**MARMARA ÜNİVERSİTESİ**

**TEKNOLOJİ FAKÜLTESİ**

**MEKATRONİK MÜHENİSLİĞİ BÖLÜMÜ**

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

**Proje1**

**PID Konum Kontrolü Deney Seti**

**SERHAT TUĞAN**

**170215026**

**ÖMER GÜNDOĞDU**

**171216821**

[**Dr. Öğr. Üyesi Hüseyin YÜCE**](http://abys.marmara.edu.tr/huseyin.yuce/)

**-2018**

**PID Konum Kontrolü Deney Seti**

**Giriş**

Bu proje uçan sistemlerin konum kontrolünün PID ile sağlanması için tasarlanmıştır. Rasperry pi , IMU sensöründen aldığı konum bilgileri doğrulusunda 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, Rasperry pi,IMU Sensörü,ESC,Güç adaptörü,Breadboard,Brushless Motor ve Mengeneli Platform kullanılmıştır.

Projede, IMU’dan alınan realtime verileri Rasperry 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.

**Gerekli Donanım Bileşenleri**

1. 1 adet Rasperry pi 3
2. 1 adet ESC motor sürücü
3. 1 adet Brushless motor
4. 1 adet Mengene 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**

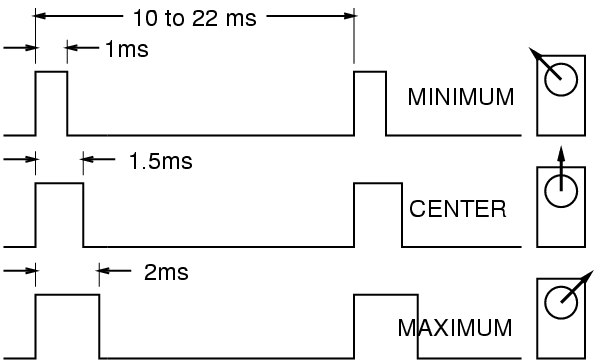
1. **Rasperry pi 3:** Raspberry Pi kredi kartı boyutunda "gerçek bir bilgisayardır". Raspberry Pi'yi tüm dünyada çocukların alıp kullanabileceği, basit programlama yapabilecekleri hatta deneylerinde kullanabileceği ucuz, küçük ve yetenekli bir bilgisayardır.

