang_z);
xlim([xmin xmax]);
    %ylim([0 200]);
    %xlabel([num2str(floor(time(j)/1000))])
    % zaman=time(j)/1000;
    % xlabel([num2str(floor(zaman/60)) ':' num2str(floor(mod(zaman,60)))])
drawnow;
    % end
j=j+1;
end
end;
fclose(fid);

```

```
fclose(s1);  
delete(s1);  
clear s1;
```

sifir.m file

```
fclose(fid);  
fclose(s1);  
delete(s1);  
clear s1;
```


PC-Arduino File.

```

int throt = 11; // connected to digital pin 9
int ailen = 10;
int elevt= 9;
int ruddr= 6;
char arr[30]; //= { "!hiz:222,111,444,333"};
void setup()
{
  pinMode(throt, OUTPUT); // sets the pin as output
  pinMode(ailen, OUTPUT);
  pinMode(elevt, OUTPUT);
  pinMode(ruddr, OUTPUT);
  Serial.begin(115200);delay(5);
}
void loop()
{
  delay(70);

  if(Serial.available()>0) {

    char str[30];
    int x,y,z,t;
    int errors = 0;
    // char arr[26]; // Allocate some space for the string
    char inChar; // Where to store the character read
    byte index = 0; // Index into array; where to store the character

    while(index < 20) // One less than the size of the array
    {
      inChar = Serial.read(); // Read a character
      arr[index] = inChar; // Store it
      index++; // Increment where to write next
      /// Serial.println(arr);
      arr[index] = '\0'; // Null terminate the string
    }
    strcpy(str, arr);
    char *chpt = strtok(str, ":");
    if (chpt == NULL) {
      Serial.println("First strok returns NULL");
      ++errors;
    }
  }
}

```

```

if (errors == 0) {
    chpt = strtok(NULL, ",");
    if (chpt == NULL) {
        Serial.println("-1 strok returns NULL");
        ++errors;
    }
    else {
        x = atof(chpt);
    }
}

if (errors == 0) {
    chpt = strtok(NULL, ",");
    if (chpt == NULL) {
        Serial.println("Second strok returns NULL");
        ++errors;
    }
    else {
        y = atof(chpt);
    }
}

if (errors == 0) {
    chpt = strtok(NULL, ",");
    if (chpt == NULL) {
        Serial.println("Third strok returns NULL");
        ++errors;
    }
    else {
        z = atof(chpt);
    }
}

if (errors == 0) {
    chpt = strtok(NULL, ",\r\n");
    if (chpt == NULL) {
        Serial.println("Fourth strok returns NULL");
        ++errors;
        // This is an input error: do something to handle it.
    }
    t = atof(chpt);
}

```

```
}  
if (errors == 0) {  
  Serial.print("(");  
  Serial.print(x);  
  Serial.print(", ");  
  Serial.print(y);  
  Serial.print(", ");  
  Serial.print(z);  
  Serial.print(", ");  
  Serial.print(t);  
  Serial.println(")");  
  Serial.println("\n");  
  analogWrite(throt, x);  
  analogWrite(ailen, y);  
  analogWrite(elevt, z);  
  analogWrite(ruddr, t);  
}  
}  
}
```

Python Simulation Codes – 1

```

from visual import *

from visual.graph import *

import serial

import string

import math

import pylab

import numpy

from Tkinter import *

import tkMessageBox

import Tkinter

from time import time

from scipy import signal

    accdisp = gdisplay(x=500, y=0, width=700, height=200,title='ACC X - ACC Y -
ACC      Z',      xtitle='t',      ytitle='g',xmax=2000,xmin=-100,      ymax=512,
ymin=0,foreground=color.white, background=color.black)

    accx = gcurve(color=color.red)

    accy = gcurve(color=color.green)

    accz = gcurve(color=color.blue)

    accfilter = gdisplay(x=500, y=200, width=700, height=200,title='Moving Average
Filter  ACC  X-Y-Z',  xtitle='t',  ytitle='g',xmax=2000,xmin=-100,  ymax=512,
ymin=0,foreground=color.white, background=color.black)

    filterx=gcurve(color=color.red)

    filtery=gcurve(color=color.green)

    filterz=gcurve(color=color.blue)

    funct3 = gdots(color=color.yellow)

    grad2rad = 3.141592/180.0

```

```

# Check COM port and baud rate

ser = serial.Serial(port='COM5',baudrate=57600, timeout=1)

