CONTENTS

NOTATION

Abbreviation	Meaning
PCB	Printed Circuit Board
PWM	Pulse Width Modulation
ESC	Electronic Speed Control
BEC	Battery Eliminator Circuit
PDB	Power Distribution Board
Li-Po	Lithium-Ion
UART	Universal Asynchronous Receiver Transmitter
RPM	Revolutions Per Minute
CW	Clock Wise
CCW	Counter Clock Wise
GPS	Global Positioning System
GPGGA	Global Positioning System Fix Data
IMU	Inertial Measurement Unit
DOF	Degrees of Freedom
SCL	Serial Clock
SDA	Serial Data
CPU	Central Process Unit
RAM	Random Access Memory
SDHC	Secure Disc High Capacity
HDMI	High Definition Multimedia Interface
VNC	Virtual Network Computing
GPIO	General Purpose Input Output
NED	North-East-Down
NMEA	The National Marine Electronics Association
PID	Proportional Integral Derivative
LQ	Linear Quadratic

1. INTRODUCTION

1.1 Unmanned Aerial Vehicle (UAV)

Unmanned aerial vehicle (UAV), commonly known as a drone (Turkish ground-controlled aircraft), is a type of aircraft that can be remotely controlled. UAVs are divided into two classes: the first one is the aircraft that can be operated by remote control and the other is the aircraft that can automatically move on a specific flight plan.
Today, vehicles in many different shapes, sizes, configurations, and characters are produced. Historically, UAVs are simply "drones". However, independent control systems have been developed a lot. It can be controlled without crew and can fly at a certain altitude without stopping.[1]

In addition to military use, UAVs are used in civilian applications for our daily use, and developments in this field are increasing day by day. As an example of civil use areas of UAVs; cartography (orthophoto and digital altitude modeling, volume and area calculations, etc.), search and rescue activities, intelligence and security use, environmental observations, pollution detection, weather monitoring, fire monitoring, observation of coast and coastline, infrastructures (pipelines, airports, roads, rivers, dams, etc.) observation, agricultural practices (precise farming practices and product yield tracking), air crime scene discovery, urban transformation studies, monitoring natural disasters, archaeological studies, creating 3D city models, etc. fields can be counted.[2]

1.2 Quadcopter

A quadcopter or quadrotor is a type of helicopter with four rotors. Quadcopters generally have two rotors spinning clockwise (CW) and two counterclockwise (CCW). Flight control is provided by independent variation of the speed and hence lift and torque of each rotor. Pitch and roll are controlled by varying the net centre of thrust, with yay controlled by varying the net torque. Unlike conventional helicopters, quadrotors do not usually have cyclic pitch control, in which the angle of the blades varies dynamically as they turn around the rotor hub. In the early days of flight, quadcopters (then referred to either as 'quadrotors' or simply as 'helicopters') were seen as a possible solution to some of the persistent problems in vertical flight. Torque-induced control issues (as well as efficiency issues originating from the tail rotor, which generates no useful lift) can be eliminated by counter-

rotation, and the relatively short blades are much easier to construct. These vehicles were among the first successful heavier-than-air vertical take off and landing (VTOL) vehicles.

There are many reasons why the unmanned aerial vehicle to be designed in this project is preferred as a quadcopter. The first is the simple and easier-to-control structure of Quadcopters. By driving 4 motors at different speeds, the quadcopter can be easily guided. The Quadcopters body structure is simple and easy to find on the market, so the body can find it without difficulty in procurement. The structure of the quadcopter provides advantage of more flexible maneuver calibration due to its smaller structure compared to other unmanned aerial vehicles. Another advantage is that the quadcopter does not need a runway for take-off and landing, such as fixed-wing unmanned aerial vehicles, and they can make vertical take-off and landing. Lastly, quadcopters have been preferred in this project because of their wide range of usage, because there are many researches and projects about them, they are open to implementation and can be developed.

Figure 1.2.1

2. STATE OF THE ART

We need some flight control cards to control unmanned aerial vehicles. In this project, it is aimed to set up a control system with a raspberry pi device without using flight control cards but taking these as an example. These cards are embedded structures with a microcontroller. In the project, these embedded structures will be tested on a single board computer. Some of these cards and their features are described in this section.

2.1 ArduPilot APM

ArduPilot Mega (APM) is a highly developed autopilot system used to provide autonomous mobility in air and land vehicles. Multirotor can be used with remote control model vehicles as well as aircraft such as aircraft and helicopters. It allows the user to turn a stationary, rotary, or multi-wing vehicle (even cars and boats) into a fully autonomous vehicle. Capable of performing GPS tasks programmed with waypoints. APM is an Arduino Mega based control board. There are an ATmega2560 microcontroller and ATmega32u2 microcontroller for USB-Serial conversion. In addition to microcontrollers, sensors such as accelerometers, gyroscopes, barometers, and magnetometers required for autonomous movement are integrated on the board. ESC and servo connections required for the vehicle to be used are located on the card. Ready-made connectors for external sensor and telemetry connections are located on the board. Various user interfaces are available for programing, reviewing logs, even smartphones, and tablets. The optional MAVLink telemetry radio allows two-way radio communication that provides full control and provides live data to the computer or tablet. [3][4]

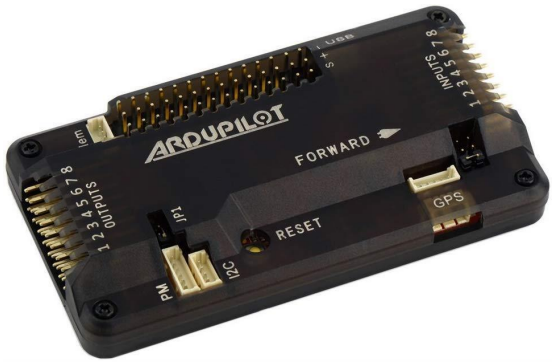


Figure 2.1.1

2.2 Emblid Navio2

Emblid Navio2 Flight control card is an open-source development board that can be used in all types of unmanned systems (drone, aircraft, vehicle, etc.). It can be mounted directly on the Raspberry Pi card. There are some internal sensors on the Emlid Navio2 board. Depending on the need, you can use these internal incoming sensor modules, or depending on the need for a different feature or module, extra modules can be connected with the connection connectors on it. Emblid Navio2 flight control board supports ArduPilot and ROS systems. Navio2 Flight Control Card works with the Linux operating system. The device has a 64-bit quad-core ARMv8 CPU running at 1.2GHz speed and 1GB of RAM. In this way, it can perform the transactions very quickly and process the data. Navio2 has two IMU chips (MPU9250 9DOF IMU and LSM9DS1 9DOF IMU) to improve flight experience and for redundancy. Transactions from other chips on the bus, which MS5611barometer sensor is connected to, can produce noise during the conversion. That's why MS5611 was left the only sensor on the I2C bus to overcome this. Main limitation of PCA9685 chip is the inability to control frequencies for separate channels. This leads to problems with motors and servos that work on different frequencies. To eliminate that problem a microcontroller is used. It allows to set frequencies for output channels by groups. PPM/SBUS decoding done by microcontroller instead of DMA. On Navio2 a microcontroller handles PPM/SBUS sampling leaving processor cores of Raspberry Pi 2 for your tasks. Navio2 is the first HAT to utilize AUX SPI controller on Raspberry Pi. Using two SPI controllers allows to distribute sensors more efficiently. PWM, ADC, SBUS and PPM are integrated in Linux sysfs allowing for easy access from any programming language. [5][6]

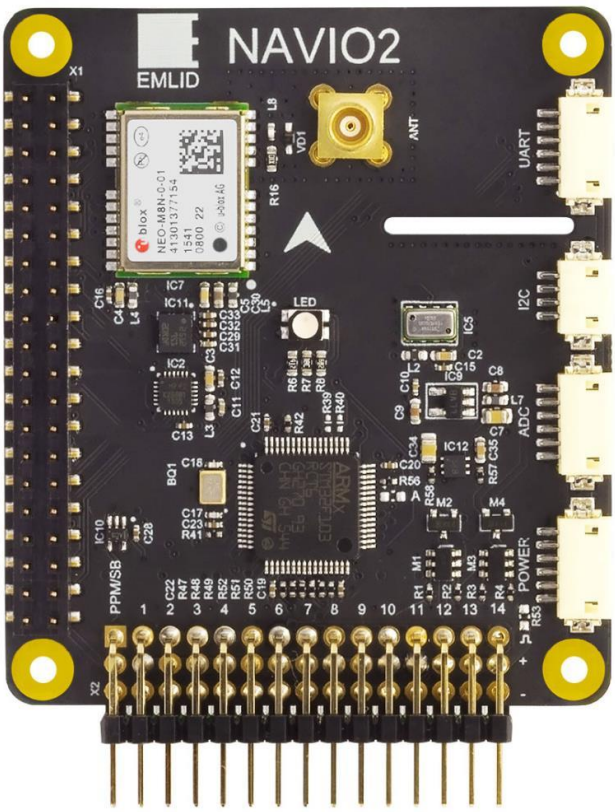


Figure 2.2.1

2.3 Pixhawk

Pixhawk is a highly developed autopilot system that can be used to give autonomous mobility to air and ground vehicles. Besides multi-rotor, aircraft such as aircraft and helicopters, it can also be used with remote control model vehicles. The Pixhawk flight controller is based on the rapidly developing and refined Ardupilot mega or 3DR robotic open source project "APM". This flight controller enables the user to turn fixed wing, rotary wing, or multi-wing vehicles (even boats and cars) into a fully autonomous vehicle, which can even perform GPS tasks programmed with waypoints with the optional GPS Module. It is a complete autopilot that can provide support for bidirectional telemetry with autonomous stabilization, waypoint-based navigation, and radio telemetry modules. Various user interfaces are available for programing, reviewing logs, and even some apps for smartphones and tablets. The optional telemetry radio allows two-way radio communication that provides full control and provides live data to the computer or tablet. The advantages of the Pixhawk system include integrated multi-use, a Unix / Linux-like programming environment, completely new autopilot functions such as complex tasks and flight behavior commands, and a special PX4 driver layer that provides tight timing between all processes. These advanced capabilities ensure that there are no vehicle limitations. Pixhawk allows existing APM and PX4 operators to switch to this system seamlessly. Pixhawk is a PPM-input autopilot. RC input is taken from a single pin. [7][8]

Figure 2.3.1

3. THEORETICAL BACKGROUND

3.1 Euler Angles

The equations of motion have been derived for an axis system fixed to the quadcopter. Unfortunately, the position and orientation of the airplane cannot be described relative to the moving body axis frame. The body-fixed coordinate system can be used to relate measurements and estimations to the inertial system. In aeronautical applications this is commonly a North-East-Down (ned) coordinate system, with the X axis pointing fore, the Y axis pointing starboard and the Z axis pointing to the keel of the quadcopter. The rotational velocity of the quadcopter is measured by the angular rates [p q r]. The quadcopter's velocities along [Xb Yb Zb]] are [u,v,w]. The three rotations needed to transform the global coordinate system [x y z] to the local system [Xb Yb Zb]. To relate the orientation of the local coordinate system relative to a global one, the Euler angles roll (φ), pitch (θ), and yaw (ψ) can be used. Each rotation results in a new coordinate system,and the two intermediate coordinate systems are denoted [x1 y1 z1] and [x2 y2 z2]. [Xb Yb Zb] is reached by rotating ψ radians around zf, θ radians around y1 and φ radians around x2. Imagine the quadcopter to be positioned so that the body axis system is parallel to the fixed frame and then apply the following rotations:

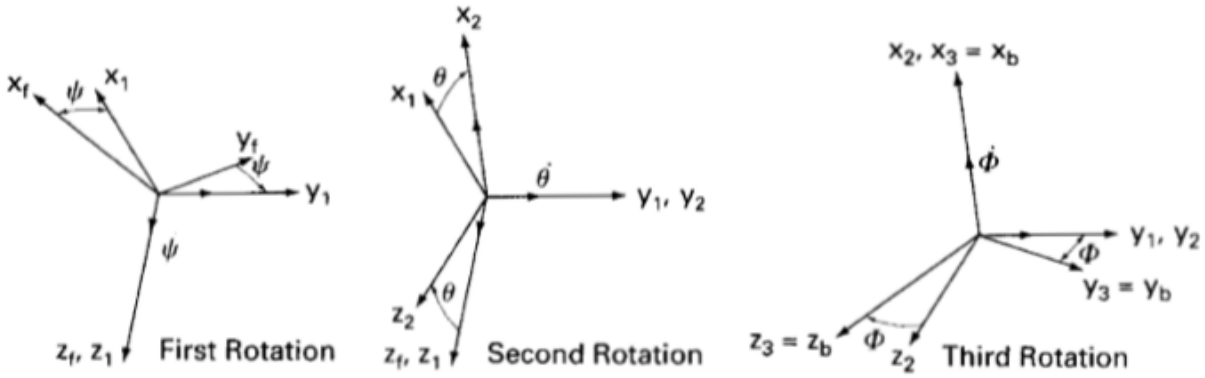


Figure 3.1.1 : shown the orientation and position of the quadcopter can be defined in terms of a fixed frame of reference.At time t = 0, the two reference frames coincide.

First rotation: Rotate the [xf yf zf] frame about zf through the **yaw angle ψ** to the frame to [x1 y1 z1].

Second rotation: Rotate the [x1 y1 z1] frame about y1, through the **pitch angle θ** bringing the frame to [x2 y2 z2].

