**MECHATRONICS ENGINEERING GRADUATION PROJECT CODES**

Prepared by

ADEM CANDEMİR

BASHEER ALTAWIL

ESRA UÇAR

SARAN SAPMAZ

Some of the codes which we used them to implement our algorithmic categories and control scenarios of robot manipulator consists of both Arduino, and python scratch codes.

We expressed them as will be shown in the following sections .

## 1-Trajector generation task

This algorithm has been created to be used in trajectory generation for all manipulators and it can be change instantly by manipulation just with end effector position and orientation desired values.

```
import sys                                  ###main code for trajectory generation

import time

import numpy as np

import math as m

import matplotlib.pyplot as plt

from pylab import *

import time

import serial

import sympy

a2=230;b2=230;TH2F1=(1214-1172); TH2F2=(1228-1158);

XE2=b2*m.cos((TH2F1+TH2F2)*m.pi/180)+a2*m.cos(TH2F1*m.pi/180);

YE2=b2*m.sin((TH2F1+TH2F2)*m.pi/180)+a2*m.sin(TH2F1*m.pi/180);

print(XE2,YE2)                   ##THE END OF FORWARD

tf21=8;tf22=8  #time period definition

t=np.arange(0,8,1);# path in between points
```

```
#x2=80;y2=200 #second station of end effector

a2=230;b2=230  #links length

#algorithm to obtain via point and end effector positions

x22, y22 = sympy.symbols("x22 y22", real=True)  #to solve system equation


x21=XE2;x23=0;y21=YE2;y23=321;#position w.r.t manipulator itself

#print(x21,x23,y21,y23)

WW2=m.sqrt((y23-y21)**2+(x23-x21)**2)

#print(WW2)

THE2=30      ##bending angle (it can be change)

QQ21=(x22-x21)**2+(y22-y21)**2  ##first circle equ

QQ22=(x22-x23)**2+(y22-y23)**2  ##second circle equ

L2=(WW2/2)/(m.cos(THE2*m.pi/180))  ##midpoint of distance

CC21=L2**2  ##radius of 1st circle

CC22=L2**2  ##radius for 2nd circle

eq21 = sympy.Eq(QQ21, CC21)

eq22 = sympy.Eq(QQ22, CC22)

solution2=sympy.solve([eq21, eq22])  ##system equations solution

print(solution2)

S21=solution2[0]

S22=solution2[1]

VI2=[list(S21.values()),list(S22.values())]##values for both solution

##first solution

print(VI2)

x22=VI2[0][0]

y22=VI2[0][1]

######via point has finish###

###INVERSE K FOR INTIAL POINT

A2O =-2*a2*x21;
```

```
B2O =-2*a2*y21;

C2O = x21**2 +a2**2 + y21**2 -b2**2;

th210=2*m.atan2((-B2O -m.sqrt(B2O**2 + A2O**2 - C2O**2)),(C2O - A2O))*(180/m.pi);

th220=(m.atan2(y21-a2*m.sin(th210*m.pi/180),x21-a2*m.cos(th210*m.pi/180))*180/m.pi)-th210;

#INVERSE K FOR via POINT

AM2=-2*a2*x22;

BM2=-2*a2*y22;

CM2=x22**2 +a2**2 + y22**2 -b2**2;

th211=2*m.atan2((-BM2 -m.sqrt(BM2**2 + AM2**2 - CM2**2)),(CM2 - AM2))*(180/m.pi);

th221=(m.atan2(y22-a2*m.sin(th211*m.pi/180),x22-a2*m.cos(th211*m.pi/180))*180/m.pi)-th211;

#INVERSE K FOR FINAL POINT

AF2=-2*a2*x23;

BF2=-2*a2*y23;

CF2=x23**2 +a2**2 +y23**2-b2**2;

th212=2*m.atan2((-BF2 -m.sqrt(BF2**2 + AF2**2 - CF2**2)),(CF2 -AF2))*(180/m.pi);