# Main scene

scene=display(title="Quadrotor Position")

scene.range=(1.2,1.2,1.2)

#scene.forward = (0,-1,-0.25)

scene.forward = (1,0,-0.25)

scene.up=(0,0,1)

# Second scene (Roll, Pitch, Yaw)

scene2 = display(title=' IMU ',x=0, y=0, width=500, height=200,center=(0,0,0),
background=(0,0,0))

scene2.range=(1,1,1)

scene.width=500

scene.y=200

scene2.select()

#Roll, Pitch, Yaw

cil_roll = cylinder(pos=(-0.4,0,0),axis=(0.2,0,0),radius=0.01,color=color.red)

cil_roll2 = cylinder(pos=(-0.4,0,0),axis=(-0.2,0,0),radius=0.01,color=color.red)

cil_pitch = cylinder(pos=(0.1,0,0),axis=(0.2,0,0),radius=0.01,color=color.green)

cil_pitch2 = cylinder(pos=(0.1,0,0),axis=(-0.2,0,0),radius=0.01,color=color.green)

#cil_course = cylinder(pos=(0.6,0,0),axis=(0.2,0,0),radius=0.01,color=color.blue)

#cil_course2=cylinder(pos=(0.6,0,0),axis=(-0.2,0,0),radius=0.01,color=color.blue)

arrow_course=arrow(pos=(0.6,0,0),color=color.cyan,axis=(-0.2,0,0),
shaftwidth=0.02, fixedwidth=1)

#Roll,Pitch,Yaw labels

label(pos=(-0.4,0.3,0),text="Roll",box=0,opacity=0)

label(pos=(0.1,0.3,0),text="Pitch",box=0,opacity=0)

```

```

label(pos=(0.55,0.3,0),text="Yaw",box=0,opacity=0)
label(pos=(0.6,0.22,0),text="N",box=0,opacity=0,color=color.yellow)
label(pos=(0.6,-0.22,0),text="S",box=0,opacity=0,color=color.yellow)
label(pos=(0.38,0,0),text="W",box=0,opacity=0,color=color.yellow)
label(pos=(0.82,0,0),text="E",box=0,opacity=0,color=color.yellow)
label(pos=(0.75,0.15,0),height=7,text="NE",box=0,color=color.yellow)
label(pos=(0.45,0.15,0),height=7,text="NW",box=0,color=color.yellow)
label(pos=(0.75,-0.15,0),height=7,text="SE",box=0,color=color.yellow)
label(pos=(0.45,-0.15,0),height=7,text="SW",box=0,color=color.yellow)

L1 = label(pos=(-0.4,0.22,0),text="-",box=0,opacity=0)
L2 = label(pos=(0.1,0.22,0),text="-",box=0,opacity=0)
L3 = label(pos=(0.7,0.3,0),text="-",box=0,opacity=0)

# Main scene objects
scene.select()

# Reference axis (x,y,z)
arrow(color=color.green,axis=(1,0,0), shaftwidth=0.02, fixedwidth=1)
arrow(color=color.green,axis=(0,-1,0), shaftwidth=0.02 , fixedwidth=1)
arrow(color=color.green,axis=(0,0,-1), shaftwidth=0.02, fixedwidth=1)

# labels
label(pos=(0,0,0.8),text="Quadrotor Position",box=0,opacity=0)
label(pos=(1,0,0),text="X",box=0,opacity=0)
label(pos=(0,-1,0),text="Y",box=0,opacity=0)
label(pos=(0,0,-1),text="Z",box=0,opacity=0)

# IMU object
platform = box(length=1, height=0.05, width=1, color=color.red)

```

```

p_line = box(length=1,height=0.08,width=0.1,color=color.yellow)

plat_arrow=arrow(color=color.green,axis=(1,0,0), shaftwidth=0.06, fixedwidth=1)

f = open("Serial"+str(time())+".txt", 'w')

roll=0

pitch=0

yaw=0

t=0

a=0

b=0

u=0

while 1:

    t=t+1

    line = ser.readline()

    if "!ANA:" in line:

        line = line.replace("!ANA:", "") # Delete "!ANA:"

        line = line.replace("\r\n", "") # Delete "\r\n"

        f.write(line) # Write to the output log file

        words = string.split(line, ",") # Fields split

        print words

        #print len(words)

        if len(words) > 5:

            try:

                roll = float(words[0])*grad2rad

                pitch = float(words[1])*grad2rad

                yaw = float(words[2])*grad2rad

            except:

```

```

print "Invalid line"