Third rotation: Rotate the [x2 y2 z2] frame about x2, through the **roll angle φ** to bring the frame to [x3 y3 Z3] the actual orientation of the body frame relative to the fixed frame.

***([x3 y3 Z3] = [Xb Yb Zb])

Having defined the Euler angles, one can determine the flight velocities components relative to the fixed reference frame. To accomplish this, let the velocity components along the [xf yf zf] , frame be [dx/dt dy/dt dz/dt] and similarly let the subscripts first rotation and second rotation denote the components along [x1 y1 z1] and [x2 y2 z2] respectively. We can show that:

$$\frac{dx}{dt} = u_1 \cos\psi - v_1 \sin\psi \qquad \frac{dy}{dt} = u_1 \sin\psi + v_1 \cos\psi \qquad \frac{dz}{dt} = w_1 \tag{1}$$

Figure 3.1.2

let us use the shorthand notation:
Sφ = sin(φ), Cφ = cos(φ)
Sθ = sin(θ), Cθ = cos(θ)
Sψ = sin(ψ), Cψ = cos(ψ)
In manner similar to equation 1, [x1 y1 z1] can be expressed in terms of [x2 y2 z2]:

U1 = u2Cθ + w2Sθ v1 = v2 w1 = -u2Sθ + w2Cθ (2)

U2 = u v2 = vCφ - wSφ w2 = vSφ + wCφ (3)

where u, v, and w are the velocity components along the [xf yf zf] frame. Determine the absolute velocity in terms of the Euler angles and velocity components [dx /dt dy/dt dz/dt] in the body frame:

$$\begin{bmatrix} \frac{dx}{dt} \\ \frac{dy}{dt} \\ \frac{dz}{dt} \end{bmatrix} = \begin{bmatrix} C_\theta C_\psi & S_\Phi S_\theta C_\psi - C_\Phi S_\psi & C_\Phi S_\theta C_\psi + S_\Phi S_\psi \\ C_\theta S_\psi & S_\Phi S_\theta S_\psi + C_\Phi C_\psi & C_\Phi S_\theta S_\psi - S_\Phi C_\psi \\ -S_\theta & S_\theta C_\theta & C_\Phi C_\theta \end{bmatrix} \begin{bmatrix} u \\ v \\ w \end{bmatrix} \tag{4}$$

Figure 3.1.3

Integration of these equations [dx /dt dy/dt dz/dt] yields the quadcopter's position [Xb Yb Zb]) relative to the fixed frame of reference [xf yf zf].

The relationship between the angular velocities [p q r] in the body frame [xf yf zf] and the Euler rates (ψ', θ', φ') also can be determined from Figure 3.1.3:

$$\begin{bmatrix} p \\ q \\ r \end{bmatrix} = \begin{bmatrix} 1 & 0 & -S_\theta \\ 0 & C_\Phi & C_\theta S_\Phi \\ 0 & -S_\Phi & C_\theta C_\Phi \end{bmatrix} \begin{bmatrix} \dot{\Phi} \\ \dot{\theta} \\ \dot{\psi} \end{bmatrix} \quad (5)$$

Figure 3.1.4

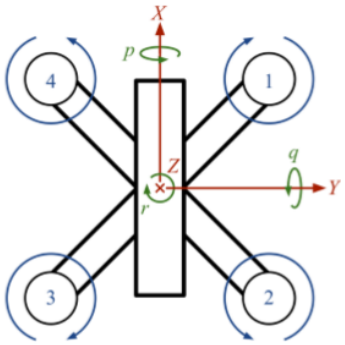


Figure 3.1.5 : The body-fixed coordinate system and the angular rates of the quadcopter

be solved for the Euler rates (ψ', θ', φ') in terms of the body angular velocities [p q r]:

$$\begin{bmatrix} \dot{\Phi} \\ \dot{\theta} \\ \dot{\psi} \end{bmatrix} = \begin{bmatrix} 1 & S_\Phi \tan\theta & C_\Phi \tan\theta \\ 0 & C_\Phi & -S_\Phi \\ 0 & S_\Phi \sec\theta & C_\Phi \sec\theta \end{bmatrix} \begin{bmatrix} p \\ q \\ r \end{bmatrix} \quad (6)$$

Figure 3.1.6

By integrating these Euler rates equations [ψ' θ' φ'], we can determine the Euler angles ψ, θ and φ. So, These angles show quadcopter's movement as shown in Figure 3.1.7 [9][10][11].

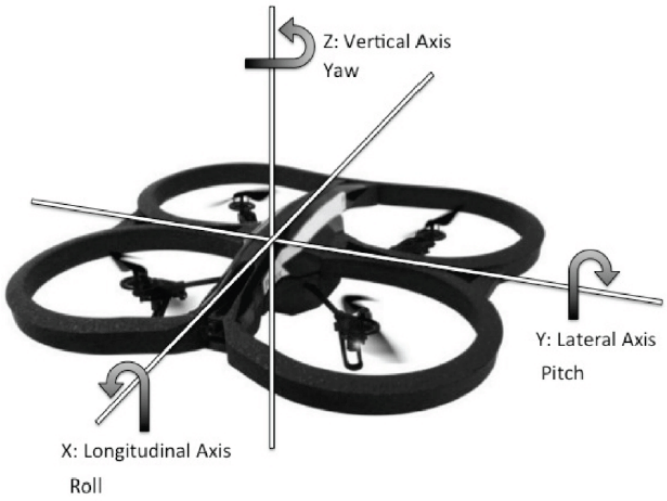


Figure 3.1.7

3.2 Quadrotor Flight Control Movements

Figure 3.2.1 shows the illustration for quadrotor flight control movement. Take-off or landing movement is controlled by increasing or decreasing all four motors in throttle control, pitch control which is in y-axis is controlled by varying speed of front and back motors. For roll movement which is in x-axis is controlled by varying speed of right and left motors. For Yaw movement which is in z-axis is controlled by varying the speed of pair propeller in opposite axis.

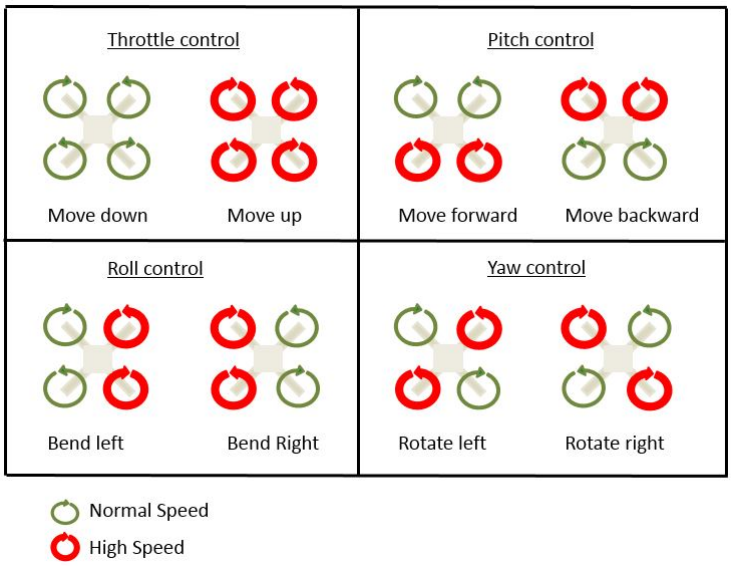


Figure 3.2.1

3.3 Kalman Filter

The noise covariances R[k] and Q[k] are also assumed to be constant. Using this model, state estimates x[k] and state covariances P[k] can be obtained. This is done in two steps, the time update that updates the state estimates according to the model, and the measurement update that updates the state estimates using observations. The innovation covariance S and the Kalman gain K as.

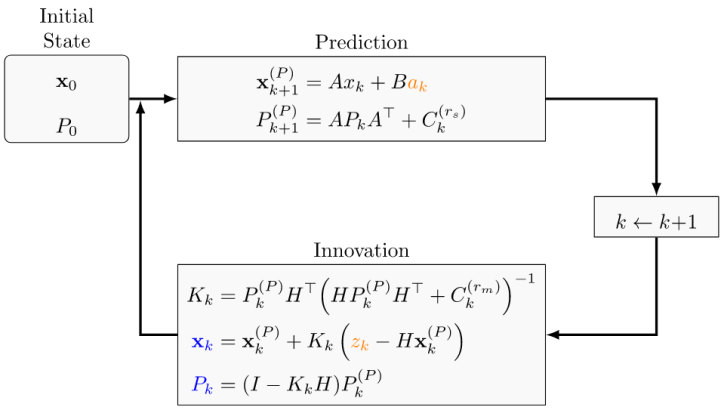


Figure 3.3.1

In this project, the Kalman filter designed for MPU6050 is used. The algorithm can be accessed with the command git clone https://github.com/rocheparadox/Kalman-Filter-Python-for-mpu6050.But this filter gives a different accuracy rate for quadcopter with a different body. This algorithm is as follows :

1- Initial state ;

```
#KALMAN FILTER
class KalmanAngle:
    def __init__(self):
        self.QAngle = 0.001
        self.QBias = 0.003
        self.RMeasure = 0.03
        self.angle = 0.0
        self.bias = 0.0
        self.rate = 0.0
        self.P=[[0.0,0.0],[0.0,0.0]]
```

Figure 3.3.2

2- Prediction ;

```
def getAngle(self,newAngle, newRate,dt):
    #step 1:
    self.rate = newRate - self.bias;    #new_rate is the latest Gyro measurement
    self.angle += dt * self.rate;

    #Step 2:
    self.P[0][0] += dt * (dt*self.P[1][1] -self.P[0][1] - self.P[1][0] + self.QAngle)
    self.P[0][1] -= dt * self.P[1][1]
    self.P[1][0] -= dt * self.P[1][1]
    self.P[1][1] += self.QBias * dt
```

Figure 3.3.3

3- İnnovation ;

```
#Step 3: Innovation
y = newAngle - self.angle
#Step 4: Innovation covariance
s = self.P[0][0] + self.RMeasure
#Step 5:    Kalman Gain
K=[0.0,0.0]
K[0] = self.P[0][0]/s
K[1] = self.P[1][0]/s
#Step 6: Update the Angle
self.angle += K[0] * y
self.bias  += K[1] * y
```

Figure 3.3.4

4- Error covariance is cov(e[k]) = R[k] and calculating this.

```
#Step 7: Calculate estimation error covariance - Update the error covariance
P00Temp = self.P[0][0]
P01Temp = self.P[0][1]
self.P[0][0] -= K[0] * P00Temp;
self.P[0][1] -= K[0] * P01Temp;
self.P[1][0] -= K[1] * P00Temp;
self.P[1][1] -= K[1] * P01Temp;
return self.angle
```

Figure 3.3.5

After these sections, gyroscope and acceleration data are obtained by using the steps in the algorithm written for MPU6050, and these data are converted to roll and pitch angles by converting them to degrees and by deriving them.

Output of Kalman Filter:

```
Angle X: -29.14239571656869   Angle Y: 0
Angle X: -28.867627535622116   Angle Y: 0
Angle X: -28.672313371374802   Angle Y: 0
Angle X: -28.403678545121455   Angle Y: 0
Angle X: -28.10549375077996   Angle Y: 0
Angle X: -27.79935982108256   Angle Y: 0
Angle X: -27.534342284219687   Angle Y: 0
Angle X: -27.337194684605432   Angle Y: 0
Angle X: -27.213717381815364   Angle Y: 0
Angle X: -27.135191053249308   Angle Y: 0
Angle X: -27.084273333028246   Angle Y: 0
```

Figure 3.3.6

As it is understood from the result, the data obtained for the x-axis is momentary and correct, but it is wrong for the y-axis.

