



**MARMARA ÜNİVERSİTESİ**  
**TEKNOLOJİ FAKÜLTESİ**



**MEKATRONİK MÜHENDİSLİĞİ BÖLÜMÜ**

**MRM3006.1 Gömülü Sistemler ve Uygulamaları Dersi**

## **Proje2**

**Pid Konum Kontrolü Ve Bilgilerin Tcp ile Aktarılıp Grafiğinin Çizilmesi**

**SERHAT TUĞAN**

**170215026**

**ÖMER GÜNDOĞDU**

**171216821**

**Dr. Öğr. Üyesi Hüseyin YÜCE**

**-2018**

## Pid Konum Kontrolü Ve Bilgilerin Tcp ile Aktarilip Grafiğinin Çizilmesi

### Giriş

Bu proje uçan sistemlerin konum kontrolünün PID ile sağlanması için tasarlanmıştır. Raspberry pi , IMU sensöründen aldığı konum bilgileri doğrultusunda sistemin dengede kalmasını,hızlı veya yavaş hareket etmesini sağlar.Bu uçan sistemlerin havaya yükselmesi, havada dengede kalması ve yere alçalmasını sağlamaktadır.Bu projede sistemi oluşturmak için, Raspberry pi,IMU Sensörü,ESC,Güç adaptörü,Breadboard,Brushless Motor ve Mergeneli Platform kullanılmıştır.

Projede, IMU'dan alınan realtime verileri Raspberry pi de işlenir.Burada ki önemli kısım IMU'dan alınan realtime verilerini kullanarak motor sürücü(ESC) ile PWM sinyali üreterek Brushless motorun devir sayısını artırıp azaltarak sistemin havalanması, dengede kalması ve aşağıya inmesini sağlayarak uçan bir sistemin kontrolünü sağlıyoruz.

Birinci projeye ek olarak IMU Sensöründen alınan verilerin Tcp ile aktarilip grafiğinin çizilmesini ve değerlerin ekranda gösterilmesini sağlıyoruz.

Program kodlarının çalışma mantığı Raspberry'ye yazdığımız Tcp kodları sayesinde buradaki IMU sensöründen alınan verileri Laptop bilgisayarımızda yazdığımız Tcp server kodlarına göndererek kodların işlenip değerlerin ve grafiğin ekranda gösterilmesini sağlıyoruz.

### Gerekli Donanım Bileşenleri

1. 1 adet Raspberry pi 3
2. 1 adet ESC motor sürücü
3. 1 adet Brushless motor
4. 1 adet Mergene Platform
5. 1 adet IMU Sensörü
6. 1 adet Breadboard
7. 1 adet güç kaynağı
8. 1 adet güç adaptörü
9. 1 adet pervane

### Gerekli Yazılım Bileşenleri

1. Raspbian Jessie OS ([www.raspbian.org](http://www.raspbian.org))

### Kullanılan Bileşenlerin Özellikleri

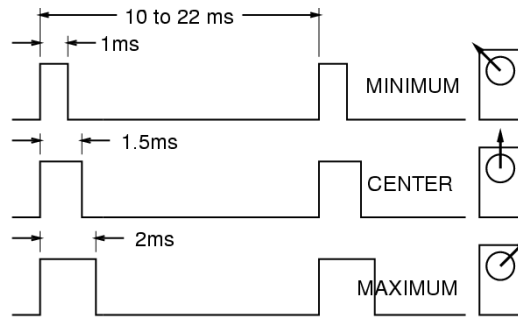
1. **Raspberry pi 3:** Raspberry Pi kredi kartı boyutunda "gerçek bir bilgisayardır". Raspberry Pi'yi tüm dünyada çocukların alıp kullanabileceği, basit programlama yapabilecekleri hatta deneylerinde kullanabileceği ucuz, küçük ve yetenekli bir bilgisayardır.

Düşük güç tüketimi: Standart kullanımda 5V 500mA civarında akım çeken Raspberry Pi 3 maksimum 5V 2,5A (12,5W) akım tüketmektedir.