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

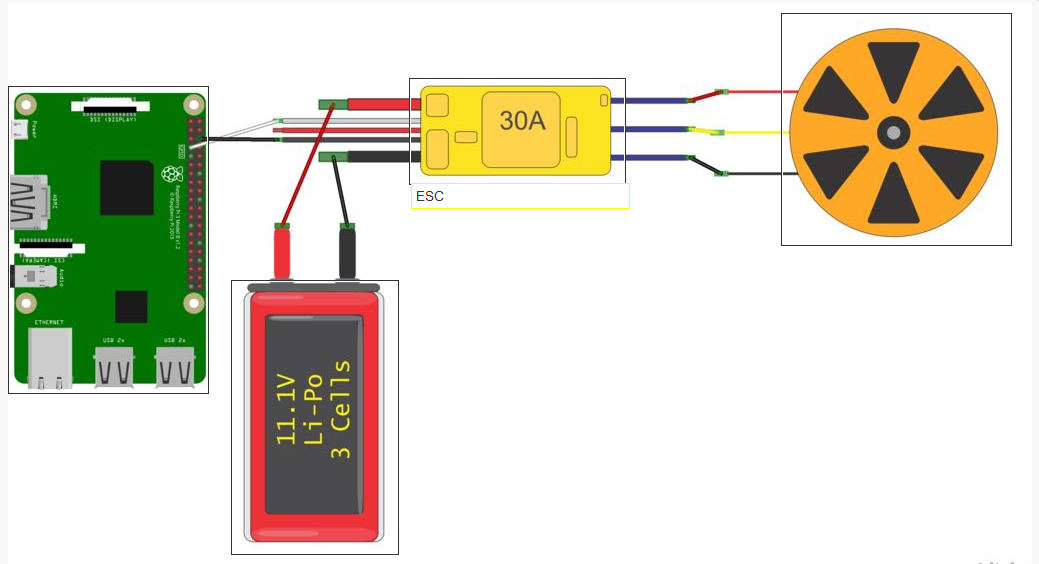


Raspberry-Pi-3.png

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



brushless\_pwm\_darbe.png



esc\_baglantisi.png

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



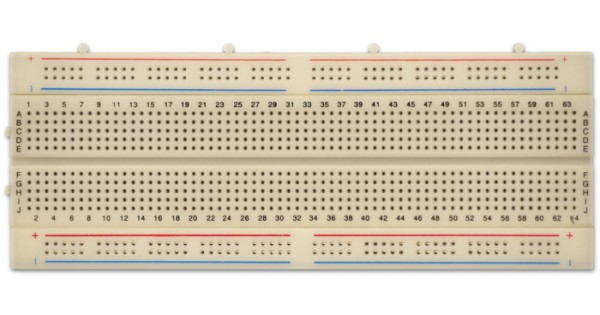
Brushless\_motor.png

1. **Mengene Platform:** Üzerine kurduğumuz sistemin platformu.
2. **IMU Sensörü:** MPU6050, 3-eksen jiroskop ve 3-eksen ivmeölçere sahip bir IMU (inertial measurement unit – ataletsel ölçü birimi) sensörüdür. 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

1. **Breadboard:** projeler yaparken en büyük yardımcılarınızdan birisi [devre](http://arduinoturkiye.com/etiket/devre/) tahtası (breadboard) olacaktır. Devre tahtası ile projelerimizi lehim yapmadan kolayca kurabiliriz. [Genel](http://arduinoturkiye.com/kategori/genel/) olarak içerisinde birbirine bağlı hatları barındıran devre tahtası üzerine elektronik bileşenleri yerleştirerek projelerimizi çalışır hale getirebiliriz.



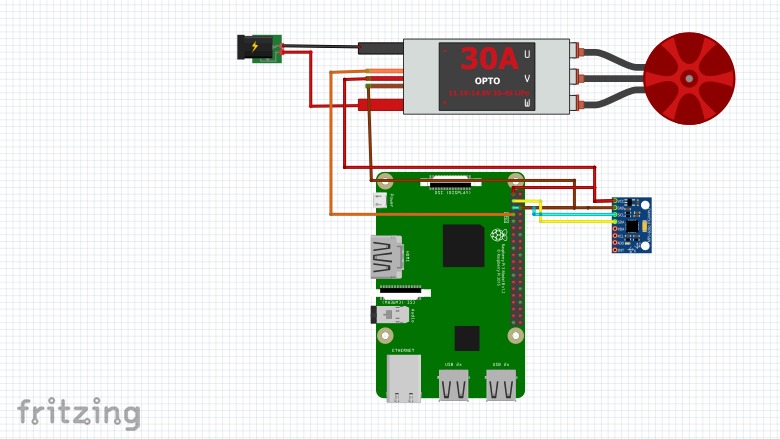
Breadboard.png

1. **Bilgisayar güç kaynağı,** prizden elektriği alıp onu bilgisayarınızın değişik parçaları için gereksinim duydukları değişik gerilimlere ayarlayan parçadır. Genellikle [metal](http://tr.wikipedia.org/wiki/Metal) bir [kasa](http://tr.wikipedia.org/wiki/Kasa) yerleştirilmiş,içinde [transformatör](http://tr.wikipedia.org/wiki/Transformat%C3%B6r) veya [elektronik](http://tr.wikipedia.org/wiki/Elektronik) [devreler](http://tr.wikipedia.org/wiki/Devre) bulunan, [bilgisayar](http://tr.wikipedia.org/wiki/Bilgisayar) birimlerinin çalışmaları için gereksinim duyulan farklı gerilim değerlerinde [doğru akım](http://tr.wikipedia.org/wiki/Do%C4%9Fru_Ak%C4%B1m) sağlayan [Donanım](http://tr.wikipedia.org/wiki/Donan%C4%B1m) donanımdır.
2. **1045L pervane:** Brushles Motor miline bağladığımız pervanedir.Sistemin havalanmasını sağlar.