The Kalman filter consists of equations that give the actual and common values of the x and y-axis by filtering four data from the IMU (gyroscope and accelerometer). By removing the noise in the data from the sensors, stable values are obtained. It is much more complicated than that, though it is simply this way because it depends on the system kinematics, the relationship between the airframe coordinate system and the global coordinate system, the propulsion of the vehicle. By creating these equations, filtering using the necessary algorithm can be done with professionals working in this field. The filtered data creates errors that differ from the actual values from the desired values, and the errors pass through the PID control algorithm and form a loop as follows : [12]

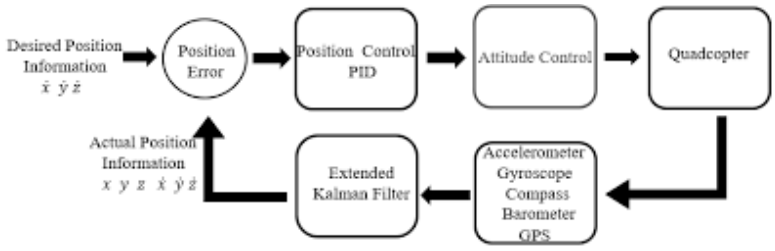


Figure 3.3.7

3.4 PID (Proportional–Integral–Derivative) Control

In order to keep the quadcopter in balance, motors should be controlled with IMU data. But since the data from the sensors cannot be used for direct control, it is filtered and added to the speed values of the ESC according to the movements of the quadcopter with the output of the PID using the PID controller. PID calculations are as follows :

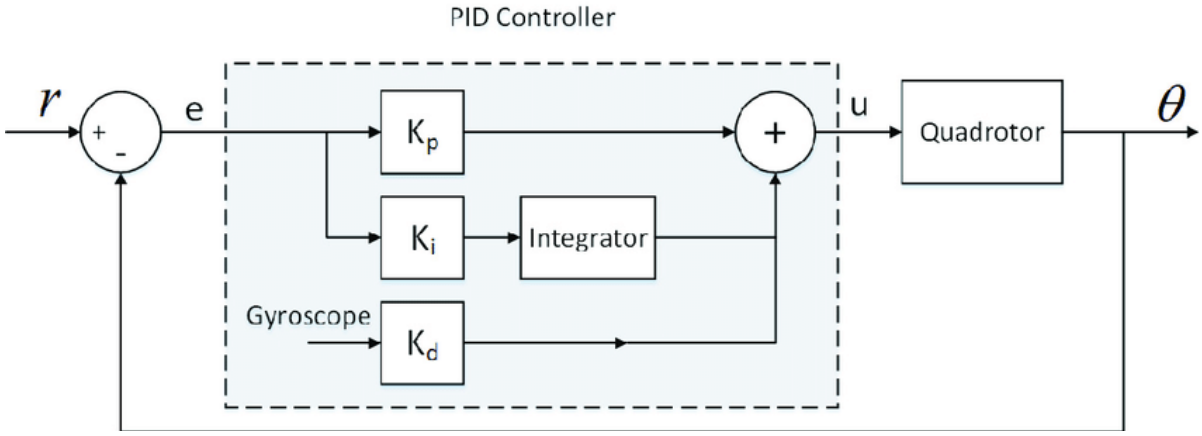


Figure 3.4.1

e = desired value – actual value

PID = Kp.e(t) + Ki$\int_0^t e(t)$ + Kd d/dt e(t)

Kp = Proportional Constant
Ki = İntegral Constant
Kd = Derivative Constant

For algorithm;
PID = Kp*error + (Ki*integral*dt) + (Kd*derr/dt)
Error = desired point – actual point
Integral = previous integral + error
Derr = error- previous error
Dt = execution time

4. DESIGN OF THE SYSTEM

4.1 HARDWARE

4.1.1 Airframe

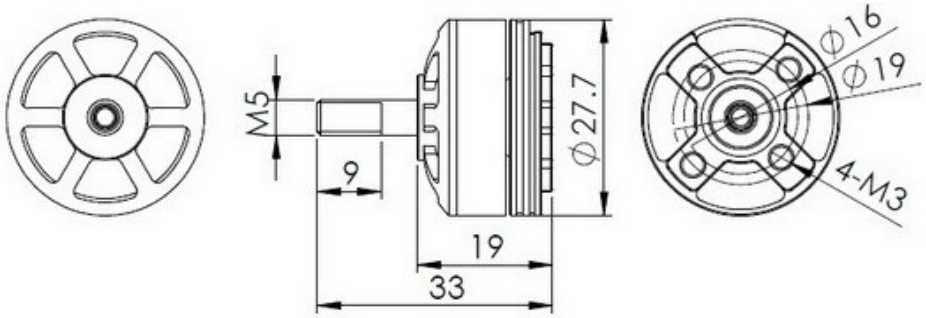
When we started the project, we had two options for drone body selection. These are using a ready drone body or making our own drone body. As the quadcopter will carry loads in this project, it was our main objective when choosing a body structure that was durable and light. The drone body had to be durable for the problem to be experienced in load carrying and flight tests. On the other hand, our engine choices had to be light to carry flight dynamics and higher loads. Since meeting these conditions in the drone body that we will build requires a high cost and time, we decided to buy a ready drone body. When we evaluate according to the cost, the **F14889 380mm** drone body was one of the leading options. The model with a glass fiber frame kit allows 380mm optional motor distance and 5”-10” propeller usage. The model weighs approximately 200 grams and can control the take-off weight as a total of 500 grams as a whole. When starting this project, this product was preferred because our total departure weight was close to these levels. Easily assembled and tested according to the body assembly guide.

Figure 4.1.1.1

4.1.2 Motors

Engines are one of the basic hardware products of our project that enables the movement of the quadcopter. For this reason, it was chosen as a result of detailed research and calculations. Brushless DC motors are preferred as motor type. Brushless motors do not have brushes and collectors. Here, the magnets are located on the shaft of the motor, and the windings are fixed. There are 3 connections in brushless motors connected to different phases of the windings inside the motor. By giving electric current to different phases respectively, a magnetic field is created in reverse direction on the magnets inside the rotor, and thus the motor rotates. Unlike brushed motors, it does not need a brush and collector to apply tension to the coils. Thus, friction loss due to friction is less. Brushless DC motors are divided into two as inrunner and outrunner according to rotor types. In addition, there is another type of hybrid structure. In Inrunner motors, the rotor is located inside the motor. In RC models, inrunner type brushless motors are preferred due to the high KV (revolutions per voltage) value. In Outrunner type motors, the windings are located inside the motor, while the rotor is outside. Outrunner type brushless motors with high torque are preferred in model aircraft, helicopters, and drones. Brushless motors cannot be controlled by applying direct voltage or using simple drive circuits. We must definitely use a driver to control brushless motors. This driver circuit is called ESC (electronic speed control) for short. Since the current and voltage values of motors of different sizes and types will vary, different ESCs are available according to different current and voltage values.

The engine selection was primarily selected by making thrust calculations according to the starting weight. DYS SE2205 engines are preferred for use in this project. Engine features are as follows.



Specifications:

KV	2550
Configu ration	NP
Stator Diameter	22.0mm
Stator Length	5.0mm
Shaft Diameter	3.0mm
Motor Dimensions(Dia.*Len)	Φ27.7×19.0mm
Weight (g)	30g
No.of Cells(Lipo)	3-5S
Max Continuous current(A)	27.8A
Max Continuous Power(W)	411.4W
Internal resistance	0.08ohm

Figure 4.1.2.1

MOTOR TECHNICAL DATA:							
	NO LOAD		ON LOAD				LOAD TYPE
VOLTAGE	CURRENT	SPEED	CURRENT	Pull	Power	EEP	Battery/prop
V	A	rpm	A	g	W	%	
11.1	0.7	28416	2.8	200	31.1	6.4	LiPox3/5045
			5.8	400	64.4	6.2	
			13.2	570	146.5	3.9	
			3.1	200	34.4	5.8	LiPox3/6045
			7.6	400	84.4	4.7	
			19.1	750	212.0	3.5	
			4.5	200	50.0	4.0	LiPox3/T5045
			11.3	400	125.4	3.2	
			15.5	600	172.1	3.5	
			3.6	200	40.0	5.0	LiPox3/T6045
			9.3	400	103.2	3.9	
			21.8	700	242.0	2.9	
14.8	0.8	37888	4.5	300	66.6	4.5	LiPox4/5045
			11.9	600	176.1	3.4	
			19.1	840	282.7	3.0	
			4.1	300	60.7	4.9	LiPox4/6045
			11.0	600	162.8	3.7	
			26.2	1020	387.8	2.6	
			4.1	300	60.7	4.9	LiPox4/T5045
			11.8	400	174.6	2.3	
			22.3	820	330.0	2.5	
			4.3	300	63.6	4.7	LiPox4/T6045
			13.2	600	195.4	3.1	
			27.8	890	411.4	2.2	

Figure 4.1.2.2

In this project, 3S (11.1V-2500mAh) Li-Po battery and 5045 probe set will be used according to the hardware features determined during the design phase. Battery and probe selection will be explained separately. According to these choices, looking at the table above, our engine will reach 28416 rpm with 11.1V at no load. When we load our engine with 5045 probes, we will be able to get 570g of thrust at a maximum value of 13.2 Amps. However, we can obtain a maximum of 200g of thrust from each engine, according to 2500mAh of our battery. If we calculate this for 4 engines (4x200), our total impulse value will be 800g. It will be correct to use these engines since this thrust value is within our maximum starting weight.

4.1.3 Propellers

Propeller is a system that converts the rotation transmitted from the motor to the thrust force. Converts rotary motion to linear motion. It consists of two or more propeller blades. We can think of each blade as an airplane wing. It has a wing profile, just like the plane wing.

They are attached to the propeller shaft from the center of the propellers. When the shaft turns, the propellers also rotate. There are two types of dimensions in our propeller: length and pitch. The length is the diameter of the disc that the propeller creates while rotating. The pitch can be defined as the travel distance of the propeller in 1 full turn. When both measures increase in the propeller, the energy required to rotate also increases. Propellers provide thrust by rotating and moving it into the air. When it turns faster, it moves more air. As a result, more pushes are generated. By increasing the pitch and length of the propeller, we can achieve more thrust, but to do this will be the high current consumption. Naturally, the surface area of the pallets will increase and the drag will also increase. More air will be pushed, but more energy will be needed to rotate the engine. Generally, the smaller propeller can spin at higher RPMs. Because the motor will not force itself to turn and will rotate at low current. It will be softer and more responsive. Lower-sized propellers are

suitable for rapid RPM changes. This will provide us more stable quadcopter movements. Increasing the number of pulses in the propeller will increase the surface area and more thrust will be provided, but the current consumption will increase and we will encounter more continuity. For this reason, 2-blade propellers have been chosen in this project. In each multi-copter, we see two different motor rotation directions: clockwise (CW) and counterclockwise (counter-clockwise CCW). Conjugated CW and CCW propellers are required here to produce thrust. Also, and most importantly, they balance each other along the pitching axis. In this project, 2-blade propellers (5045 * 2) with 5th length and 4.5th pitch were used. In order to be affordable and lightweight, the quality of plastic material was preferred. [13]

Figure 4.1.3.1

4.1.4 Lipo Battery

The term Li-Po emerged with the shortening of Lithium Polymer batteries. It is a rechargeable Lithium-Ion battery type that uses polymer electrolyte instead of liquid electrolyte. Lipo batteries are made up of cells, and each of the cells is marked with an S icon. There are 1S, 2S, 3S, 4S, 5S, 6S types. The 1S value corresponds to 3.7 V. The C value of lipo batteries refers to their capacity. For example, if we examine the 3S 11.1V 2500 mAh 25C lipo battery, the lipo battery can give 2.5 A and it shows the current capacity that the 35X2.5A = 87.5 A battery can give maximum. [14]

Figure 4.1.4.1

4.1.4.1 Lipo Battery Selection for Quadcopter

- The number of cells of the lipo battery is determined by looking at the efficiency values of the technical table of the engine we selected. For example, since the efficiency values of the motor selected for this project are higher for 11.1V, a 3S battery is selected. The ESC used at this point should also support the 3S battery.

- The lipo battery should be selected according to the desired flight time because all parts of the quadcopters are selected by weight and the weight of the batteries increases as the mAh value increases. For example, if we want a 20 A system to fly an hour, at least 20.000mAh battery is required. Since this project is a prototype, the engines have low trust. Therefore, a 190 gr 3S battery was used.

- It is determined from the table that the motor uses 2.8 A for 200 g trust. The 2500 mAh battery can give 150A value in one minute. It is calculated in 80% efficiency and our value is 120A / sec. The current (120A) the battery can deliver in one minute is divided by the total current that the four engines will use and our flight time is found as 13.3 minutes. With this information, the mAh value of the battery can be determined according to the flight time we want.

- We calculate the highest current value used by the motor we choose with a 3S battery. 13.2x4 = 52.8 A is found. The motors will probably not reach this current value, but for safety, a battery capable of delivering this value should be selected. For example, a 25C battery (25x2.5 = 62.5A) is sufficient for this project.

- Price and brand can also be selected according to the budget allocated.

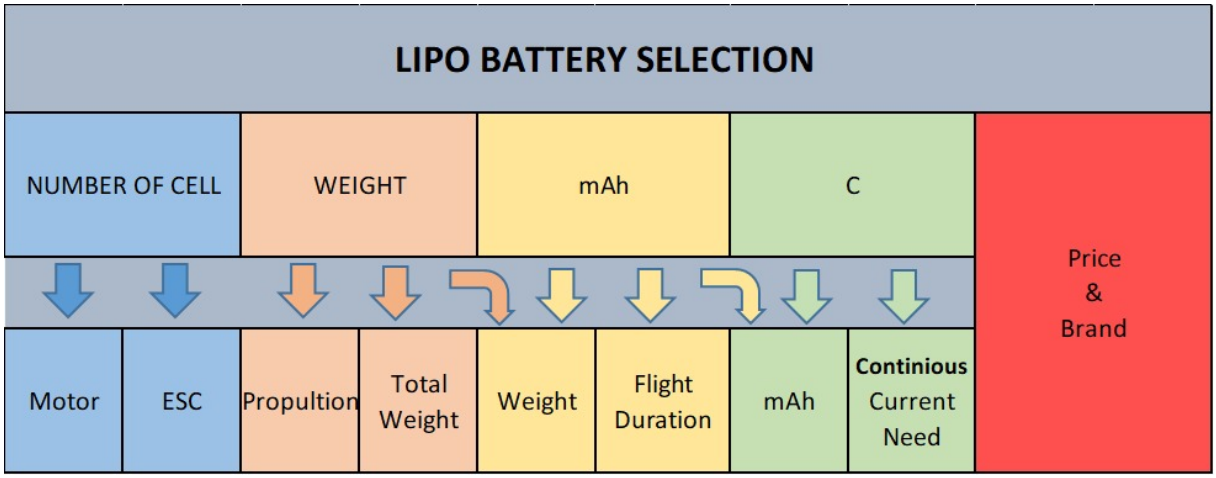


Figure 4.1.4.2

4.1.5 Electronic Speed Controller (ESC)

Electronic speed control or ESC is an electronic circuit that controls and regulates the speed of an electric motor. It can also provide motor reversing and dynamic braking. For example, we control an RC model car's forward or backward and speed with ESC. ESCs can generally be collected under two headings. The first of these is Brushed ESC: The motor

connection consists of 2 cables and used with brushed motors. The second one is Brushless ESC: The motor connection consists of 3 cables and used with brushless motors. The most important thing to consider when choosing an ESC is to buy a brushed ESC if the motor is brushed, and brushless ESC if it is brushless. Because the way of connection in the two systems is different. Brushed ESCs are connected to the motor with 2 (+ and -) cables, brushless ESCs are connected to the motor as 3 (+, - and signal) cables. Since the motors we use in our project are brushless dc motors, we used brushless ESC's. Our second priority when choosing this ESC was the system we would use. Since a quadcopter was built in this project, we had to choose a suitable ESC with a high response speed. Drone ESCs, on the other hand, are usually 400Hz and higher so that they can react faster. And finally, our criterion when choosing ESC is the maximum operating current of the engine we will use in our system. We know that the maximum draw current is 13.2 A, assuming that our motor is loaded as in the design diagram, as stated in the section where the motor features are described. The most suitable ESC for this is a 20A ESC. However, it is considered appropriate to use an ESC of 30 A by taking into account the changes that may occur during the transition from design to production and the excess current draw problem that may arise due to the instant overload of the engine. The image and technical information of the used ESC are given below.