th222=(m.atan2(y23-a2*m.sin(th212*m.pi/180),x23-a2*m.cos(th212*m.pi/180))*180/m.pi)-th212;

#8 by 8 matrices FOR FIRST ACTUATOR (perameters coiffients)

B21=np.array([[th210],[th211],[th211],[th212],[0],[0],[0],[0]]) #values of abgles obtained from inverse
kinematic 1.st actuator

A21=np.array([[1 ,0 ,0, 0, 0 ,0, 0, 0],[1 ,tf21, tf21*tf21, tf21*tf21*tf21, 0, 0, 0 ,0],
    [0, 0 ,0 ,0 ,1 ,0, 0, 0 ],[0 ,0 ,0 ,0, 1, tf22, tf22*tf22, tf22*tf22*tf22],
    [0 ,1 ,0 ,0, 0 ,0, 0, 0 ],[0, 0, 0, 0, 0, 1, 2*tf22, 3*tf22*tf22],
    [0 ,1 ,2*tf21+3*tf21*tf21, 0, 0, -1, 0 ,0,] ,[0, 0, 2, 6*tf21, 0 ,0 ,-2, 0] ])

A21N= np.linalg.inv(A21)

C21=np.matmul(A21N,B21)

print(C21)

#print(th10,th11,th12)

t21s1=C21[0,0]+C21[1,0]*t+C21[2,0]*t**2+C21[3,0]*t**3; #first segment equation for 1.st act

t21s2=C21[4,0]+C21[5,0]*t+C21[6,0]*t**2+C21[7,0]*t**3; #second segment equation  for 1st act
```

```
#8 by 8 matrices FOR second ACTUATOR

B22=np.array([[th220],[th221],[th221],[th222],[0],[0],[0],[0]]) #values of angles obtained from inverse kinematic 1.st actuator

#8 by 8 matrices FOR second ACTUATOR (perameters coiffients)

A22=np.array([[1 ,0 ,0, 0, 0 ,0, 0, 0],[1 ,tf21, tf21*tf21, tf21*tf21*tf21, 0, 0, 0 ,0],

   [0, 0 ,0 ,0 ,1 ,0, 0, 0 ],[0 ,0 ,0 ,0, 1, tf22, tf22*tf22, tf22*tf22*tf22],

   [0 ,1 ,0 ,0, 0 ,0, 0, 0 ],[0, 0, 0, 0, 0, 1, 2*tf22, 3*tf22*tf22],

   [0 ,1 ,2*tf21+3*tf21*tf21, 0, 0, -1, 0 ,0,] ,[0, 0, 2, 6*tf21, 0 ,0 ,-2, 0] ])

A2N=np.linalg.inv(A22)

C22=np.matmul(A2N,B22)

#print(th20,th21,th22)

t22s1=C22[0,0]+C22[1,0]*t+C22[2,0]*t**2+C22[3,0]*t**3; #first segment equation for 1.st act

t22s2=C22[4,0]+C22[5,0]*t+C22[6,0]*t**2+C22[7,0]*t**3; #second segment equation  for

tt21=[];tt22=[];tt23=[];tt24=[]  #empty lists for thetas array to be send like integers

for aq21 in t21s1*100:

   bb21=('%0.0f'%aq21)

   #print(bb1)

   tt21.append(int(bb21))

   #time.sleep(1)

tm21s1=tt21  #list to be send for first segment of first motor

#print(tm21s1)

for aq22 in t21s2*100:


   bb22=('%0.0f'%aq22)

   #print(bb22)

   tt22.append(int(bb22))

   #time.sleep(1)

tm21s2=tt22  #list to be send for second segment of first motor

#print(tm21s2)
```

```
for aq23 in t22s1*100:
    bb23=('%0.0f'%aq23)
    #print(bb23)
    tt23.append(int(bb23))
    #time.sleep(1)
tm22s1=tt23  #list to be send for first segment of second motor
#print(tm22s1)
for aq24 in t22s2*100:
    bb24=('%0.0f'%aq24)
    ##print(bb4)
    tt24.append(int(bb24))
    #time.sleep(1)
tm22s2=tt24  #list to be send for second segment of second motor
#print(tm22s2)
print(tm21s1)
print(tm22s1)
print(tm21s2)
print(tm22s2)
```

## 2-Main code to communicate between Raspberry Pi and Atmega microcontroller

In this code we design a standard algorithm that will be used in our communication between master and other slaves