axis=(cos(pitch)*cos(yaw),-cos(pitch)*sin(yaw),sin(pitch))

up=(sin(roll)*sin(yaw)+cos(roll)*sin(pitch)*cos(yaw),sin(roll)*cos(yaw)-
cos(roll)*sin(pitch)*sin(yaw),-cos(roll)*cos(pitch))

platform.axis=axis

platform.up=up

platform.length=1.0

platform.width=0.65

plat_arrow.axis=axis

plat_arrow.up=up

plat_arrow.length=0.8

p_line.axis=axis

p_line.up=up

# alt satır - ile çarpıldı

cil_roll.axis=(-0.2*cos(roll),-0.2*sin(roll),0)

cil_roll2.axis=(-0.2*cos(roll),-0.2*sin(roll),0)

cil_pitch.axis=(0.2*cos(pitch),0.2*sin(pitch),0)

cil_pitch2.axis=(-0.2*cos(pitch),-0.2*sin(pitch),0)

arrow_course.axis=(0.2*sin(yaw),0.2*cos(yaw),0)

L1.text = str(float(words[0]))

L2.text = str(float(words[1]))

L3.text = str(float(words[2]))

if (t-(a*2000))==2000:

    t=0;accdisp.display.visible=False;a=a+1;t=t+(a*2000);

    accdisp = gdisplay(x=500, y=0, width=700, height=200,title='ACC X - ACC Y -
    ACC Z', xtitle='t', ytitle='g',xmax=2000*(a+1),xmin=a*2000, ymax=512,
    ymin=0,foreground=color.white, background=color.black);

```



```
accx = gcurve(color=color.red); accy = gcurve(color=color.green); accz =  
gcurve(color=color.blue); b=b+1;  
  
if (t-((a-1)*2000))==2000:  
  
    accfilter.display.visible=False;  
  
    accfilter = gdisplay(x=500, y=200, width=700, height=200,title='Moving Average  
Filter ACC X-Y-Z', xtitle='t', ytitle='g', xmax=2000*(b+1), xmin=b*2000, ymax=512,  
ymin=0, foreground=color.white, background=color.black);  
  
    filterx = gcurve(color=color.red); filtery = gcurve(color=color.green); filterz =  
gcurve(color=color.blue);  
  
        W=1  
  
        Y=1  
  
        Z=1  
  
temp=words[3]  
  
temp2=words[4]  
  
temp3=words[5]  
  
T=[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0]  
  
L=[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0]  
  
M=[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0]  
  
T[u]=int(temp)  
  
L[u]=int(temp2)  
  
M[u]=int(temp3)  
  
u=u+1  
  
if u==31:  
  
    u=0; xf=sigal.lfilter(ones(W)/W, 1, T ); filterx.plot( pos=(t,xf[30]) );  
  
    xf1=sigal.lfilter(ones(Y)/Y, 1, L ); filtery.plot( pos=(t,xf1[30]) );  
  
    xf2=sigal.lfilter(ones(Z)/Z, 1, M ); filterz.plot( pos=(t,xf2[30]) )  
  
accx.plot( pos=(t, words[3]) )
```

```
accy.plot( pos=(t, words[4]) )
```

```
accz.plot( pos=(t, words[5]) )
```

```
ser.close
```

```
f.close
```

Pyhton Simulation Codes – 2

```
from visual import *
from visual.graph import *
import serial
import string
import math
import pylab
import numpy
from Tkinter import *
import tkMessageBox
import Tkinter
from time import time
from easygui import *
from decimal import *
import time
from scipy import signal
import threading
var2=0
var3=0
manual_menu=0
ser = serial.Serial(port='COM8',baudrate=115200, timeout=1)
root = Tk()
var = IntVar()
var2= IntVar()
var3= IntVar()
var4= IntVar()
```

```

import threading

# Main scene

scene=display(title="Quadrotor Position",x=200, y=0, center=(0,0,0),
background=(0,0,0))

scene.range=(1.2,1.2,1.2)

#scene.forward = (0,-1,-0.25)

scene.forward = (1,0,-0.25)

scene.up=(0,0,1)

# Second scene (Roll, Pitch, Yaw)

scene2 = display(title=' IMU ',x=200, y=0, width=500, height=200,center=(0,0,0),
background=(0,0,0))

scene2.range=(1,1,1)

scene.width=500

scene.y=200

scene2.select()