Figure 4.1.5.1

Technicial Specifications
Output: Continuous 30A, 40A up to 10 seconds
Input Voltage: 2-3S Lipo, 5-9 cells NiMH
BEC: 2A / 5V Linear mode BEC.
Its refresh rate: 50Hz - 432Hz.
Max speed: 210000rpm for 2 Pole Brushless Motors
70000rpm Bruhsless Motors for 6 poles
35000 rpm Brushless Motors for 12 poles
Size: 50mm x 27mm x 9mm
Weight: 22g

Very high currents pass over the ESCs. And Every ESC has a capacity and endurance limit. ESC will illuminate if this limit is exceeded in current (Amperes) and volts. ESC must not be allowed to force it to avoid damage. For this reason, tolerance ranges are kept wide. Since a 3S battery will be used in the system, the input voltage is selected in this range. This battery voltage input is applied to the input of ESC from red (+) and black (-) cables. ESV

will be disturbed when the voltage is over the voltage value supported by ESC. Ground connection (GND), speed signal (PWM) connection, and BEC (5V) output can be made from another 3-pin input jumper cables. There is no need for BEC output in our system. It is normal for ESCs to get a little warm during operation. If overheating occurs during use, ESC is having trouble. It should be kept for a while to cool and should not be used. [15]

4.1.6 Power Distrubition Board (PDB)

PDB (Power Distribution Board), mounted on the drone; is the circuit board that organizes the power connections of batteries, ESCs, and other onboard systems. It is not necessary for all drones, but it helps to create a regular and organized drone. It comes self-assembled in some flight controllers. We made it appropriate to include this card in our system to make a regular and stable drone in the project. Our card is responsible for distributing the power coming from the battery to the ESCs, thanks to this card, a more successful and stable distribution process is performed. There is no problem due to the distribution of power in the system. In addition, thanks to the DC / DC synchronous regulator operating at 95% efficiency on the device, the BEC (5V) output can feed other parts of our system and our controller. Thanks to the buzzer on the card, we can alert the user at low voltage levels. We found it appropriate to use the Matek LED and POWER HUB 5in1 V2, PDB (Power Distribution Card) card in our system. This card has been selected according to the input voltage value in accordance with our system requirements and stock status. The connection diagram is as follows. []

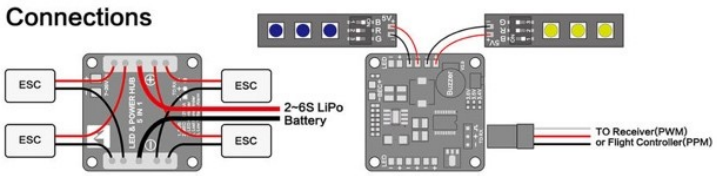


Figure 4.1.6.1

Figure 4.1.6.2

4.1.7 Global Positioning System (GPS)

Since the main purpose of this project is to develop a quadcopter capable of autonomous flight, we need to add many sensors to our device. The first one is the GPS sensor that our device will find its direction. With the GPS sensor, we will place on our device, we can follow instant data and process this data. GPS sensor is one of the basic elements of our project. Because the coordinates are taken and processed via the GPS sensor and the directions will be created in this way. For this reason, a high sensitivity GPS sensor should be used. UBLOX GY-NEO6MV2 GPS Module was used in this project in terms of costs, stock status, and resource diversity. With the NEO6 IC on the GY-NEO6MV2 GPS module, it can perform location control and tracking using GPS in many projects, especially flight control systems. With this module, which has a sensitivity of 5 meters, high quality and precise location information are obtained. There is a 25x25mm ceramic antenna with the module. This module, which uses UART (RX-TX) communication unit, can be easily used in

our projects. In order to receive data from the GPS module via the raspbian serial console, the following steps are made on the raspbian console screen:[16]

1- First of all, it opens to edit the config.txt file as below.

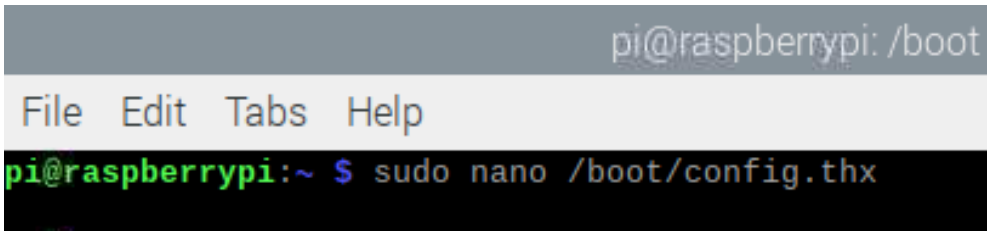


Figure 4.1.7.1

2- To the end of the config.txt file
"dtparam=spi=on
dtoverlay=pi3-disable-bt
core_freq=250
enable_uart=1
force_turbo=1"
it should be saved by adding the lines, pressing Ctrl+X, then pressing Y.

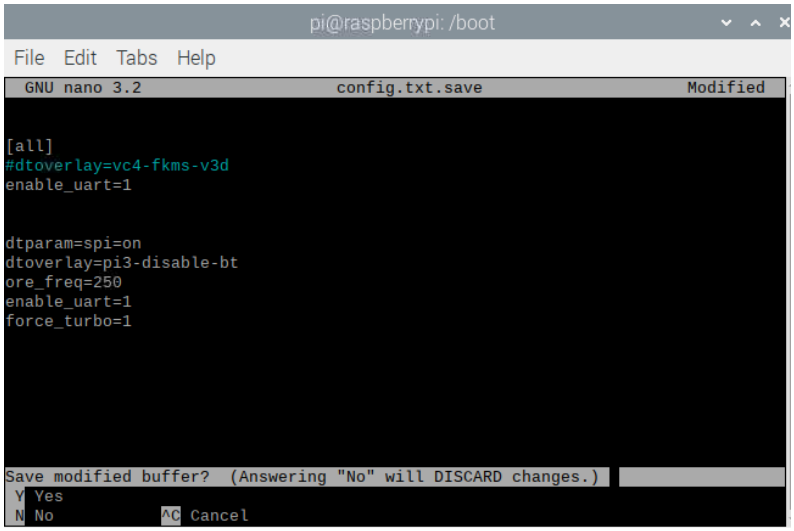


Figure 4.1.7.2

3- Raspbian uses the UART as a serial console and so we need to turn off that functionality. But before doing this, the cmdline.txt file should be backed up for security purposes.

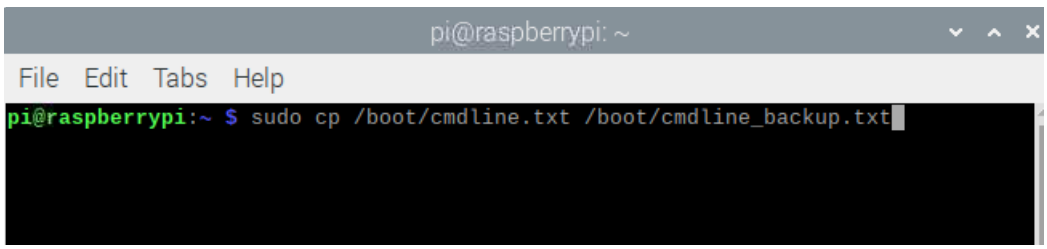


Figure 4.1.7.3

4- cmdline.txt file is opened with the command below.

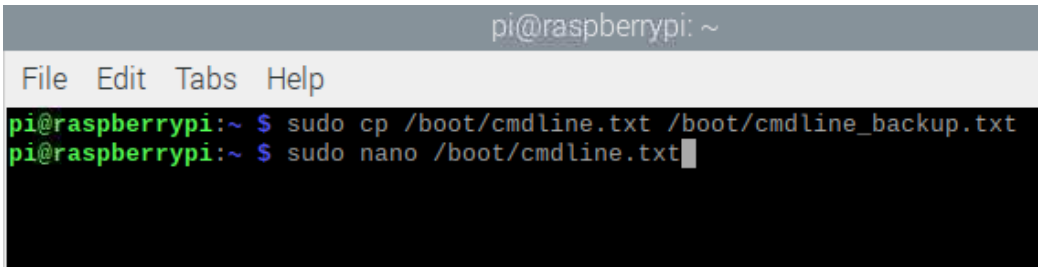


Figure 4.1.7.4

5- "dwc_otg.lpm_enable=0 console=tty1 root=/dev/mmcblk0p2 rootfstype=ext4 elevator=deadline fsck.repair=yes rootwait quiet splash plymouth.ignore-serial-consoles"
Copy the line and paste it into the cmdline.txt file.

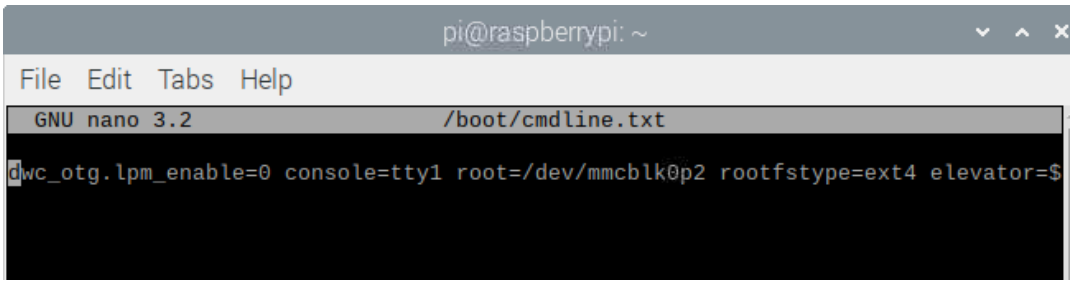


Figure 4.1.7.5

6- After completing the operations, the raspbian operating system is restarted.

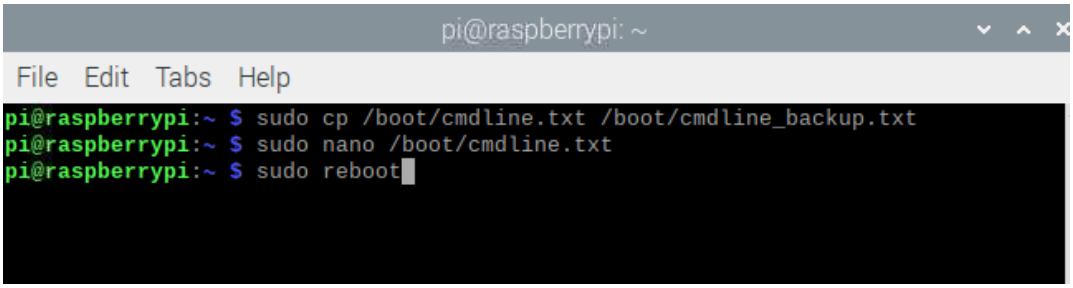


Figure 4.1.7.6

7- After connecting the sensor with raspberry pi, you can connect to the serial-0 screen with "ttyAMA0" by typing the following command on the console screen.

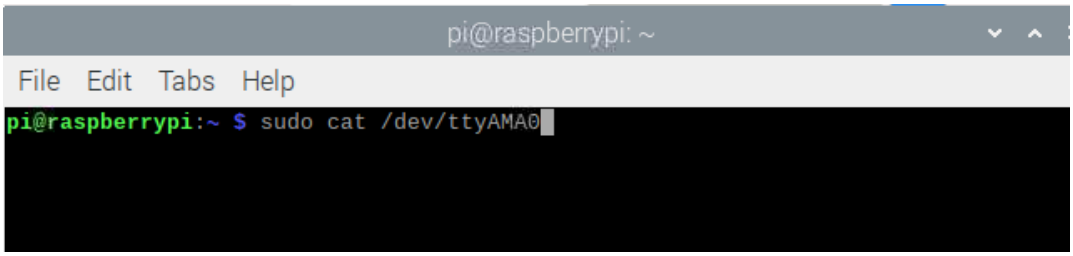


Figure 4.1.7.7

It is necessary to be in the open area in order to get the value from the sensor, and if the blue led flashes on it, the sensor is active, otherwise, the sensor cannot send data.