### 2-a-Master code(Python)

```
 import sys
import smbus
import time
import numpy as np
```

```
import math as m

import matplotlib.pyplot as plt

from pylab import *

import time

import serial

import sympy

import RPi.GPIO as GPIO

from smbus2 import SMBus  #communication protocol that used in i2c

bus = smbus.SMBus(1)

address = 0x08 #communication address

def writeNumber(a,b,c,d,e,f,g,h,i,j,k,l,m,n,o,p,aa,bb,cc,dd,ii,jj,kk,ll,mm,nn,oo,pp,rr,ss,tt,uu):

   bus.write_i2c_block_data(address, a,[
b,c,d,e,f,g,h,i,j,k,l,m,n,o,p,aa,bb,cc,dd,ii,jj,kk,ll,mm,nn,oo,pp,rr,ss,tt,uu]) ##the total ammount of data
that will be send to atmega

   return -1

def requestreading(): ##reading function definition

   block=bus.read_i2c_block_data(address,0,3) ##to read potontiometer values that will be taken from
arduino chip


   return block


p=1

while True:

   try:

      b=requestreading() ##data requesting from arduino

      p1=int(b[0]*4.0118) #to convet analog values of potontiometer 1 to its original value 1023

      tetha11=int(p1*(5/1023)*720) ##convert analog value to angle value

      p2=int(b[2]*4.0118)  #to convet analog values of potontiometer 1 to its original value 1023

      tetha12=int(p2*(5/1023)*720)   ##convert analog value to angle value

      print("p1 %s  p2 %s :"%(tetha11,tetha12)) ##to print those values into serial monitor
```

```
##foroward kinamtic  calculation
a2=230;b2=230;TH2F1=(tetha11-1172); TH2F2=(tetha12-1158);
XE2=b2*m.cos((TH2F1+TH2F2)*m.pi/180)+a2*m.cos(TH2F1*m.pi/180);
YE2=b2*m.sin((TH2F1+TH2F2)*m.pi/180)+a2*m.sin(TH2F1*m.pi/180);
#print(XE2,YE2)
                    ##starting point of trajectory generation
tf21=8;tf22=8  #time period definition
t=np.arange(0,8,1);   ##time sectioning slices definition for quadratic equation
#x2=80;y2=200 #second station of end effector
a2=230;b2=230  #links length
#algorithm to obtain via point and end effector positions
x22, y22 = sympy.symbols("x22 y22", real=True)  #to solve system equation
x21=XE2;x23=0;y21=YE2;y23=321;#position w.r.t manipulator itself
WW2=m.sqrt((y23-y21)**2+(x23-x21)**2)
THE2=30      ##bending angle for via point inclination (it can be change)
QQ21=(x22-x21)**2+(y22-y21)**2  ##first circle equ
QQ22=(x22-x23)**2+(y22-y23)**2  ##second circle equ
L2=(WW2/2)/(m.cos(THE2*m.pi/180))  ##midpoint of distance
CC21=L2**2  ##radius of 1st circle
CC22=L2**2  ##radius for 2nd circle
eq21 = sympy.Eq(QQ21, CC21)  ##first eqation symboling
eq22 = sympy.Eq(QQ22, CC22)    ##first eqation symboling
solution2=sympy.solve([eq21, eq22])  ##system equations solution
#print(solution2)
S21=solution2[0]
S22=solution2[1]
VI2=[list(S21.values()),list(S22.values())]##values for both solution
##first solution
x22=VI2[0][0]
```

```
y22=VI2[0][1]

######via point has finish###

###INVERSE K FOR INTIAL POINT

A2O =-2*a2*x21;

B2O =-2*a2*y21;

C2O = x21**2 +a2**2 + y21**2 -b2**2;

th210=2*m.atan2((-B2O -m.sqrt(B2O**2 + A2O**2 - C2O**2)),(C2O - A2O))*(180/m.pi);  ##first motor joint angle