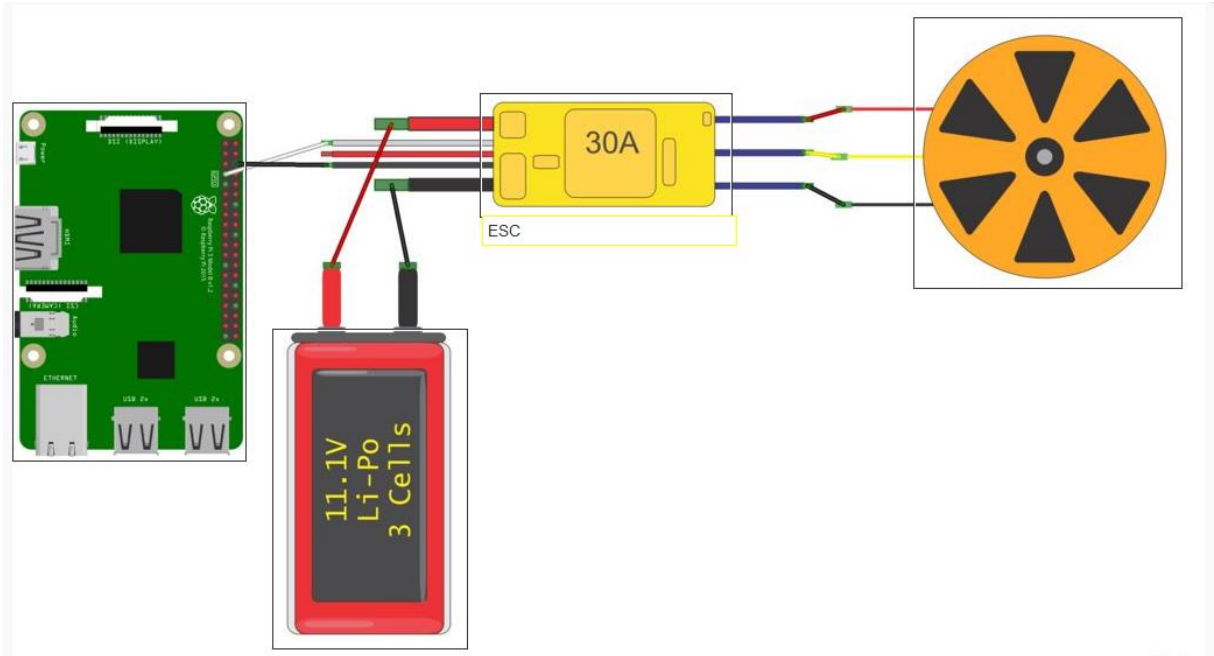


Raspberry-Pi-3.png

2. **ESC motor sürücü:** Fırçasız Motorun hızını ve dinamik bir fren gibi davranmasını mümkün kılan elektronik bir devredir.



brushless\_pwm\_darbe.png



esc\_baglantisi.png

3. **Brushless motor:** Fırçasız doğru akım motoru (brushless dc motor), komütasyon işlemini mekanik olarak değil elektronik olarak sağlayan bir motor türüdür.



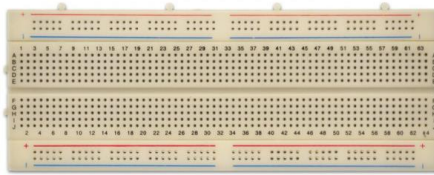
Brushless\_motor.png

4. **Mengene Platform:** Üzerine kurduğumuz sistemin platformu.
5. **IMU Sensörü:** MPU6050, 3-eksen jiroskop ve 3-eksen ivmeölçere sahip bir IMU (inertial measurement unit – ataletsel ölçü birimi) sensördür. Cisimlerin hareket ve ivmelerini ölçmek için kullanılır. İnsansız hava araçlarının en temel sensörü bu ve benzeri IMU'lardır. Aynı zamanda denge robotları, kamera stabilizasyon aletleri gibi cihazlarda da kullanılırlar.



mpu6050\_breakout.png

6. **Breadboard:** projeler yaparken en büyük yardımcılarımızdan birisi devre tahtası (breadboard) olacaktır. Devre tahtası ile projelerimizi lehim yapmadan kolayca kurabiliriz. Genel olarak içerisinde birbirine bağlı hatları barındıran devre tahtası üzerine elektronik bileşenleri yerleştirerek projelerimizi çalışır hale getirebiliriz.