Figure 4.1.7.8

The sensor data is as follows and the values continue to flow until the serial port is closed.

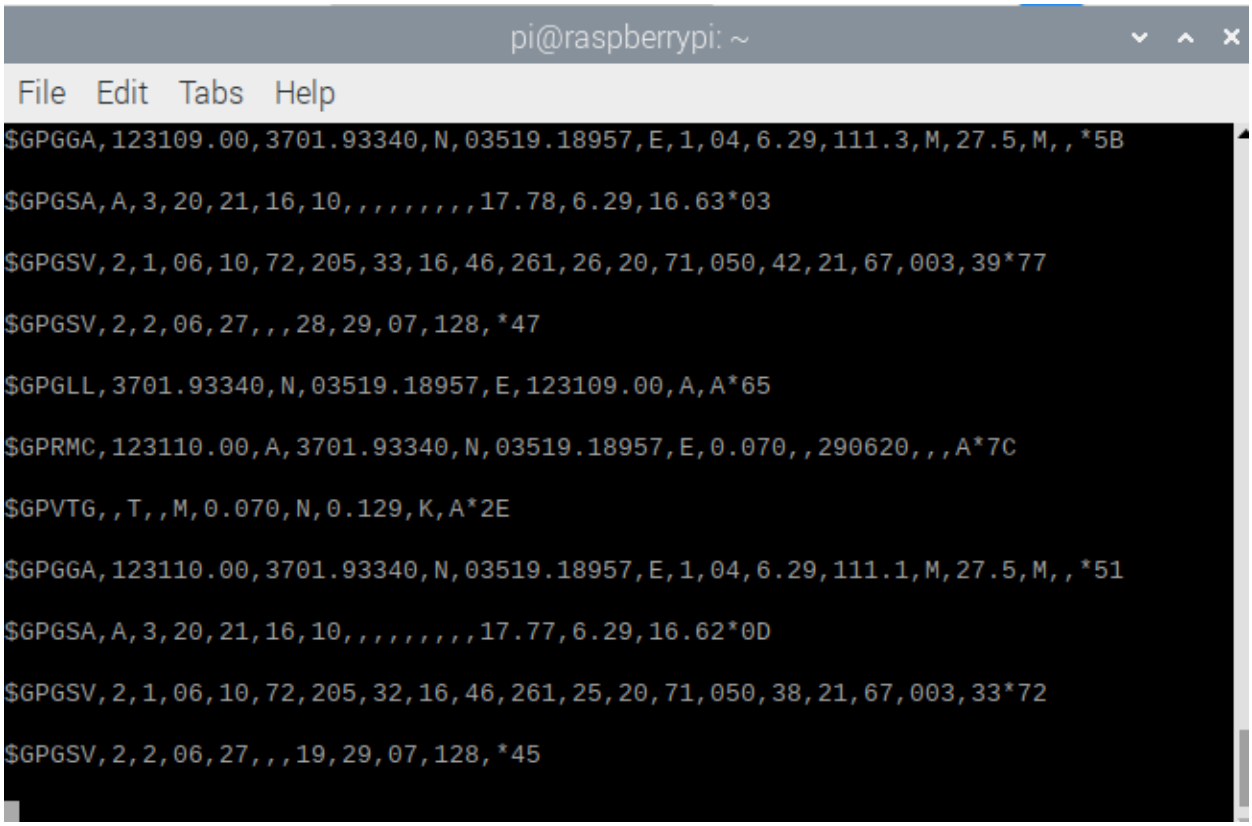


Figure 4.1.7.9

Not all of the above values are used in location determination. $GPGGA data is used to find the point where the quadcopter is located. $GPGGA-Global Positioning System Fix Data. The parameters of this value are as follows.

Name or Field	Example	Description
Sentence Identifier	$GPGGA	GGA protocol header
Time	123.109	12:31:09
Latitude	3701,9334,N	ddmm.mmmm
Longitude	3519.18957,E	dddmm.mmmm
Fix Quality:	1	Data is from a GPS fix
Number of Satellites	4	4 Satellites are in view
HDOP	6,29	Relative accuracy of horizontal position
Altitude	111.3 ,M	111.3 meters above mean sea level
Height of geoid above WGS84 ellipsoid	27.5 ,M	27.5 meters
Time since last DGPS update	Blank	No last update
DGPS reference station id	Blank	No station id
Checksum	*58	Used by program to check for transmission errors

Figure 4.1.7.10

4.1.8 Inertial Measurement Unit (IMU)

The IMU is an electronic unit that collects angular velocity and linear acceleration data sent to the main processor in a single module. IMU consists of gyroscope and acceleration sensors. Gyroscope and accelerometer cannot provide safe and stable data separately. Therefore, two sensors are combined by taking each other as a reference, and information such as speed and position are taken from a single unit, namely the IMU. The term Degrees of Freedom (DOF) characterizes the degree of freedom of the IMU. It is expressed as an IMU 6DOF with 3 axis gyro and 3 axis accelerometer. In the project, this operation was done using the Kalman Filter. [17]

4.1.8.1 Gyroscope and Accelerometer

We will need an accelerometer and gyro sensor to balance our quadcopter and make our rotations in a controlled manner. For this, we need a 3-axis acceleration and a 3-axis gyro sensor. This integrated IMU structure that we will get 6 axis movement in total is the MPU-6050 sensor. Thanks to this sensor, we can receive these two different data on one device. Our sensor will give the acceleration in x-y-z coordinates in m/s and the motion in degrees/seconds (angular velocity) thanks to the gyroscope sensor. As it is understood from the name of acceleration, it gives the acceleration of the device in the direction of the axis, that is, acceleration, and the gyroscope gives us the motion information of the device. Gyroscope information in a fixed position is taken as zero or a value very close to zero. Our sensor communicates with raspberry pi over the I^2C protocol. We need 4 cables to connect our sensor to the device. These are VCC, GND, SCL and SDA pins.

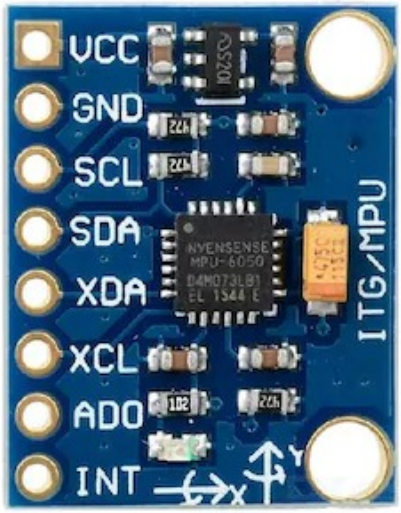


Figure 4.1.8.1.1

4.1.8.2 Air Pressure Sensor

In our project, the BMP180 Digital Air Pressure Sensor was used to measure the height of the quadcopter. The BMP180 digital outlet pressure sensor is a very small and very useful product that digitally outputs by measuring the air pressure in the environment. Due to the understanding of the sensor height according to the air pressure, it can be used in many systems, especially in multicopter projects. We can make altitude calculations according to the air pressure information we receive on this sensor and use this data in our device. BMP180 can measure the pressure value between 300-1100hPa and gives information about the altitude between 500 meters and 9000 meters. It has a very high resolution, such as 0.03 hPa (0.25 meters). High precision data supply is one of the main reasons why we prefer this device. Our device operates with 3.3V input voltage and is connected to our controller with I^2C protocol. We can connect to our raspberry pi device via SCA and SDA pins.

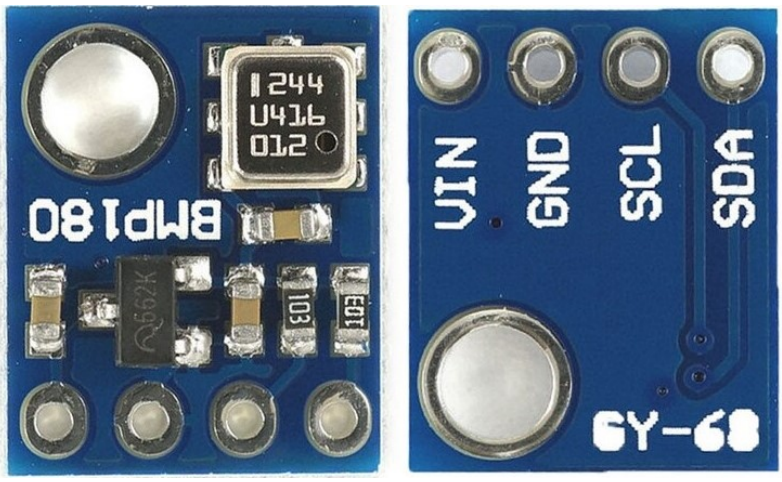


Figure 4.1.8.2.1

The average pressure of the atmosphere at sea level is 1013.25 hPa (or mbar). This drops off to zero as you climb towards the vacuum of space. Because the curve of this drop-

off is well understood, you can compute the altitude difference between two pressure measurements (p and p0) by using this equation:

$$altitude = 44330 * \left(1 - \left(\frac{p}{p_0}\right)^{\frac{1}{5.255}}\right)$$

Figure 4.1.8.2.2

We have two options here. If we take the device p0 as the sea level pressure, we will measure our altitude (altitude) from the sea level. If we set the p0 value as the pressure of the current position, the current position will be accepted as the zero points and the calculation will be made, that is, our relative altitude value will be calculated. [18]

- MPU6050 and BMP180 sensors can be made using the console screen as below to check the I2C connections on the raspberry pi device.

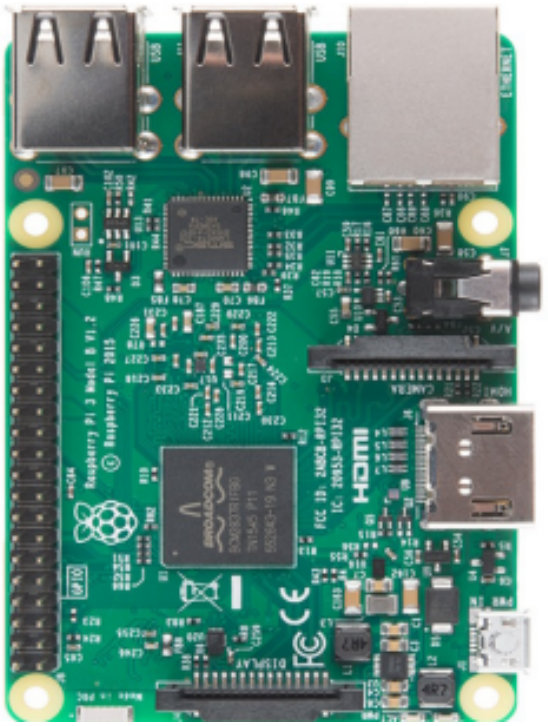
Figure 4.1.8.1

4.2 SOFTWARE

4.2.1 Raspberry Pi

In this project, Raspberry pi 3 Model B was used as a control card. The biggest factor in choosing this card is its processing speed. The technical features of the card such as GPU and RAM are available in Figure 4.2.1.1. This card was chosen because it will greatly affect the quadcopter's stability in the air, as well as the speed of processing sensor data, reading and evaluating GPS data, as well as the speed of PWM signals to be sent to the motors. Also,

raspberry pi provides the ideal environment for machine learning with easy internet access, which creates the most suitable ground for advancing the project. In this project, it is the ideal environment for determining the quadcopter route by processing the GPS data. In addition, using Rasbian, the open-source Linux operating system, extends the limits of what can be done. So much so that it can be used to drive a motor by adding a kernel module to the Linux operating system, as done in this project. It will be explained in detail in the engine software section. [19]



	Raspberry Pi 3 Model B
Processor Chipset	Broadcom BCM2837 64Bit ARMv7 Quad Core Processor powered Single Board Computer running at 1250MHz
GPU	Videocore IV
Processor Speed	QUAD Core @1250 MHz
RAM	1GB SDRAM @ 400 MHz
Storage	MicroSD
USB 2.0	4x USB Ports
Power Draw / voltage	2.5A @ 5V
GPIO	40 pin
Ethernet Port	Yes
Wi-Fi	Built in
Bluetooth LE	Built in

Figure 4.2.1.1

4.2.1.1 Raspbian Setup

1- First of all, at least 4 GB in size and SDHC compatible SD cart is required to install the operating system on our Raspberry pi card. 16 Gb was used in this project.

2- It can download the operating system software from www.raspberrypi.org. The Lite version of the software does not support the desktop, if it is desired to use the HDMI output and use its features other than the console screen, the Lite version is not suitable.

3- The SD card to be used is formatted using the SD Card Formatter program.

4- Writing the SD card of the operating system with the extension .img using Win32 Disk Imager application.

As a result of these steps, our operating system has been prepared for installation. After inserting the SD card to the Raspberry pi card, the VNC program was used by following the steps below in order not to need a monitor to see the setup and screen.

1- In order to use the VNC program, Raspberry pi is connected to the router with a LAN cable, and an internet connection is provided and the IP address is determined. Finding the IP address can be determined from the router settings or with the IPScanner program.

2- Put the IP address of the Raspberry pi device into the Putty application and connect it to the console screen of the device. Before this process, the ssh.txt file should be added to the boot file, so that ssh is enabled and access to the shell is not allowed, if this operation is not performed, the Putty application cannot be used.