th220=(m.atan2(y21-a2*m.sin(th210*m.pi/180),x21-a2*m.cos(th210*m.pi/180))*180/m.pi)-th210; ##second motor joint angle

#INVERSE K FOR via POINT

AM2=-2*a2*x22;

BM2=-2*a2*y22;

CM2=x22**2 +a2**2 + y22**2 -b2**2;

th211=2*m.atan2((-BM2 -m.sqrt(BM2**2 + AM2**2 - CM2**2)),(CM2 - AM2))*(180/m.pi);

th221=(m.atan2(y22-a2*m.sin(th211*m.pi/180),x22-a2*m.cos(th211*m.pi/180))*180/m.pi)-th211;

#INVERSE K FOR FINAL POINT

AF2=-2*a2*x23;

BF2=-2*a2*y23;

CF2=x23**2 +a2**2 +y23**2-b2**2;

th212=2*m.atan2((-BF2 -m.sqrt(BF2**2 + AF2**2 - CF2**2)),(CF2 -AF2))*(180/m.pi);

th222=(m.atan2(y23-a2*m.sin(th212*m.pi/180),x23-a2*m.cos(th212*m.pi/180))*180/m.pi)-th212;

#8 by 8 matrices FOR FIRST ACTUATOR that contains coifficients of perameters

B21=np.array([[th210],[th211],[th211],[th212],[0],[0],[0],[0]]) #values of abgles obtained from inverse kinematic 1.st actuator

A21=np.array([[1 ,0 ,0, 0, 0 ,0, 0, 0],[1 ,tf21, tf21*tf21, tf21*tf21*tf21, 0, 0, 0 ,0],

    [0, 0 ,0 ,0 ,1 ,0, 0, 0 ],[0 ,0 ,0 ,0, 1, tf22, tf22*tf22, tf22*tf22*tf22],

    [0 ,1 ,0 ,0, 0 ,0, 0, 0 ],[0, 0, 0, 0, 0, 1, 2*tf22, 3*tf22*tf22],

    [0 ,1 ,2*tf21+3*tf21*tf21, 0, 0, -1, 0 ,0,] ,[0, 0, 2, 6*tf21, 0 ,0 ,-2, 0] ])

A21N= np.linalg.inv(A21)
```

```
C21=np.matmul(A21N,B21)
#print(th10,th11,th12)
t21s1=C21[0,0]+C21[1,0]*t+C21[2,0]*t**2+C21[3,0]*t**3; #first segment equation for 1.st act
t21s2=C21[4,0]+C21[5,0]*t+C21[6,0]*t**2+C21[7,0]*t**3; #second segment equation  for 1st act
#8 by 8 matrices FOR second ACTUATOR
B22=np.array([[th220],[th221],[th221],[th222],[0],[0],[0],[0]]) #values of angles obtained from
inverse kinematic 1.st actuator
A22=np.array([[1 ,0 ,0, 0, 0 ,0, 0, 0],[1 ,tf21, tf21*tf21, tf21*tf21*tf21, 0, 0, 0 ,0],
  [0, 0 ,0 ,0 ,1 ,0, 0, 0 ],[0 ,0 ,0 ,0, 1, tf22, tf22*tf22, tf22*tf22*tf22],
  [0 ,1 ,0 ,0, 0 ,0, 0, 0 ],[0, 0, 0, 0, 0, 1, 2*tf22, 3*tf22*tf22],
  [0 ,1 ,2*tf21+3*tf21*tf21, 0, 0, -1, 0 ,0,] ,[0, 0, 2, 6*tf21, 0 ,0 ,-2, 0] ])
A2N=np.linalg.inv(A22)
C22=np.matmul(A2N,B22)
#print(th20,th21,th22)
t22s1=C22[0,0]+C22[1,0]*t+C22[2,0]*t**2+C22[3,0]*t**3; #first segment equation for 1.st act
t22s2=C22[4,0]+C22[5,0]*t+C22[6,0]*t**2+C22[7,0]*t**3; #second segment equation  for
tt21=[];tt22=[];tt23=[];tt24=[]  #empty lists for thetas erray
for aq21 in t21s1*100:
  bb21=('%0.0f'%aq21)
  #print(bb1)
  tt21.append(int(bb21))
  #tt1.append(int(bb21)/100)
  #time.sleep(1)
tm21s1=tt21  #list to be send for first segment of first motor
#print(tm21s1)
for aq22 in t21s2*100:


  bb22=('%0.0f'%aq22)
  #print(bb22)
```

```
   tt22.append(int(bb22))
   #time.sleep(1)
tm21s2=tt22  #list to be send for second segment of first motor
#print(tm21s2)
for aq23 in t22s1*100:
   bb23=('%0.0f'%aq23)
   #print(bb23)
   tt23.append(int(bb23))
   #time.sleep(1)
tm22s1=tt23  #list to be send for first segment of second motor
#print(tm22s1)
for aq24 in t22s2*100:
   bb24=('%0.0f'%aq24)
   ##print(bb4)
   tt24.append(int(bb24))
   #time.sleep(1)
tm22s2=tt24  #list to be send for second segment of second motor
#print(tm22s2)
```