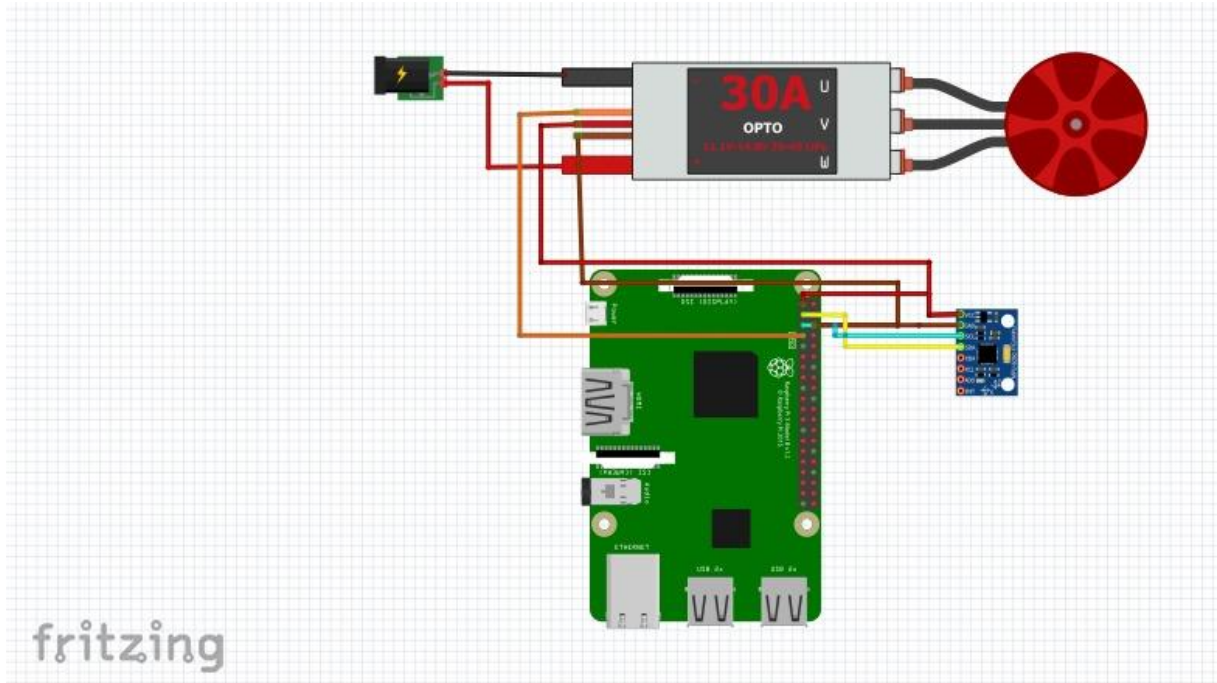
Breadboard.png

7. **Bilgisayar güç kaynağı,** prizden elektriği alıp onu bilgisayarınızın değişik parçaları için gereksinim duydukları değişik gerilimlere ayarlayan parçadır. Genellikle metal bir kasa yerleştirilmiş, içinde transformatör veya elektronik devreler bulunan, bilgisayar birimlerinin çalışmaları için gereksinim duyulan farklı gerilim değerlerinde doğru akım sağlayan Donanım donanımdır.
8. **1045L pervane:** Brushles Motor miline bağladığımız pervanedir. Sistemin havalandırmasını sağlar.

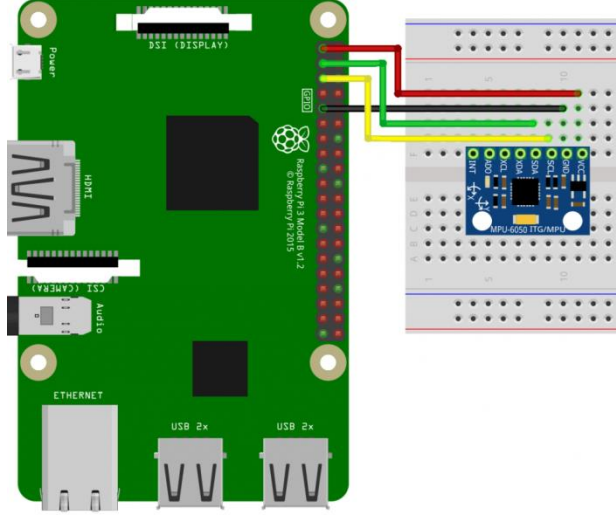


pervane.png

### Şematik Çizimi



### Raspberry Pi – MPU6050 devre şeması:



mpu6050\_IMU.png

Devremizin bağlantısını yaptıktan sonra Raspberry Pi'mizi çalıştırıyoruz. İşletim sistemi açılınca bir terminal ekranında

```
i2cdetect -y 1
```