#Roll, Pitch, Yaw

cil_roll = cylinder(pos=(-0.4,0,0),axis=(0.2,0,0),radius=0.01,color=color.red)

cil_roll2 = cylinder(pos=(-0.4,0,0),axis=(-0.2,0,0),radius=0.01,color=color.red)

cil_pitch = cylinder(pos=(0.1,0,0),axis=(0.2,0,0),radius=0.01,color=color.green)

cil_pitch2 = cylinder(pos=(0.1,0,0),axis=(-0.2,0,0),radius=0.01,color=color.green)

#cil_course = cylinder(pos=(0.6,0,0),axis=(0.2,0,0),radius=0.01,color=color.blue)

#cil_course2 = cylinder(pos=(0.6,0,0),axis=(-.2,0,0),radius=0.01,color=color.blue)

arrow_course = arrow(pos=(0.6,0,0),color=color.cyan,axis=(-0.2,0,0),
shaftwidth=0.02, fixedwidth=1)

#Roll,Pitch,Yaw labels

label(pos=(-0.4,0.3,0),text="Roll",box=0,opacity=0)

label(pos=(0.1,0.3,0),text="Pitch",box=0,opacity=0)

```

```

label(pos=(0.55,0.3,0),text="Yaw",box=0,opacity=0)
label(pos=(0.6,0.22,0),text="N",box=0,opacity=0,color=color.yellow)
label(pos=(0.6,-0.22,0),text="S",box=0,opacity=0,color=color.yellow)
label(pos=(0.38,0,0),text="W",box=0,opacity=0,color=color.yellow)
label(pos=(0.82,0,0),text="E",box=0,opacity=0,color=color.yellow)
label(pos=(0.75,0.15,0),height=7,text="NE",box=0,color=color.yellow)
label(pos=(0.45,0.15,0),height=7,text="NW",box=0,color=color.yellow)
label(pos=(0.75,-0.15,0),height=7,text="SE",box=0,color=color.yellow)
label(pos=(0.45,-0.15,0),height=7,text="SW",box=0,color=color.yellow)

L1 = label(pos=(-0.4,0.22,0),text="-",box=0,opacity=0)
L2 = label(pos=(0.1,0.22,0),text="-",box=0,opacity=0)
L3 = label(pos=(0.7,0.3,0),text="-",box=0,opacity=0)

# Main scene objects
scene.select()

# Reference axis (x,y,z)
arrow(color=color.green,axis=(1,0,0), shaftwidth=0.02, fixedwidth=1)
arrow(color=color.green,axis=(0,-1,0), shaftwidth=0.02 , fixedwidth=1)
arrow(color=color.green,axis=(0,0,-1), shaftwidth=0.02, fixedwidth=1)

# labels
label(pos=(0,0,0.8),text="Quadrotor Position",box=0,opacity=0)
label(pos=(1,0,0),text="X",box=0,opacity=0)
label(pos=(0,-1,0),text="Y",box=0,opacity=0)
label(pos=(0,0,-1),text="Z",box=0,opacity=0)

# IMU object
platform = box(length=1, height=0.05, width=1, color=color.red)

```

```

p_line = box(length=1,height=0.08,width=0.1,color=color.yellow)

plat_arrow=arrow(color=color.green,axis=(1,0,0), shaftwidth=0.06, fixedwidth=1)

def printit():

    threading.Timer(0.8, printit).start()

    selection=str(var3.get())+", "+str(var2.get())+", "+str(var.get())+", "+str(var4.get())

    label.config(text= selection)

    ser.write("!hiz:" + (selection))

    ser.write("!hiz:" + (selection))

    print ("!hiz:" + (selection) )

def manualst():

    global manual_menu

    printit()

    if manual_menu==0:

        manual_menu=1

        scale_1 = Scale( root, variable = var, orient=HORIZONTAL, from_=0,
to=255, resolution=1, sliderlength=24, length=180, label='Throttle',
repeatdelay=180)

        scale_1.pack(anchor=CENTER)

        scale_1.set(0)

        scale_2 = Scale( root, variable = var2, orient=HORIZONTAL, from_=0,
to=255, resolution=1, sliderlength=24, length=180, label='Ailen', repeatdelay=180)

        scale_2.pack(anchor=CENTER)

        scale_2.set(128)

        scale_3 = Scale( root, variable = var3, orient=HORIZONTAL, from_=0,
to=255, resolution=1, sliderlength=24, length=180, label='Elevator',
repeatdelay=180)