##data that will be send toward atmega

```
TK=[tm21s1[0]//255,tm21s1[1]//255,tm21s1[2]//255,tm21s1[3]//255,tm21s1[4]//255,tm21s1[5]//255,tm21s1[6]//255,tm21s1[7]//255,tm22s1[0]//255,tm22s1[1]//255,tm22s1[2]//255,tm22s1[3]//255,tm22s1[4]//255,tm22s1[5]//255,tm22s1[6]//255,tm22s1[7]//255]
TM=[tm21s1[0]%255,tm21s1[1]%255,tm21s1[2]%255,tm21s1[3]%255,tm21s1[4]%255,tm21s1[5]%255,tm21s1[6]%255,tm21s1[7]%255,tm22s1[0]%255,tm22s1[1]%255,tm22s1[2]%255,tm22s1[3]%255,tm22s1[4]%255,tm22s1[5]%255,tm22s1[6]%255,tm22s1[7]%255]
TK2=[tm21s2[0]//255,tm21s2[1]//255,tm21s2[2]//255,tm21s2[3]//255,tm21s2[4]//255,tm21s2[5]//255,tm21s2[6]//255,tm21s2[7]//255,tm22s2[0]//255,tm22s2[1]//255,tm22s2[2]//255,tm22s2[3]//255,tm22s2[4]//255,tm22s2[5]//255,tm22s2[6]//255,tm22s2[7]//255]
TM2=[tm21s2[0]%255,tm21s2[1]%255,tm21s2[2]%255,tm21s2[3]%255,tm21s2[4]%255,tm21s2[5]%255,tm21s2[6]%255,tm21s2[7]%255,tm22s2[0]%255,tm22s2[1]%255,tm22s2[2]%255,tm22s2[3]%255,tm22s2[4]%255,tm22s2[5]%255,tm22s2[6]%255,tm22s2[7]%255]
writeNumber(TK[0],TK2[0],TK[1],TK2[1],TK[2],TK2[2],TK[3],TK2[3],TK[4],TK2[4],TK[5],TK2[5],TK[6],TK2[6]
```

```
,TK[7],TK2[7],TK[8],TK2[8],TK[9],TK2[9],TK[10],TK2[10],TK[11],TK2[11],TK[12],TK2[12],TK[13],TK2[13],TK[
14],TK2[14],TK[15],TK2[15])
writeNumber(TM[0],TM2[0],TM[1],TM2[1],TM[2],TM2[2],TM[3],TM2[3],TM[4],TM2[4],TM[5],TM2[5],T
M[6],TM2[6],TM[7],TM2[7],TM[8],TM2[8],TM[9],TM2[9],TM[10],TM2[10],TM[11],TM2[11],TM[12],TM2[
12],TM[13],TM2[13],TM[14],TM2[14],TM[15],TM2[15])

    print(tm21s1)

    print(tm22s1)

    print(tm21s2)

    print(tm22s2)

    time.sleep(0.5)             #delay one second

    if p>0:

      break

  except KeyboardInterrupt:

    quit()
```