komutunu vererek sisteme bağlı olan tüm I2C cihazların listelenmesini sağlıyoruz. Bağlantımız doğru ise aşağıdaki gibi **68** nolu adreste sensörümüzün görünmesi gereklidir:

```
pi@raspberrypi: ~  
File Edit Tabs Help  
pi@raspberrypi:~$ i2cdetect -y 1  
 0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f  
00: -- -- -- -- -- -- -- -- -- -- -- -- -- --  
10: -- -- -- -- -- -- -- -- -- -- -- -- -- --  
20: -- -- -- -- -- -- -- -- -- -- -- -- -- --  
30: -- -- -- -- -- -- -- -- -- -- -- -- -- --  
40: -- -- -- -- -- -- -- -- -- -- -- -- -- --  
50: -- -- -- -- -- -- -- -- -- -- -- -- -- --  
60: -- -- -- -- -- -- -- 68 -- -- -- -- -- --  
70: -- -- -- -- -- -- -- -- -- -- -- -- -- --  
pi@raspberrypi:~$
```

i2cdetect.png

**Not: Eğer Raspberry Pi'nin 256MB RAM belleğe sahip Model B sürümünü kullanıyorsak kodu i2cdetect -y 0 olarak değiştirmemiz gereklidir.**

Sensörümüz sistem tarafından sorunsuzca algılandıysa Python kodumuzu çalıştırmaya hazırız demektir:

```
import smbus
import math
import time

# Guc yonetim register'leri
power_mgmt_1 = 0x6b
power_mgmt_2 = 0x6c

def read_byte(adr):
    return bus.read_byte_data(address, adr)

def read_word(adr):
    high = bus.read_byte_data(address, adr)
    low = bus.read_byte_data(address, adr+1)
    val = (high << 8) + low
    return val

def read_word_2c(adr):
    val = read_word(adr)
    if (val >= 0x8000):
        return -((65535 - val) + 1)
    else:
        return val

def dist(a,b):
    return math.sqrt((a*a)+(b*b))

def get_y_rotation(x,y,z):
    radians = math.atan2(x, dist(y,z))
    return -math.degrees(radians)

def get_x_rotation(x,y,z):
    radians = math.atan2(y, dist(x,z))
    return math.degrees(radians)

bus = smbus.SMBus(1)
address = 0x68 #MPU6050 I2C adresi

#MPU6050 ilk calistiginda uyku modunda oldugundan, calistirmak icin
asagidaki komutu veriyoruz:
bus.write_byte_data(address, power_mgmt_1, 0)

while True:
    time.sleep(0.1)
    #Jiroskop register'larini oku
    gyro_xout = read_word_2c(0x43)
    gyro_yout = read_word_2c(0x45)
    gyro_zout = read_word_2c(0x47)

    print "Jiroskop X : ", gyro_xout, " olcekli: ", (gyro_xout / 131)
    print "Jiroskop Y : ", gyro_yout, " olcekli: ", (gyro_yout / 131)
    print "Jiroskop Z : ", gyro_zout, " olcekli: ", (gyro_zout / 131)
```

```
#Ivmeolcer register'larini oku
accel_xout = read_word_2c(0x3b)
accel_yout = read_word_2c(0x3d)
accel_zout = read_word_2c(0x3f)

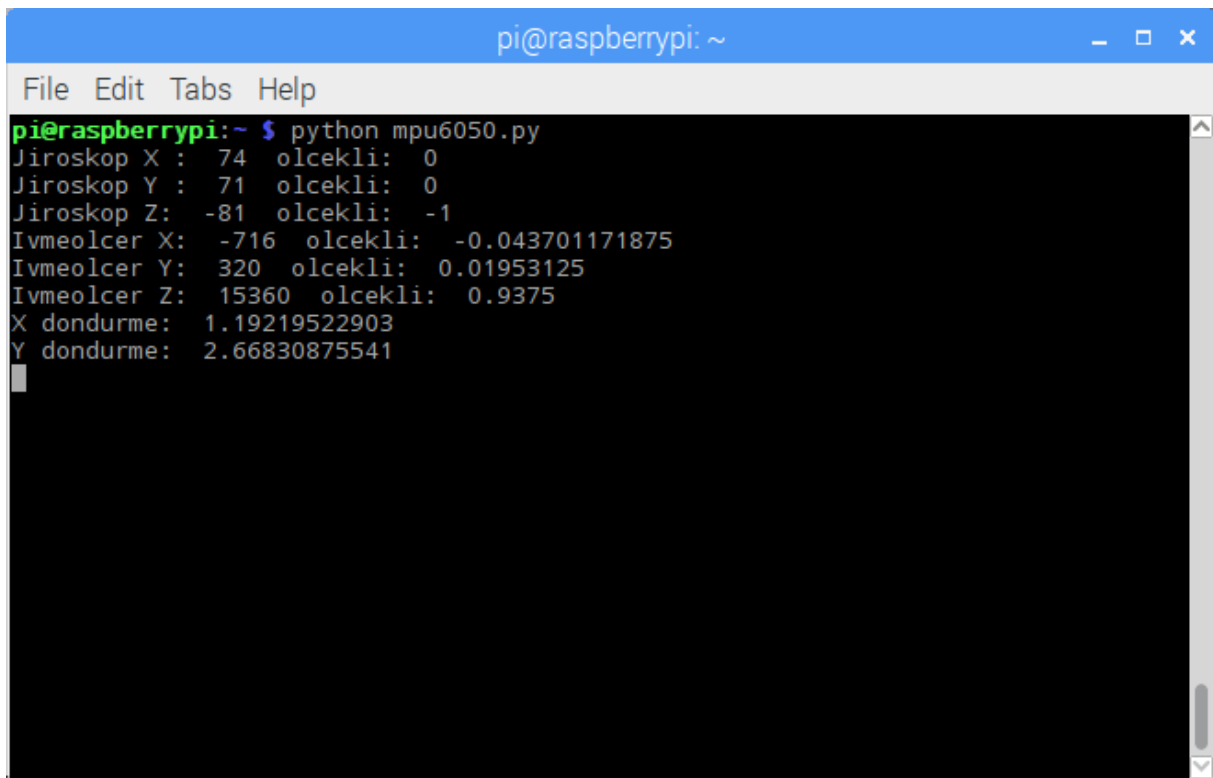
accel_xout_scaled = accel_xout / 16384.0
accel_yout_scaled = accel_yout / 16384.0
accel_zout_scaled = accel_zout / 16384.0

print "Ivmeolcer X: ", accel_xout, " olcekli: ", accel_xout_scaled
print "Ivmeolcer Y: ", accel_yout, " olcekli: ", accel_yout_scaled
print "Ivmeolcer Z: ", accel_zout, " olcekli: ", accel_zout_scaled

print "X dondurme: " , get_x_rotation(accel_xout_scaled,
accel_yout_scaled, accel_zout_scaled)
print "Y dondurme: " , get_y_rotation(accel_xout_scaled,
accel_yout_scaled, accel_zout_scaled)

time.sleep(0.5)
```

Kodumuz çalışırken bilgileri bize aşağıdaki gibi vermesi gerekli:



```
pi@raspberrypi:~ $ python mpu6050.py
Jiroskop X : 74 olcekli: 0
Jiroskop Y : 71 olcekli: 0
Jiroskop Z: -81 olcekli: -1
Ivmeolcer X: -716 olcekli: -0.043701171875
Ivmeolcer Y: 320 olcekli: 0.01953125
Ivmeolcer Z: 15360 olcekli: 0.9375
X dondurme: 1.19219522903
Y dondurme: 2.66830875541
```

mpu6050.png



## Yapım Aşamaları

### Giriş.

Oransal–integral–türev denetleyicisi (PID denetleyici) endüstriyel kontrol sistemlerinde yaygın olarak kullanılan bir kontrol döngüsü geri besleme mekanizmasıdır. Bir PID kontrolör sürekli olarak bir hata değeri hesaplar  $e(t)$  istenen bir ayar noktası ve ölçülen bir işlem değişkeni arasındaki fark ve orantılı, integral ve türev terimlerin (bazen belirtilen P, I ve d sırasıyla) dayalı bir düzeltme uygular.