3- When connected to the console screen, it asks us for a password to log in first. In the initial setup, the username: pi , password: raspberry.

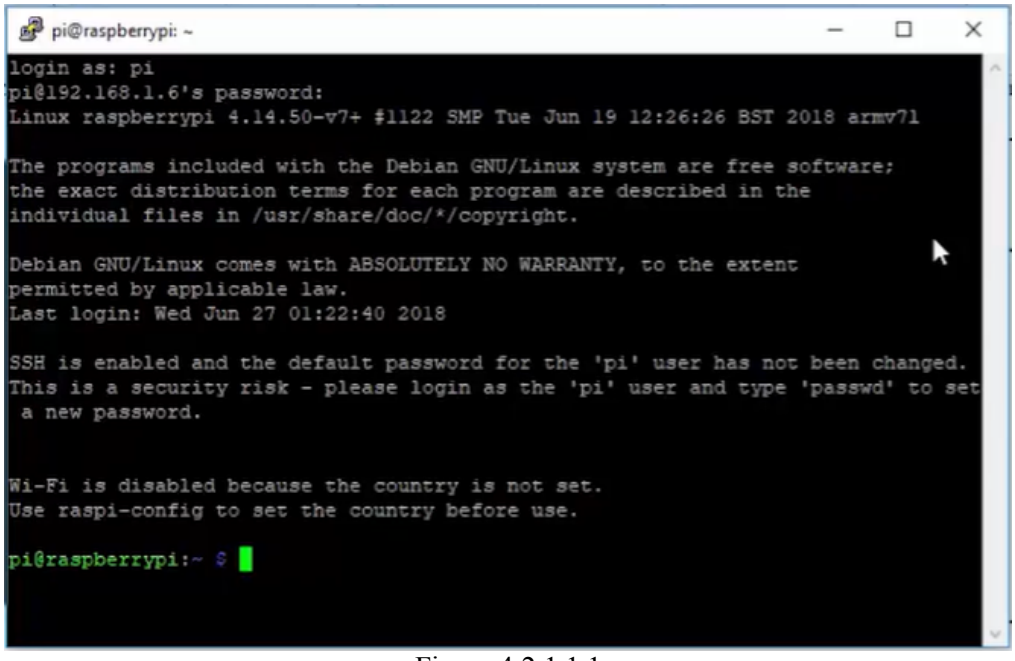


Figure 4.2.1.1.1

4- After reaching the console screen of the device, the sudo raspi-config command is entered, and the VNC feature is enabled from the settings in the following image (Interfacing Options).

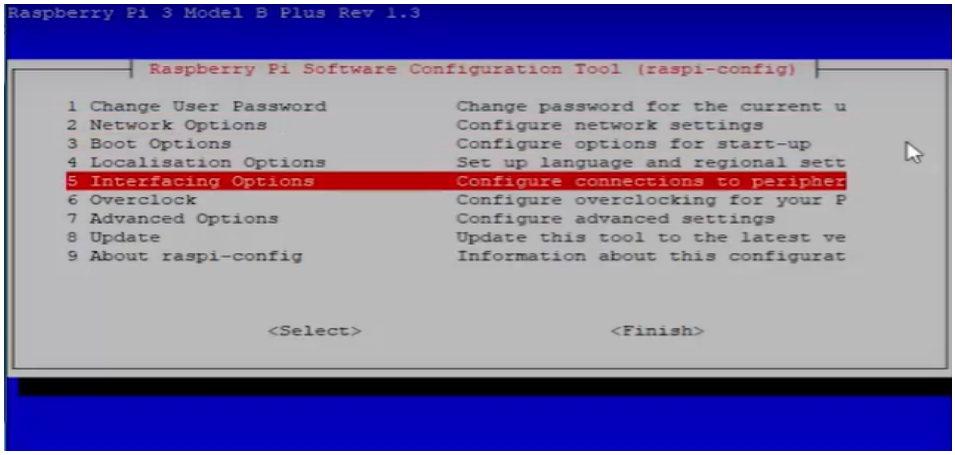


Figure 4.2.1.1.2

5- By typing the IP address of Raspberry pi into the VNC application, the screen of the device was connected. The reason for choosing the VNC application to use Raspberry Pi is that the device provides usability anytime, anywhere, even by opening the internet of your phone.

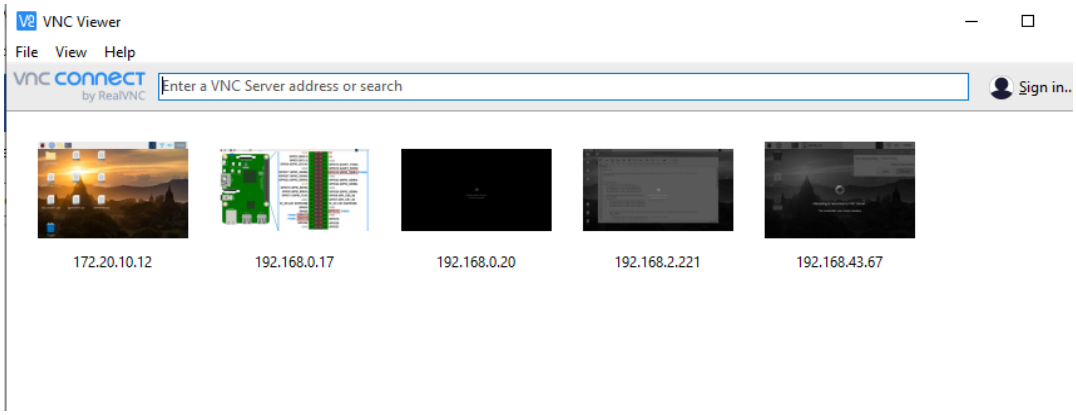


Figure 4.2.1.1.2

6- When the Raspberry pi screen opens, the initial setup settings are made. In this section, changing the password is very important for security. It is then rebooted and ready to use the device. Using the interface, settings such as I^2C, SPI can be easily activated.

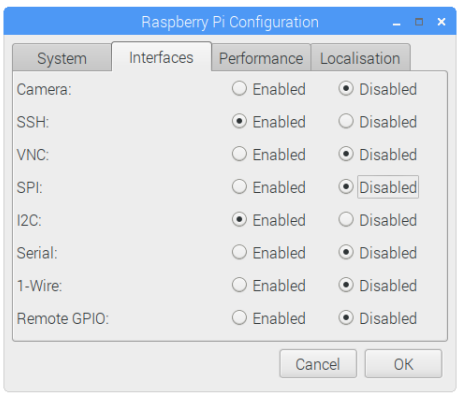


Figure 4.2.1.1.3

4.2.1.2 Python Version Update

First, the whole system is updated by using "sudo apt update" and "sudo apt full-upgrade" commands. The default version of python is 2.7 when the Raspbian operating system is installed, and the required libraries are installed with the "pip" command when writing the python code, so. But the python libraries used in the software "pip3" and its commands are used in the python3 version, so Python 3.7 is default by following the steps below.

1- All python versions are listed with the command ls / usr / bin / python *.

2- The default version is displayed with the python --version command.

3- The shell file is opened with the sudo nano ~ / .bashrc command. Bashrc functions as a shell script that runs every time the user opens the terminal.

4- alias python = '/ usr / bin / python3.7' and alias pip = pip3 alias lines are added to the .bashrc file and saved.

The latest 3.7 version released in the project was default. If the new version comes after the update process, it can be seen when all versions are listed.

5- . ~ / .bashrc This is the command to refresh the .bashrc file.

6- update-alternatives -- list python We can see the versions that will be used to renew the python version.

7- sudo update-alternatives --install /usr/bin/python python /usr/local/bin/python3.7.2 and sudo update-alternatives --install /usr/bin/pip pip /usr/local/bin/pip3 using these lines, the last version is set as default. This section is especially important. With "Alias" lines, the path is determined for python commands, but the old version is still visible by default. To change this, the new version is default with these commands.

8- It is controlled by the python --version command.

Since the Python programming language is a language whose constant syntax is renewed and new commands and libraries are added, it is useful to use the current version by performing these operations carefully. Otherwise, especially the syntax of the commands we use may not work in the old version, and the code that is actually correct may fail due to the version. In this project, library addition and execution errors were solved with updates and default refresh processes. [20]

4.2.1.3 Generating PWM Signal on Raspberry Pi

In order for the quadcopter to perform pitch, roll, and yaw movements, different PWM signals must be sent to four motors at the same time. But when we look at the Raspberry pi 3 datasheet, we can generate two different PWM outputs even though we see four PWM outputs.

GPIO Pin	PWM0/PWM1
GPIO12	PWM0
GPIO18	PWM0
GPIO13	PWM1
GPIO19	PWM1

Figure 4.2.1.3.1

In fact, there are multiple ways to solve this problem as hardware and software.

1. HARDWARE PWM

In order to generate more than two PWM signals, PCA9685 I2C to PWM converter card can be added as hardware, but since IMU sensors are connected to I2C GPIOs, motors and sensors can be connected as slaves. But if we use this transducer, we can send signals to the motors and sensors, respectively, and we cannot send signals to the motors during the feedback from the sensor. It is not possible to use this method for quadcopter design, so the vehicle cannot stay in balance.

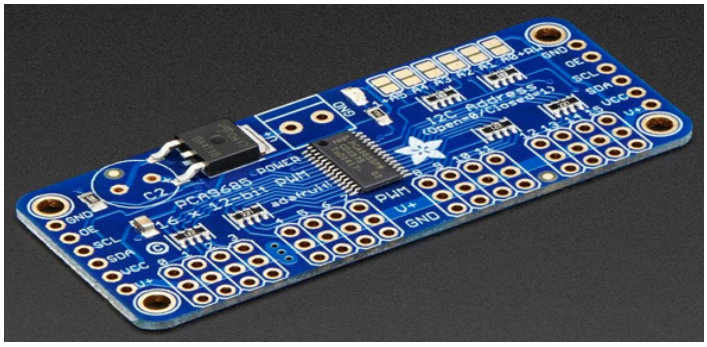


Figure 4.2.1.3.1

2. SOFTWARE PWM

As hardware is not available, ways to produce PWM with software were investigated. WiringPi library used to drive servo was investigated.

WiringPi

WiringPi includes a software-driven PWM handler capable of outputting a PWM signal on any of the Raspberry Pi's GPIO pins. To maintain a low CPU usage, the minimum pulse width is 100µS. That combined with the default suggested range of 100 gives a PWM frequency of 100Hz.But ESCs used in Radio Control applictions take standard Radio Control "PWM" signals which isn't really PWM in the classic sense, but a pulse width from 1 to 2mS. Where 1mS is hard left and 2ms is hard right, or fully on for a motor.WiringPi module can not generate this sort of signal. That's why the library is not available.

Servoblaster

Servoblaster is actually a kernel module. After downloading this module, the Linux kernel module can be loaded using the Makefile file, but for the Raspberry pi 3 model B used in the project, it can use this kernel module differently. The Linux operating system kernel is updated, though not as often, like operating systems. For this reason, servoblaster could not be added to the new Rasbian operating system with the kernel update method. The module was used with the userspace daemon method by following the steps below.

1- Firstly, to download the servoblaster module from the Github to the raspberry pi card, the git function was loaded with the command sudo apt-get install git

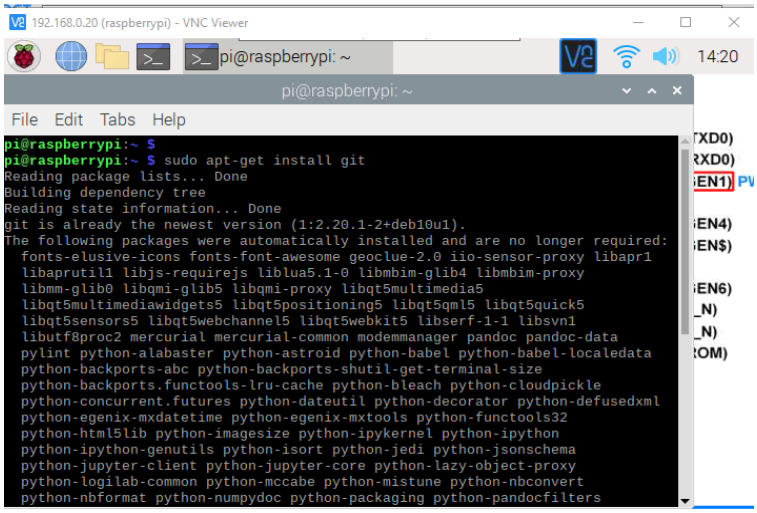


Figure 4.2.1.3.2

2- Copying the servoblasts Github link was downloaded with the command sudo git clone https://github.com/srcshelton/servoblaster.git

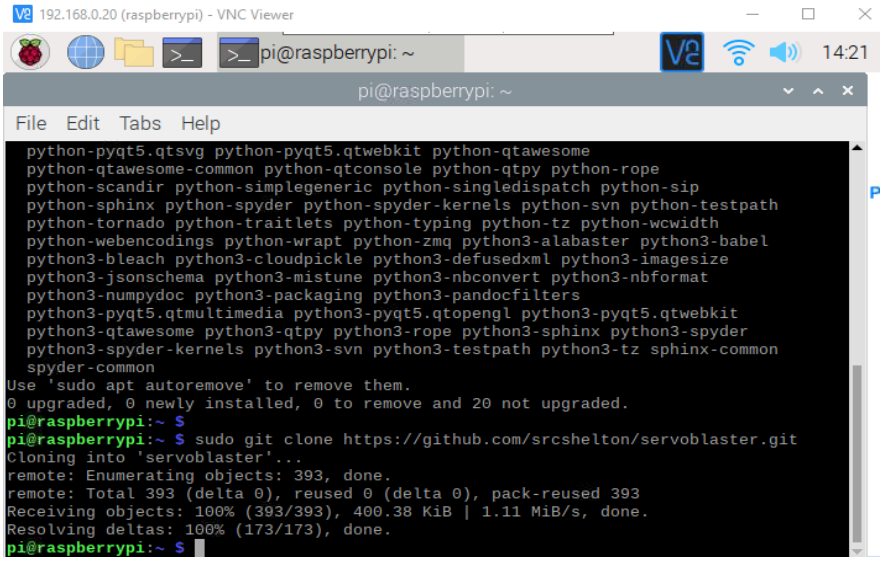


Figure 4.2.1.3.3

3- The file location was reached with the command cd /home/pi/PiBits/ServoBlaster/user on the console screen for use with the Servoblaster userspace daemon method.