## 2-b-Slave code (Arduino Atmega)

This code has been created as slave code that will be impeded into Arduino microcontrollers so that we can receive and send data in order to implement the goal which we aim to do .

```
//HABERLESME TANIMLAMALARI

#include <Wire.h>

#define SLAVE_ADDRESS 0x08

int data [32];

int data11 [32];

int datanew[16];

int datanew11[16];

int x = 0;

int y = 0;

volatile int16_t data1[2];

volatile int16_t datanew1[2];

             //PATHDEN GELEN TANIMLAMALAR

//lİNK lENGHT CONSTANT
```

```
float a =230.0;
float b = 230.0;
//İnital positions of the our robot arms end effector CONSTANT
float A1x, A1y;
//forward kinematics Angle of manipulator1
float tetha12 ; //pottan alınacak
float tetha11 ;
//global point that will be taken from user
float Ax=0;
float Ay=0;
//end effector position defining
float Axx;
float Ayy;
//inital distance(offsets)
float x1 = 0;      //IT WİLL CHANGE FOR EACH MANİPULATOR THESE ONE FOR 1.ST MANİPULATIOR
float y1 =-325.0 ; //IT WİLL CHANGE FOR EACH MANİPULATOR THESE ONE FOR 1.ST MANİPULATIOR
//inverse kinematics angle of link1
float th12, th11;
//inverse kinematics angle of link1 to origin
float tht12, tht11;
//Saranın tanımlamaları
float Kp =10.0;
float Kp2=0.8;
int i = 0, sum = 0;
float angle, angle2, voltage, voltage2, pot1, pot2, reftheta, reftheta2, error, error2, pwm, pwm2;
 int p=1;
void setup() {
Serial.begin(9600);
Wire.begin(SLAVE_ADDRESS);
```

```
Wire.onReceive(receiveData);
Wire.onRequest(sendAnalogReading);

}

void loop() {

   for( int l = 0; l<2; l++)
  {
    data1[l] = analogRead( A4 + l);
    delay(50);
    datanew1[l]=map(data1[l],0,1023,0,255);
    //Serial.println(datanew1[l]);
  }
  /*for( int p = 0; p<16; p++)
  {
    Serial.print(datanew[p]);
    Serial.print("\t");
    Serial.println(datanew11[p]);

    delay(50);
  }*/
for(int m=0;m<8;m++)
{
  pot1 = analogRead( A4);
  pot2=analogRead( A5);
  voltage = pot1 * (5.0 / 1023);
  tetha11 = voltage * 720.0 ; //initial angle of link1
  voltage2 = pot2 * (5.0 / 1023);
```

```
  tetha12 = voltage2 * 720.0 ; //initial angle link2
 //Serial.println(tetha11);
/*Serial.print(tetha11);
Serial.print("\t");
Serial.println(tetha12);*/
   th11=datanew[m];
   th12=datanew[m+8];
   delay(100);
   //SARANIN KODU BURDA OLACAK
  error = (tetha11)-(th11+1172.0);   //675 FOR FİRST MANİPULATOR FİRST MOTOR REFERENCE ANGLE.IT WİLL CAHANGE FOR EACH MANİPULATOR
  error2 = (tetha12)-(th12+1158.0);  //418 FOR FİRST MANİPULATOR SECOND MOTOR REFERENCE ANGLE.IT WİLL CAHANGE FOR EACH MANİPULATOR
  /*Serial.print(error);
  Serial.print("\t");
  Serial.println(error2);*/
  if (error > 0)
  {
    pwm = (Kp) * error;
    if (pwm > 255)
    {
      pwm = 255;
    }
    analogWrite(8, pwm); //ccw
    analogWrite(9, 0);


  }
  if (error < 0)
  {
```

```
  pwm = -(Kp * error); // pwm can not be negative
  if (pwm > 255)
  {
    pwm = 255;
  }
  analogWrite(9, pwm); //cc
  analogWrite(8, 0);

}