Mekanizma'mızı dengede tutmak için bu mekanizmayı fırçasız motor ile kontrol ederek kullanacağız. Mekanizma üzerinde motorumuzu yerleştirerek ve MPU6050 veya MPU9250 imu modülü kullanarak açısını hesaplamaktayız.

E(t) hatası, Mekanizma'nın gerçek açısı ile istenen arasındaki fark olacaktır. Arzu edilen açı değeri kendimizin belirlediği açılarda hareket olacaktır.

### 1.Aşama

Mekanizmamız için kendi tasarladığımız platformunu kullanıyoruz. Mekanizmayı kontrol etmek için kendi kodumuzu yazdık. Fakat motoru çok hassas bir şekilde kontrol etmedikçe Mekanizma sabit gitmeyecek. Bu yüzden PID kontrolünün nasıl çalıştığını önceden öğrendik ve sonrasında kodumuzu yazdık. Her fırçasız motor, PWM sinyali için bir güce sahip olacaktır. Mekanizma'mızın açısını sürekli olarak ölçmemiz, bu değeri istenilen değerle karşılaştırmalı ve eğer varsa, hatayı düzeltmeliyiz.

Motorumuzu ayarlamak için öncelikle denge konumunu belirleyerek, kodüzerinde ayarlamalar yaparak istenilen denge seviyesinde tutarız. Bu motor sadece bir eksen temsil eder, bu durumda x eksenini olacak. Bu dengeyle yapmak istediğimiz tek şey P, I ve D sabitlerimizi bulmak. Bu sabitlerin her biri bir şekilde ya da diğer tüm PID kontrolünü etkileyecek, ve biz sistemin çalışma koşulları için en iyi noktayı belirleyerek bu sabitleri bulduk ve kodumuzun içerisine aktardık.

### 2.Aşama

ESC'yi 12V ile besleyin. ESC'den BEC 5V Raspberry besler veya isterseniz USB ile de besleyebiliriz. IMU'yu i2c bağlantı pimleri SCL ve SDA kullanarak bağladık

MPU6050 modülünü mekanizma üzerinde dengede monte ettik. Ayrıca, i2c kablolarının her birini bir GND kablosu etrafına saralım. Çünkü, bu i2c iletişiminin gürültüsünü azaltacaktır. Bunun için, raspberry'den MPU6050'ye iki GND telini ve bunların her birinin etrafındaki kabloyu, i2c'nin SCL ve SDA kablolarını monte ettik.

### 3.Aşama

IMU (inertial hareket ünitesi) modülünden bazı verileri okumak zorundayız ve bu verileri kullanarak eksenin gerçek eğim açısını hesaplıyoruz. Denge için sadece bir eksen kullanacağız, x eksenini. Bu açıyı 0° ile karşılaştırmalıyız çünkü Mekanizma yatay ekseninde dengede olacak. Fırçasız motorları kontrol etmek için, her bir ESC'ye 1000us ile 2000us arasında bir darbe ile bir PWM sinyali göndermemiz gerekir.

## Python Kodu

```
#!/usr/bin/python
import RPi.GPIO as GPIO
import smbus
import math
import time
import datetime
import math
import os
#Start the pigpiod
os.system("sudo pigpiod")
time.sleep(0.1)
import pigpio
pi = pigpio.pi()

#---Kalman Accel-Gyro---#
Acceleration_angle=[0,0]
Gyro_angle=[0,0]
Total_angle=[0,0]
rad_to_deg = 180/math.pi

#////////PID CONSTANTS////////#
kp = 0.005
ki = 0.00001
kd = 0.000
#/////////////////////////////////#
pid_p = 0.0
pid_i = 0.0
pid_d = 0.0

desired_angle = 10 #Desired angle set to 0 degree or adjustable angle
previous_error = 0

pwmLeftVal = 0
pwmRightVal = 0
pwmMax = 2000
```

```

pwmMin    = 1000
pwmLeftPin = 4
pwmRightPin = 23

# Register
power_mgmt_1 = 0x6b
power_mgmt_2 = 0x6c

def read_byte(reg):
    return bus.read_byte_data(address, reg)

def read_word(reg):
    h = bus.read_byte_data(address, reg)
    l = bus.read_byte_data(address, reg+1)
    value = (h << 8) + l
    return value

def read_word_2c(reg):
    val = read_word(reg)
    if (val >= 0x8000):
        return -((65535 - val) + 1)
    else:
        return val

def get_Angle(axis,elapsedTime):

    gyroskop_xout = read_word_2c(0x43)
    gyroskop_yout = read_word_2c(0x45)
    gyroskop_zout = read_word_2c(0x47)

    acceleration_xout = read_word_2c(0x3b)
    acceleration_yout = read_word_2c(0x3d)
    acceleration_zout = read_word_2c(0x3f)

    acceleration_xout_scaled = acceleration_xout / 16384.0
    acceleration_yout_scaled = acceleration_yout / 16384.0
    acceleration_zout_scaled = acceleration_zout / 16384.0
    #16384.0 that's the value that the datasheet gives us

    gyroskop_xout_scaled = gyroskop_xout / 131.0
    gyroskop_yout_scaled = gyroskop_yout / 131.0
    gyroskop_zout_scaled = gyroskop_zout / 131.0
    #131.0 that's the value that the datasheet gives us