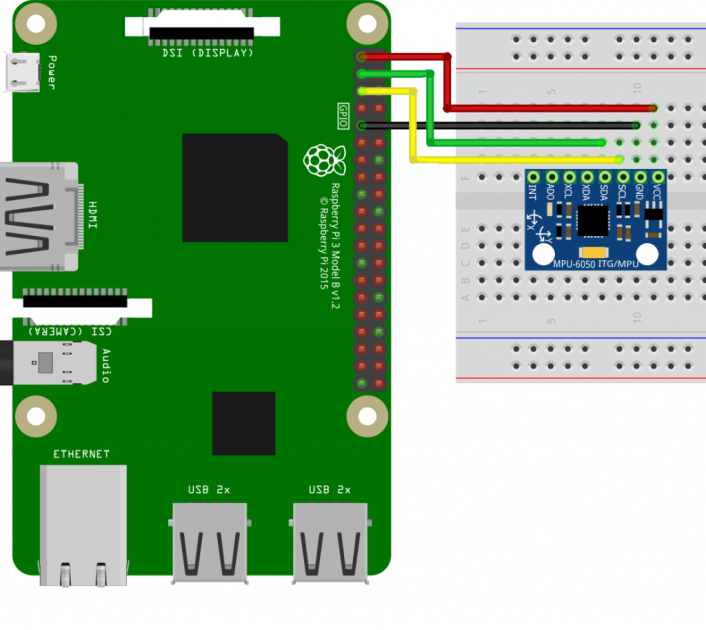


pervane.png

**Şematik Çizimi**



**Raspberry Pi – MPU6050 devre şeması:**



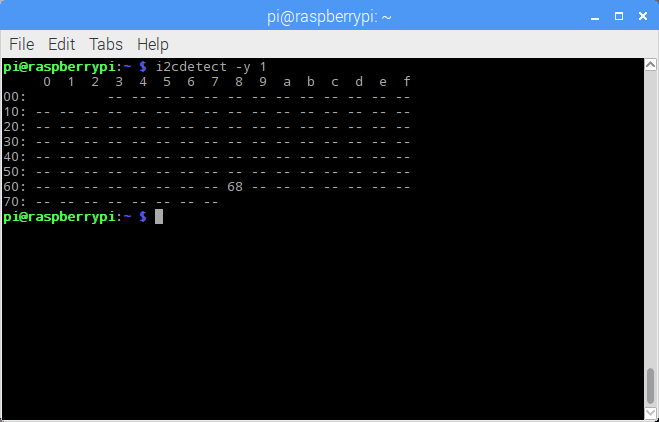
mpu6050\_IMU.png

Devremizin bağlantısını yaptıktan sonra Raspberry Pi’mizi çalaıştırıyoruz. 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:



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'lari

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

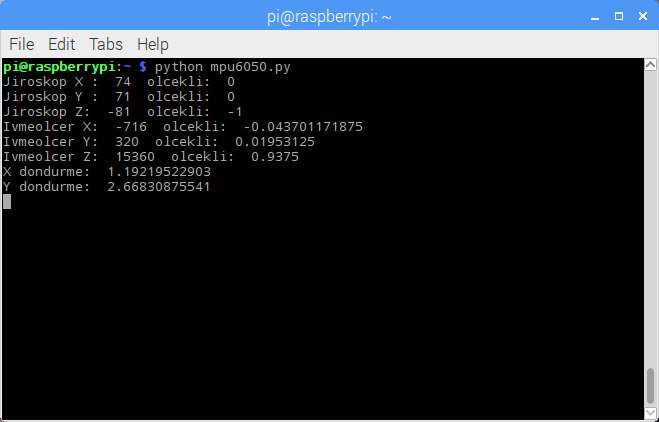
print "Ivmeolcer Z: ", accel\_zout, " olcekli: ", accel\_zout\_scaled

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:



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 ımu 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 ekseni temsil eder, bu durumda x ekseni 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 Rasberryi 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 comunnicatin 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 (inetrial hareket ünitesi) modülünden bazı verileri okumak zorundayız ve bu verileri kullanarak eksenin gerçek eğim açısını hesaplıyoruz. Denge için sadece bir eksen kullanacağız, x ekseni. Bu açıyı 0º ile karşılaştırmalıyız çünkü Mekanizma yatay eksende 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

#Finaly 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]

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")