if (error2 > 0)
{
  pwm2 = (Kp2) * error2;
  if (pwm2 > 255)
  {
    pwm2 = 255;
  }
  analogWrite(5, pwm2); //ccw
  analogWrite(7, 0);

}
if (error2 < 0)
{
  pwm2 = -(Kp2 * error2); // pwm can not be negative
  if (pwm2 > 255)
  {
    pwm2 = 255;
  }
  analogWrite(7, pwm2); //cc
```

```
    analogWrite(5, 0);


  }
  Serial.print(pwm);
  Serial.print("\t");
  Serial.println(pwm2);
 }
 Serial.println("-----------------------------------------------------------------------------------------");
 delay(1000);
  for(int n=0;n<8;n++)
  {
 pot1 = analogRead(A4);
  pot2 = analogRead(A5);
  voltage = pot1 * (5.0 / 1023);
  tetha11 = voltage * 720.0 ; //initial angle of link1
  voltage2 = pot2 * (5.0 / 1023);
  tetha12 = voltage2 * 720.0 ; //initial angle link2
   /*Serial.print(tetha11);
    Serial.print("\t");
    Serial.print(tetha12);
    Serial.println();*/
 th11=datanew11[n];
 th12=datanew11[n+8];
 delay(100);
  /*Serial.print(th12);
  Serial.print("\t");
  Serial.println(th11);*/
  //SARANIN KODU BURDA OLACAK
```

```
}

  error = (tetha11)-(th11+1172.0);   //675 FOR FİRST MANİPULATOR FİRST MOTOR REFERENCE ANGLE.IT WİLL CAHANGE FOR EACH MANİPULATOR

  error2 = (tetha12)-(th12+1158.0);  //418 FOR FİRST MANİPULATOR SECOND MOTOR REFERENCE ANGLE.IT WİLL CAHANGE FOR EACH MANİPULATOR

 /* Serial.print(error);

  Serial.print("\t");

  Serial.println(error2);*/

  if (error > 0)

  {

    pwm = (Kp) * error;

    if (pwm > 255)

    {

      pwm = 255;

    }

    analogWrite(8, pwm); //ccw

    analogWrite(9, 0);


  }

  if (error < 0)

  {

    pwm = -(Kp * error); // pwm can not be negative

    if (pwm > 255)

    {

      pwm = 255;

    }

    analogWrite(9, pwm); //cc

    analogWrite(8, 0);


  }
```

```
  if (error2 > 0)
  {
    pwm2 = (Kp2) * error2;
    if (pwm2 > 255)
    {
      pwm2 = 255;
    }
    analogWrite(5, pwm2); //ccw
    analogWrite(7, 0);

  }
  if (error2 < 0)
  {
    pwm2 = -(Kp2 * error2); // pwm can not be negative
    if (pwm2 > 255)
    {
      pwm2 = 255;
    }
    analogWrite(7, pwm2); //cc
    analogWrite(5, 0);

  }
   Serial.print(pwm);
  Serial.print("\t");
  Serial.println(pwm2);
  }
}
void receiveData(int byteCount) {
```

```
  while(Wire.available()) {              //Wire.available() returns the number of bytes available for retrieval
with Wire.read(). Or it returns TRUE for values >0.

      data[x]=Wire.read();

      data11[y]=Wire.read();

        x++;

      y++;


 }

    for(int i=0;i<16;i++)

    {

    //Serial.println((data[i])*(255)+data[i+16]);

    datanew[i]=((data[i])*(255)+data[i+16])/100;

    datanew11[i]=((data11[i])*(255)+data11[i+16])/100;

    }

    for(int k=0;k<16;k++)

    {

     if(datanew[k]<0)

     {

       datanew[k]=datanew[k]+256;

     }

      if(datanew11[k]<0)

     {

       datanew11[k]=datanew11[k]+256;

     }

    }

  }

void sendAnalogReading(){

 Wire.write( (uint8_t *) datanew1, sizeof( datanew1));

}
```