```

```

#---X---#
Acceleration_angle[0] =
math.atan(acceleration_yout_scaled/math.sqrt(math.pow(acceleration_xout_scaled,2)+

math.pow(acceleration_zout_scaled,2)))*rad_to_deg
#---Y---#
Acceleration_angle[1] = math.atan(-
1*acceleration_xout_scaled/math.sqrt(math.pow(acceleration_yout_scaled,2)+

math.pow(acceleration_zout_scaled,2)))*rad_to_deg
#---X---#
Gyro_angle[0]=gyroskop_xout_scaled
#---Y---#
Gyro_angle[1]=gyroskop_yout_scaled

#Finally we can apply the final filter (Kalman Filter)
#---X axis angle---#
Total_angle[0] = 0.98*(Total_angle[0] + Gyro_angle[0]*elapsedTime) +
0.02*Acceleration_angle[0];
#---Y axis angle---#
Total_angle[1] = 0.98*(Total_angle[1] + Gyro_angle[1]*elapsedTime) +
0.02*Acceleration_angle[1];

if(axis == "X"):
    #print("X: ",Total_angle[0])
    return Total_angle[0]
elif(axis == "Y"):
    #print("Y: ",Total_angle[1])
    return Total_angle[1]

def save(angle): # Açı bilgisini bir dosyaya her iterasyonda kaydediyoruz.
    file = open("/home/pi/data.txt","w+")
    file.write(str(angle))
    file.write("\n")
    file.close()

bus = smbus.SMBus(1)
address = 0x68
bus.write_byte_data(address, power_mgmt_1, 0)

#---Serial Communication---#
h1 = pi.serial_open("/dev/serial0",3000000)

#/////Initial PWM/////#
pi.set_servo_pulsewidth(pwmLeftPin, 900)
pi.set_servo_pulsewidth(pwmRightPin,900)
time.sleep(3)
Time = datetime.datetime.now()
try:
    while True:
        timePrev = Time
        Time = datetime.datetime.now()
        elapsedTime = Time - timePrev

```

```

elapsedTime = elapsedTime.seconds + elapsedTime.microseconds*0.000001#us to
second

##### PID #####
angle = get_Angle("X",elapsedTime)
error = angle - desired_angle

#proportional value of the PID is proportional constant multiplied by error
pid_p = kp*error

#integral value of the PID is sum the previous integral value with the error multiplied
by the integral constant
pid_i = pid_i+(ki*error)

#derivative value of the PID is speed of change of error multiplied by derivative the
constant
pid_d = kd*((error - previous_error)/elapsedTime)

previous_error = error #Remember to store the previous error

#PID value is the sum of each of this 3 parts
PID = pid_p + pid_i + pid_d

#set PWM to drive BDCM (Brushless DC Motor)
pwmLeftVal = pwmLeftVal + PID
pwmRightVal = pwmRightVal - PID

if pwmLeftVal < pwmMin:
    pwmLeftVal = pwmMin

elif pwmLeftVal > pwmMax:
    pwmLeftVal = pwmMax

if pwmRightVal < pwmMin:
    pwmRightVal = pwmMin

elif pwmRightVal > pwmMax:
    pwmRightVal = pwmMax

pi.set_servo_pulsewidth(pwmLeftPin,pwmLeftVal)
pi.set_servo_pulsewidth(pwmRightPin,pwmRightVal)
pi.serial_write(h1,(str(angle)+",").encode('utf-8'))
pi.serial_write(h1,(str(pwmLeftVal)+",").encode('utf-8'))
pi.serial_write(h1,(str(desired_angle)+"\n").encode('utf-8'))
#time.sleep(0.02)
except:
    pi.stop()
    os.system("sudo pkill pigpiod")