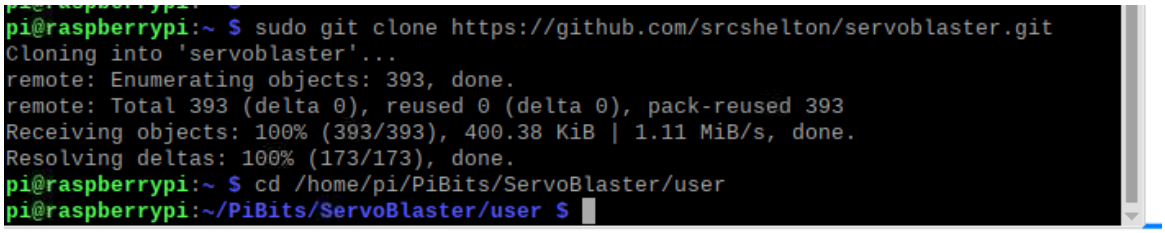


Figure 4.2.1.3.4

4- The servod.c file was loaded with the sudo make servod or sudo make install command.

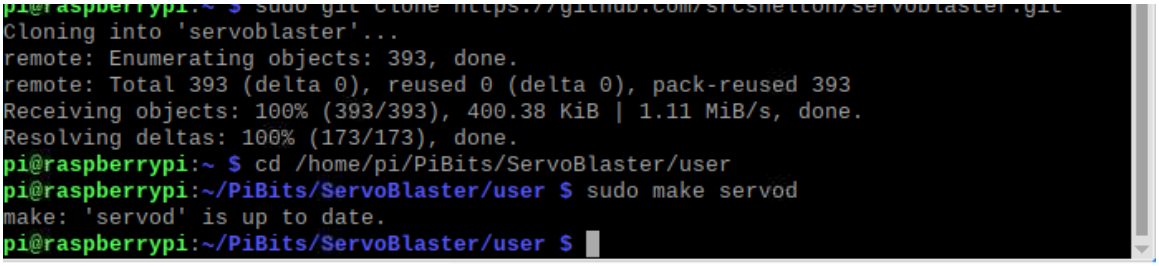


Figure 4.2.1.3.5

5- The ls command was used to list the content in the User location and the servod (green color) was loaded as shown in the image.

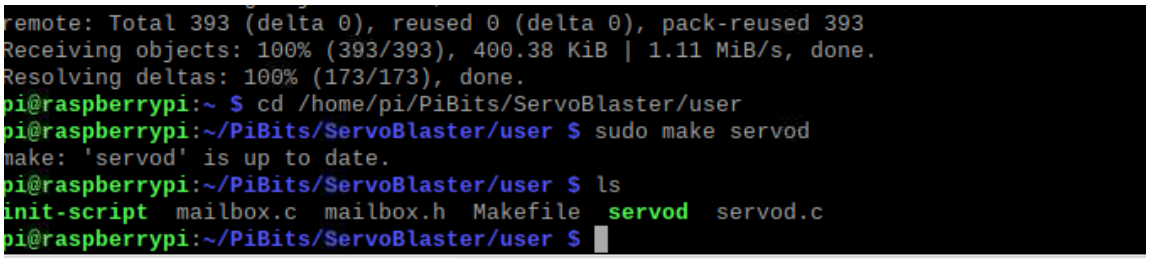


Figure 4.2.1.3.6

6- The sudo ./servod command shows the technical features of the module, which pins are used as PWM output.

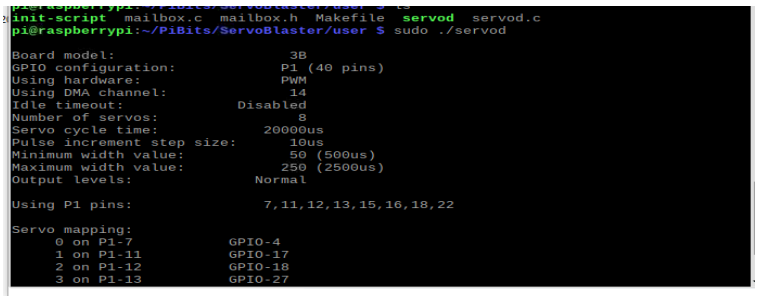


Figure 4.2.1.3.7

7- The PWM signals received from the selected pins are as follows.

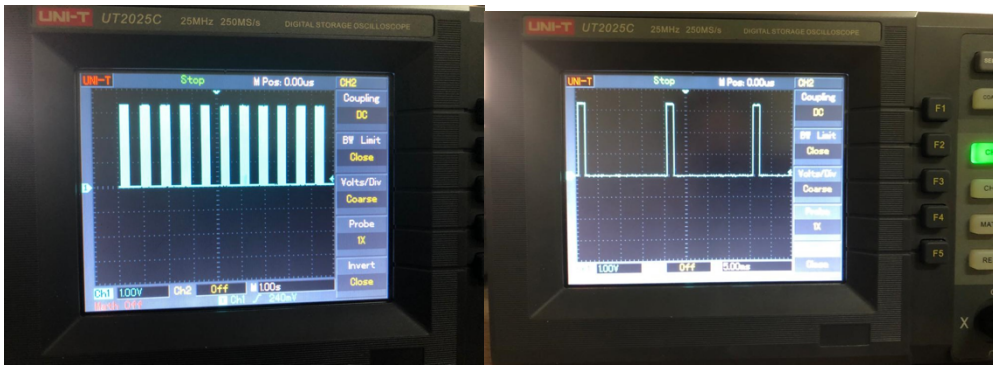


Figure 4.2.1.3.8

This is software for the RaspberryPi, which provides an interface to drivemultiple servos via the GPIO pins. You control the servo positions by sendingcommands to the driver saying what pulse width a particular servo output shouldbuse.

By default is it configured to drive 8 servos, although you can configure it todrive up to 21. Servos typically need an active high pulse of somewherebetween 0.5ms and 2.5ms, where the pulse width controls the position of theservo. The pulse should be repeated approximately every 20ms, although pulsefrequency is not critical. The pulse width is critical, as that translatesdirectly to the servo position.

Commands that can be used to make changes to the basic settings:

Usage: ./servod <options>
Options:

--pcm	tells servod to use PCM rather than PWM hardware to implement delays
--idle-timeout=Nms	tells servod to stop sending servo pulses for a given output N milliseconds after the last update
--cycle-time=Nus	Control pulse cycle time in microseconds, default 20000us
--step-size=Nus	Pulse width increment step size in microseconds, default 10us
--min={N|Nus|N%}	specifies the minimum allowed pulse width, default 50 steps or 500us
--max={N|Nus|N%}	specifies the maximum allowed pulse width, default 250 steps or 2500us
--invert	Inverts outputs
--dma-chan=N	tells servod which dma channel to use, default 14
--p1pins=<list>	tells servod which pins on the P1 header to use
--p5pins=<list>	tells servod which pins on the P5 header to use

The driver creates a device file, /dev/servoblaster, in to which you can sendcommands. The command format is either
<servo-number>=<servo-position> or P<header>-<pin>=<servo-position>

For the first format <servo-number> is the sevo number, which by default is anumber between 0 and 7, inclusive. For the second format <header> is either '1' or '5', depending on which header your servo is connected to, and <pin> isthe pin number on that header you have

connected it to. By default <servo-position> is the pulse width you want in units of 10us, although thatcan be changed via command line arguments, and can also be specified in unitsof microseconds, or as a percentage of the maximum allowed pulse width.

So, if you want to set servo 3 to a pulse width of 1.2ms you could do this atthe shell prompt:
echo 3=120 > /dev/servoblaster
120 is in units of 10us by default, so that is 1200us, or 1.2ms.
Alternatively, using the second command format, if you had a servo connected toP1 pin 12 you could set that to a width of 1.2ms as follows:
echo P1-12=120 > /dev/servoblaster

The code defaults to driving 8 servos, the control signals of which should beconnected to P1 header pins as follows:

Servo number	GPIO number	Pin in P1 header
0	4	P1-7
1	17	P1-11
2	18	P1-12
3	21/27	P1-13
4	22	P1-15
5	23	P1-16
6	24	P1-18
7	25	P1-22

Figure 4.2.1.3.9

The servoblaster module and operating commands are described in detail during the motor driving process. [21]

4.2.2 Motor Control

4.2.2.1 Calibration

Calibration is required first to perform a brushless motor check. Calibration is done to notify the engine and motor driver of the range they can operate. In this project, calibration is done as follows:

- By disconnecting the battery, a maximum value signal is sent to the motor. In this project, the maximum value is determined as 2450us. (Important: Normally 2500us is the max value of servoblaster module, but when the motor was calibrated with 2500us, it was not possible to control all of the motors at the same time.)

```
def calibration():
    for x in gpio_pwm:
        os.system("echo P1-"+ str(x) +"=0> /dev/servoblaster")
        print("echo P1-"+ str(x) +"=0> /dev/servoblaster")
    time.sleep(1)
    print("Disconnect the battery and press Enter")
    inp = input()
    if inp == '':
        for x in gpio_pwm:
            os.system("echo P1-"+ str(x) +"=2450us> /dev/servoblaster") # Max value of
            print("echo P1-"+ str(x) +"=2450us> /dev/servoblaster")
        print("Connect the battery NOW.. you will here two beeps, then wait for a gradua
```

Figure 4.2.2.1.1

- The battery is connected and after hearing two beeps from the engine, a minimum value signal is sent to the motors.

```
print("Connect the battery NOW.. you will here two beeps, then wait for a gradua
inp = input()
if inp == '':
    for x in gpio_pwm:
        os.system("echo P1-"+ str(x) +"=1500us> /dev/servoblaster") # Min value
        print("echo P1-"+ str(x) +"=1500us> /dev/servoblaster")
    print("wait ring tone for a few secons")
    time.sleep(6)
    print("Please, wait again")
    time.sleep(4)
```

Figure 4.2.2.1.2

- After waiting for a few seconds, the arming process is done by sending the stop signal and minimum value signal and the motor is ready to run at the desired speed.

```
print("wait ring tone for a few secons")
time.sleep(6)
print("Please, wait again")
time.sleep(4)
print("Please, wait again for arming")
for x in gpio_pwm:
    os.system("echo P1-"+ str(x) +"=0> /dev/servoblaster")
    print("echo P1-"+ str(x) +"=0> /dev/servoblaster")
time.sleep(2)
print("Arming ESC now ...")
for x in gpio_pwm:
    os.system("echo P1-"+ str(x) +"=1500us> /dev/servoblaster")# Min value
    print("echo P1-"+ str(x) +"=1500us> /dev/servoblaster")
time.sleep(1)
```

Figure 4.2.2.1.3

4.2.2.2 Controls

Since motor control commands are done with bash commands, os library is used to run these commands on the console. The library can be loaded with the pip install os command. Also, loops must be used in the software, since the signal is sent with the bash command.

Take – Off

In order for the quadcopter to take off, four motors must be synchronized at the same time. It gradually increases speed and continues until it takes off.

```
def elevation():
    for a in speed:
        for x in gpio_pwm:
            os.system("echo P1-"+ str(x) +"="+ str(a) +"us > /dev/servoblaster")
            print("echo P1-"+ str(x) +"="+ str(a) +"us > /dev/servoblaster")
        time.sleep(4)
```

Figure 4.2.2.2.1

Move Forward

For the quadcopter to move forward, the front motors remain constant at the speed they can stay in the air, while the rear motors are gradually increased in speed, allowing them to move forward. The increase interval was made with 10us. If significant speed increases are made, the motors stop with an error. Therefore, speed change should be done gradually with small intervals.

```
def move_forward():
    for a in forward:
        for x in forward_pwm:
            os.system("echo P1-"+ str(x) +"="+ str(a) +"us > /dev/servoblaster")
            print("echo P1-"+ str(x) +"="+ str(a) +"us > /dev/servoblaster")
            os.system("echo P1-11=1570us > /dev/servoblaster")
            print("echo P1-11=1570us > /dev/servoblaster")
            os.system("echo P1-12=1570us > /dev/servoblaster")
            print("echo P1-12=1570us > /dev/servoblaster")
        time.sleep(4)
```

Figure 4.2.2.2.2

Right / Left Turn

In order to move to the right, the speeds of the motors on the left should be increased. Variable a represents the motor pins on the left and variable x is the speed. The same process can be applied to the motors on the right, for turning left.

```
def turn_right():
    for a in right:
        for x in right_pwm:
            os.system("echo P1-"+ str(x) +"="+ str(a) +"us > /dev/servoblaster")
            print("echo P1-"+ str(x) +"="+ str(a) +"us > /dev/servoblaster")
        time.sleep(8)
```

Figure 4.2.2.2.3

```
def turn_left():
    for a in left:

        for x in left_pwm:
            os.system("echo P1-"+ str(x) +"="+ str(a) +"us > /dev/servoblaster")
            print("echo P1-"+ str(x) +"="+ str(a) +"us > /dev/servoblaster")
        time.sleep(8)
```

Figure 4.2.2.2.4

Turning Off

A function to close all PWM signals to be used after landing should be written.

```
def stop():
    for x in gpio_pwm:
        os.system("echo P1-"+ str(x) +"=0> /dev/servoblaster")
        print("echo P1-"+ str(x) +"=0> /dev/servoblaster")

    os.system('sudo killall /home/pi/PiBits/ServoBlaster/user/servod')
```

Figure 4.2.2.2.5

In order for the motors to run smoothly and synchronously in the air and maintain their position, the axis values coming from the IMU unit must be filtered, and the obtained axis values must be added to the motor values by passing them through PID control and speed balancing must be done.