        scale_3.pack(anchor=CENTER)

        scale_3.set(128)

```

```

scale_4 = Scale( root, variable = var4, orient=HORIZONTAL, from_=0,
to=255, resolution=1, sliderlength=24, length=180, label='Ruddr', repeatdelay=180)

scale_4.pack(anchor=CENTER)

scale_4.set(128)

button= Button(root, text="Set Speed Values", command=sel )

button.pack(anchor=CENTER)

if manual_menu==1:

label.config(text='Manual Stabilization is already On')

global words

def automaticst():

    ser2 = serial.Serial(port='COM5',baudrate=57600, timeout=1)

time.sleep(2)

label.config(text='Automatic Stabilization is On')

W=1

Y=1

Z=1

T=[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0]

L=[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0]

M=[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0]

u=0

x=128;y=128;z=0;t=128;

roll=0

pitch=0

yaw=0

t=0

a=0

```

```

b=0

while 1:

z=230

    values=str(x)+',' + str(y) + ',' + str(z) + ','+ str(t)

print values;ser.write("!hiz:"+(values))

line = ser2.readline()

if "!ANA:" in line:

line = line.replace("!ANA:", "") # Delete "!ANA:"

words = string.split(line, ",") # Fields split

time.sleep(0.01)

if len(words) > 5:

try:

roll = float(words[0])*grad2rad

pitch = float(words[1])*grad2rad

yaw = float(words[2])*grad2rad

except:

print "Invalid line"

scene2.select()

axis=(cos(pitch)*cos(yaw),-cos(pitch)*sin(yaw),sin(pitch))

    up=(sin(roll)*sin(yaw)+cos(roll)*sin(pitch)*cos(yaw),sin(roll)*cos(yaw)-
cos(roll)*sin(pitch)*sin(yaw),-cos(roll)*cos(pitch))

platform.axis=axis

platform.up=up

platform.length=1.0

platform.width=0.65

    plat_arrow.axis=axis

```



```

    plat_arrow.up=up
    plat_arrow.length=0.8
    p_line.axis=axis
    p_line.up=up
    cil_roll.axis=(-0.2*cos(roll),-0.2*sin(roll),0)
    cil_roll2.axis=(-0.2*cos(roll),-0.2*sin(roll),0)
    cil_pitch.axis=(0.2*cos(pitch),0.2*sin(pitch),0)
    cil_pitch2.axis=(-0.2*cos(pitch),-0.2*sin(pitch),0)
    arrow_course.axis=(0.2*sin(yaw),0.2*cos(yaw),0)

    L1.text = str(float(words[0]))
    L2.text = str(float(words[1]))
    L3.text = str(float(words[2]))

temp=words[0]
print temp
try:
    T[u]=temp
except ValueError:
    T[u]=0
print T[u]
u=u+1
if u==18:
    u=0;
xf=T[17]
    xf2=L[17]
print xf
if xf < -2 :
```

```

        y=y+15;values=str(x)+ ',' + str(y) + ',' + str(z) + ','+ str(t);print
values;ser.write("!hiz:"+ (values));

    if xf > 5:

        y=y-15;values=str(x)+ ',' + str(y) + ',' + str(z) + ','+ str(t);print
values;ser.write("!hiz:"+ (values));

    if y>249:

        y=245;

    elif y<5:

        y=5;

    button_1= Button(root, text="Automatic Stabilization",command=automaticst )

    button_1.pack(anchor='nw')

    button_2= Button(root, text="Manual Stabilization", command=manualst)

    button_2.pack(anchor='nw')

    class App:

    def __init__(self, master):

        frame = Frame(master)

        frame.pack()

        self.button = Button(frame, text="QUIT", fg="red", command=frame.quit)

        self.button.pack(side=LEFT)

        self.hi_there = Button(frame, text="Hello", command=self.say_hi)

        self.hi_there.pack(side=LEFT)

        def say_hi(self):

            print "I opened all relays"

            ser.write("!hiz:1,1,1,1,1,1,1")

            ser.write("!hiz:1,1,1,1,1,1,1")

        def sel():

            selection=str(var.get())+" "+str(var2.get())+" "+str(var3.get())+" "+str(var4.get())

```

```
label.config(text= selection)
ser.write("!hiz:" + (selection))
ser.write("!hiz:" + (selection))
print ("!hiz:" + (selection) )
label= Label(root)
label.pack()
app = App(root)
root.mainloop()
```