```

- **TCP SERVER KODLARI**

### **Bu kodlar Rasperry den gelen bilgileri Ekranda Laptopta gösterir**

```
#!/usr/bin/env  
python
```

```
import time  
import collections  
import matplotlib.pyplot as plt  
import matplotlib.animation as animation  
import struct  
import pandas as pd  
import socket          # Import socket module  
import time  
s = socket.socket()      # Create a socket object  
host = "192.168.43.169" # 192.168.1.105          # Get local machine name  
port = 12345 # Port  
s.bind((host, port))     # Bind to the port  
s.listen(5)  
c, addr = s.accept()      # Establish connection with client.  
class tcpPlot:  
def __init__(self, plotLength = 100, dataNumBytes = 2):  
self.plotMaxLength = plotLength  
self.dataNumBytes = dataNumBytes  
self.rawData = bytearray(dataNumBytes)  
self.data = collections.deque([0] * plotLength, maxlen=plotLength)  
self.isRun = True  
self.isReceiving = False  
self.thread = None  
self.plotTimer = 0  
self.previousTimer = 0  
def getTCPData(self, frame, lines, lineValueText, lineLabel, timeText):  
currentTimer = time.clock()  
self.plotTimer = int((currentTimer - self.previousTimer) * 1000) # the first reading  
will be erroneous  
self.previousTimer = currentTimer  
timeText.set_text('Plot Interval = ' + str(self.plotTimer) + 'ms')  
#value1, = struct.unpack('f', self.rawData) # use 'h' for a 2 byte integer  
value = c.recv(1024)  
#print(value)  
value = value[0:5]  
print(value)  
#print(type(value))  
value = value.decode('UTF-8')  
value = (-1) * float(value)  
self.data.append(value) # we get the latest data point and append it to our array  
lines.set_data(range(self.plotMaxLength), self.data)
```

```

lineValueText.set_text('[' + lineLabel + ']' + str(value))
# self.csvData.append(self.data[-1])
def close(self):
self.isRun = False
def main():
maxPlotLength = 100
dataNumBytes = 4          # number of bytes of 1 data point
s = tcpPlot(maxPlotLength, dataNumBytes)    # initializes all required variables
# plotting starts below
pltInterval = 50          # Period at which the plot animation updates [ms]
xmin = 0
xmax = maxPlotLength
ymin = -(100) # 50
ymax = 100 # 50
fig = plt.figure()
ax = plt.axes(xlim=(xmin, xmax), ylim=(float(ymin - (ymax - ymin) / 10),
float(ymax + (ymax - ymin) / 10)))
ax.set_title('Angle Time Graphic')
ax.set_xlabel("time")
ax.set_ylabel("Angle")
lineLabel = 'Angle'
timeText = ax.text(0.50, 0.95, "", transform=ax.transAxes)
lines = ax.plot([], [], label=lineLabel)[0]
lineValueText = ax.text(0.50, 0.90, "", transform=ax.transAxes)
anim = animation.FuncAnimation(fig, s.getTCPData, fargs=(lines, lineValueText,
lineLabel, timeText), interval=pltInterval) # fargs has to be a tuple
plt.legend(loc="upper left")
plt.show()
s.close()
if __name__ == '__main__':
main()

```

- **TCP KODLARI**

**Raspberryden servere very yollamak için yazdığımız kod**

```
#!/usr/bin/python
# This is client.py
file

import socket                # Import socket module
import time
s = socket.socket()           # Create a socket object
host = "192.168.43.169"      # Get local machine name
port = 12345 # port
s.connect((host, port))
while True:
file = open("/home/pi/data.txt","r+")
mesaj = file.read()
s.send(mesaj.encode('utf-8'))
time.sleep(0.01)
s.close()
time.sleep(1)
```