4.2.3 Global Positioning System (GPS) Software

- NMEA (The National Marine Electronics Association) information is the data we can get from the NEO-6M-GPS module. This data set contains many variables and GPGGA is used in this project, global position data. Time, latitude, and longitude data to be used in this data can be taken as follows. [22]

```
def GPS_Info():
    global NMEA_buff
    global lat_in_degrees
    global long_in_degrees
    nmea_time = []
    nmea_latitude = []
    nmea_longitude = []
    nmea_altitude=[]
    nmea_time = NMEA_buff[0]                    #extract time from GPGGA string
    nmea_latitude = NMEA_buff[1]                #extract latitude from GPGGA string
    nmea_longitude = NMEA_buff[3]               #extract longitude from GPGGA string
    nmea_altitude=NMEA_buff[6]
```

Figure 4.2.3.1

- Since GPGGA data is taken as a string from the data set, it can be converted to degrees as follows.

```
lat = float(nmea_latitude)                  #convert string into float for calculation
longi = float(nmea_longitude)               #convertr string into float for calculation

lat_in_degrees = lat/100    #get latitude in degree decimal format
long_in_degrees =longi/100 #get longitude in degree decimal format
```

Figure 4.2.3.2

- Serial screen opening command to read the values from the serial port screen:

```
gpgga_info = "$GPGGA,"
ser = serial.Serial ("/dev/ttyAMA0",9600)              #Open port with baud rate
GPGGA_buffer = 0
NMEA_buff = 0
lat_in_degrees = 0
long_in_degrees = 0
```

Figure 4.2.3.3

- The .split () function can be used to use latitude and longitude from the received GPGGA data. Thus, the desired data can be obtained from a line such as $GPGGA,123519,4807.038,N,01131.000,E,1.08,0.9,545.4,M,46.9,M ,,* 47.

```
try:
    while True:
        received_data = (str)(ser.readline())                  #read NMEA string received
        GPGGA_data_available = received_data.find(gpgga_info)  #check for NMEA GPGGA string
        if (GPGGA_data_available>0):
            GPGGA_buffer = received_data.split("$GPGGA,",1)[1] #store data coming after "$GPGGA," string
            NMEA_buff = (GPGGA_buffer.split(','))              #store comma separated data in buffer
            GPS_Info()                                         #get time, latitude, longitude
```

Figure 4.2.3.4

- If the data in the GPS module is desired to be displayed on google maps, web browser library can be added and displayed on the internet as follows. (Values must be in str format.)

```
            print("lat in degrees:", lat_in_degrees," long in degree: ", long_in_degrees, '\n')
            map_link = 'http://maps.google.com/?q=' + str(lat_in_degrees) + ',' + str(long_in_degrees)
            print("<<<<<<<<press ctrl+c to plot location on google maps>>>>>>\n")              #press
            print("------------------------------------------------------------\n")

except KeyboardInterrupt:
    webbrowser.open(map_link)        #open current position information in google map
    sys.exit(0)
```

Figure 4.2.3.5

Output of Neo-6M-GPS:

```
lat in degrees: 37.0192305  long in degree:  35.1920406

<<<<<<<<press ctrl+c to plot location on google maps>>>>>>
```

Figure 4.2.3.6

4.2.4 MPU6050 Sensor Software

- We need to write the register addresses of the data to be received from the sensor. Addresses can be obtained from the datasheet of the sensor, but it is easy to use because it is a commonly used sensor.

```
#some MPU6050 Registers and their Address
PWR_MGMT_1   = 0x6B
SMPLRT_DIV   = 0x19
CONFIG       = 0x1A
GYRO_CONFIG  = 0x1B
INT_ENABLE   = 0x38
ACCEL_XOUT_H = 0x3B
ACCEL_YOUT_H = 0x3D
ACCEL_ZOUT_H = 0x3F
GYRO_XOUT_H  = 0x43
GYRO_YOUT_H  = 0x45
GYRO_ZOUT_H  = 0x47
```

Figure 4.2.4.1

- A function must be created to write the data from the sensor to predetermined addresses.

```
def MPU_Init():
    #write to sample rate register
    bus.write_byte_data(Device_Address, SMPLRT_DIV, 7)
    #Write to power management register
    bus.write_byte_data(Device_Address, PWR_MGMT_1, 1)
    #Write to Configuration register
    bus.write_byte_data(Device_Address, CONFIG, 0)
    #Write to Gyro configuration register
    bus.write_byte_data(Device_Address, GYRO_CONFIG, 24)
    #Write to interrupt enable register
    bus.write_byte_data(Device_Address, INT_ENABLE, 1)
```

Figure 4.2.4.2

- Function created to read data in registers:

```
def read_raw_data(addr):
    #Accelero and Gyro value are 16-bit
        high = bus.read_byte_data(Device_Address, addr)
        low = bus.read_byte_data(Device_Address, addr+1)
        #concatenate higher and lower value
        value = ((high << 8) | low)
        #to get signed value from mpu6050
        if(value > 32768):
                value = value - 65536
        return value

bus = smbus.SMBus(1)    # or bus = smbus.SMBus(0) for older version boards
Device_Address = 0x68   # MPU6050 device address

MPU_Init()
```

Figure 4.2.4.3

- With the data reading functions, gyroscope and accelerometer data are read and assigned to the variable.

```
#Read Accelerometer raw value
acc_x = read_raw_data(ACCEL_XOUT_H)
acc_y = read_raw_data(ACCEL_YOUT_H)
acc_z = read_raw_data(ACCEL_ZOUT_H)

#Read Gyroscope raw value
gyro_x = read_raw_data(GYRO_XOUT_H)
gyro_y = read_raw_data(GYRO_YOUT_H)
gyro_z = read_raw_data(GYRO_ZOUT_H)
```

Figure 4.2.4.4

- It is divided by the values specified in the datasheet to adjust the sensitivity of the data read from the sensor.

```
#Full scale range +/- 250 degree/C as per sensitivity scale factor
Ax = acc_x/16384.0
Ay = acc_y/16384.0
Az = acc_z/16384.0

Gx = gyro_x/131.0
Gy = gyro_y/131.0
Gz = gyro_z/131.0


print ("Gx=%.2f" %Gx, u'\u00b0'+ "/s", "\tGy=%.2f" %Gy, u'\u00b0'+ "/s", "\tGz=%.2f" %Gz, u'\u00b0'+ "/s",
sleep(1)
```

Figure 4.2.4.5

Output of MPU6050 Software:

```
Data of Gyroscope and Accelerometer
/s    Gy=0.00 °/s    Gz=0.02 °/s    Ax=0.00 g      Ay=0.00 g      Az=0.00 g
°/s   Gy=0.33 °/s    Gz=-0.32 °/s   Ax=0.08 g      Ay=0.03 g      Az=0.77 g
°/s   Gy=0.31 °/s    Gz=-0.31 °/s   Ax=0.09 g      Ay=0.04 g      Az=0.78 g
°/s   Gy=0.31 °/s    Gz=-0.34 °/s   Ax=0.08 g      Ay=0.04 g      Az=0.77 g
°/s   Gy=0.34 °/s    Gz=-0.32 °/s   Ax=0.08 g      Ay=0.04 g      Az=0.77 g
°/s   Gy=0.32 °/s    Gz=-0.34 °/s   Ax=0.09 g      Ay=0.04 g      Az=0.79 g
°/s   Gy=0.32 °/s    Gz=-0.31 °/s   Ax=0.09 g      Ay=0.04 g      Az=0.78 g
°/s   Gy=0.31 °/s    Gz=-0.33 °/s   Ax=0.09 g      Ay=0.04 g      Az=0.77 g
°/s   Gy=0.33 °/s    Gz=-0.32 °/s   Ax=0.08 g      Ay=0.04 g      Az=0.77 g
```

Figure 4.2.4.6

4.2.5 BMP180 Airpressure Sensor Software

For this sensor, the library in Github can be downloaded with the command git clone https://github.com/m-rtijn/bmp180 and altitude value can be obtained with the following

commands. However, in this project, the code was used with the library because it can cause errors because it is connected to the slave with the MPU6050 sensor and the arrangement is as follows.

```
if __name__ == "__main__":
    bmp = bmp180(0x77)


bmp = bmp180(0x77)

print("Temp: " + str(round(bmp.get_temp(),2)) + " Celcius")
print("Pressure: " + str(round(bmp.get_pressure()/100,2)) + " hPa")
print("Altitude: " + str(round(bmp.get_altitude(),2)) + " meter")
```

Figure 4.2.5.1

Output of BMP180 :

```
Temp: 29.23 Celcius
Pressure: 997.72 hPa
Altitude: 130.92 meter
>>>
```

Figure 4.2.5.2

CONCLUSIONS

Many necessary parts for the quadcopter are provided and the hardware part of the project is largely completed. The software of the sensors and motors used has been completed. However, control algorithms that will drive engines and balance the vehicle in the air have not been created with the data coming from the sensors. It was determined that different algorithms should be used for each flight movement. one pid controller and one lq controller for vertical movement, and a position controller and a trajectory controller for horizontal movement. There is a lot of research on mathematical calculations for these controls, but since many are simulations and models, it is very difficult to extract algorithms from these calculations. To solve this problem, it is necessary to work with the team carrying out an active project in the talk of unmanned aerial vehicles. The control algorithms created should be tested on the vehicle with an environment such as the Stewart platform and the most appropriate coefficients should be determined. But this project has made a great contribution to learn how to control quadcopters and how to use a raspberry pi card with Linux operating system.

REFERENCES

[1] https://en.wikipedia.org/wiki/Unmanned_aerial_vehicle
[2]https://www.researchgate.net/publication/321582123_INSANSIZ_HAVA_ARACLARI_TARIHCESI_TANIMI_DUNYADA_VE_TURKIYE_DEKI_YASAL_DURUMU
[3] https://en.wikipedia.org/wiki/ArduPilot
[4] https://ardupilot.org/copter/docs/common-apm25-and-26-overview.html#common-apm25-and-26-overview
[5] https://ardupilot.org/copter/docs/common-navio2-overview.html
[6] http://wiki.ros.org/Robots/Navio2
[7] https://ardupilot.org/copter/docs/common-pixhawk-overview.html
[8] https://docs.px4.io/v1.9.0/en/flight_controller/pixhawk.html
[9] Full Control of a Quadrotor, Samir Bouabdallah and Roland Siegwart Autonomous Systems Lab, Swiss Federal Institute of Technology, ETHZ Z¨urich, Switzerland, DOI: 10.1109/IROS.2007.4399042 · Source: IEEE Xplore
[10] Flight Stability and Automatic Control, Dr. Robert C. Nelson, Department of Aerospace and Mechanical Engineering University of Notre Dame, International Editions 1998
[11] Position and Trajectory Control of a Quadcopter Using PID and LQ Controllers, Axel Reizenstein, Master of Science Thesis in Electrical Engineering Department of Electrical Engineering, Linköping University, 2017
[12] Extended Kalman Filter Based Autonomous Flying System for Quadcopters, J. A. A. S. Somasiri, D. P. Chandima, A. G. B. P Jayasekara, 28 Sep 2018, Colombo, Sri Lanka
[13] https://www.dyd.org.tr/portfolio/mini-drone-icin-pervane-secimi/
[14] https://maker.robotistan.com/lipo-pil-rehberi/
[15] https://www.promodelhobby.com/blog/icerik/rc-esc-ne-ise-yarar-ozellikleri-ve-cesitleri-nelerdir
[16] https://sparklers-the-makers.github.io/blog/robotics/use-neo-6m-module-with-raspberry-pi/
[17] https://www.elektrikport.com/teknik-kutuphane/inertial-measurement-unit-(imu)-nedir/17369#ad-image-0
[18] https://learn.sparkfun.com/tutorials/bmp180-barometric-pressure-sensor-hookup-
[19]https://www.terraelectronica.ru/pdf/show?pdf_file=%252Fds%252Fpdf%252FT%252FTechicRP3.pdf
[20] https://learn.sparkfun.com/tutorials/python-programming-tutorial-getting-started-with-the-raspberry-pi/configure-your-pi
[21] https://github.com/richardghirst/PiBits/tree/master/ServoBlaster
[22] https://www.gpsinformation.org/dale/nmea.htm