## 3-Pure rotation task (just to rotate platform with zero translation)

This code is specified for attaching and detaching task so that by doing pure rotation we can configure all end effector operation with respect to rotation needs .

```
import numpy as np

import sympy

import math as m

import matplotlib.pyplot as plt

from pylab import *

import time

import serial

import Rpi.GPIO as GPIO

import smbus

bus = smbus.SMBus(1)

address = 0x04 #ADRESS CAN BE CHANGE

GPIO.setmode(GPIO.BCM)

GPIO.setwarnings(False)

sevo1=17

servo2=18

servo3=19

                        ##HEXAGONAL CORNERS CORDINATIONS FINDING START

GPIO.setup(servo1, GPIO.OUT) # setting servo pin for first manipulator(gripper1)

GPIO.setup(servo2, GPIO.OUT) # setting servo pin for second manipulator(gripper2)

GPIO.setup(servo3, GPIO.OUT) # setting servo pin for third manipulator(gripper3)

def setServoAngle(servo, angle):


     pwm.start(8)

     dutyCycle = angle / 18. + 2.

     pwm.ChangeDutyCycle(dutyCycle)
```

```
            sleep(0.3)
            pwm.stop()
def writeNumber(a,b):
    bus.write_i2c_block_data(address, a, [b])
    return -1
                    ###HEXAGONAL CODE START HERE##
##CENTER AND FIRST CIRCLE
xc1, yc1 = sympy.symbols("xc1  yc1", real=True)
xx1=140;yy1=80;xx2=80;yy2=80
C1E=(xc1-xx1)**2+(yc1-yy1)**2
C2E=(xc1-xx2)**2+(yc1-yy2)**2
RR=60**2
E1Q1=sympy.Eq(C1E,RR)
E1Q2=sympy.Eq(C2E,RR)
SOLUTION1=sympy.solve([E1Q1,E1Q2])
print(SOLUTION1)
SS1=SOLUTION1[0]
SS2=SOLUTION1[1]
VV1I=[list(SS1.values()),list(SS2.values())]##values for both solution
xc10=float(VV1I[0][0])  ##X-FIRST CORDINATE
yc10=float(VV1I[0][1])  ##Y-FIRST CORDINATE
xc20=float(VV1I[1][0])  ##X-SECOND CORDINATE
yc20=float(VV1I[1][1])  ##Y-SECOND CORDINATE
print(xc10,yc10,xc20,yc20)
##CENTER AND SECOND CIRCLE
xc2, yc2 = sympy.symbols("xc2  yc2", real=True)
xx12=140;yy12=80;xx22=200;yy22=80
C1E2=(xc2-xx12)**2+(yc2-yy12)**2
C2E2=(xc2-xx22)**2+(yc2-yy22)**2
```

```
RR=60**2

E2Q1=sympy.Eq(C1E2,RR)

E2Q2=sympy.Eq(C2E2,RR)

SOLUTION2=sympy.solve([E2Q1,E2Q2])

print(SOLUTION2)

S2S1=SOLUTION2[0]

S2S2=SOLUTION2[1]

VV2I=[list(S2S1.values()),list(S2S2.values())]##values for both solution

xc11=float(VV2I[0][0])  ##X-THIRD CORDINATE

yc11=float(VV2I[0][1])  ##Y-THIRD CORDINATE

xc21=float(VV2I[1][0])  ##X-FOURTH CORDINATE

yc21=float(VV2I[1][1])  ##Y-FOURTH CORDINATE

print(xc11,yc11,xc21,yc21)

xx1c=xx1-60

yy1c=yy1

xx2c=xx1+60

yy2c=yy1

print(xx1c,yy1c)

print(xx2c,yy2c)

                    ###HEXAGONAL CODE FINISH HERE##

                    ###FORWARD KINEMATIC CODE START HERE##

##forward kinametic for FIRST MANIPULATOR

a1=230;b1=230;XF1=50;YF1=40;TH1F1=20;TH1F2=330

XE1=XF1+b1*m.cos((TH1F1+TH1F2)*m.pi/180)+a1*m.cos(TH1F1*m.pi/180); #tethas coming from Pic

YE1=YF1+b1*m.sin((TH1F1+TH1F2)*m.pi/180)+a1*m.sin(TH1F1*m.pi/180);

##forward kinametic for  SECOND MANIPULATOR

a2=230;b2=230;XF2=10;YF2=10;TH2F1=20;TH2F2=330

XE2=XF2+b2*m.cos((TH2F1+TH2F2)*m.pi/180)+a2*m.cos(TH2F1*m.pi/180);

YE2=YF2+b2*m.sin((TH2F1+TH2F2)*m.pi/180)+a2*m.sin(TH2F1*m.pi/180);
```

```
##forward kinametic for  THIRD MANIPULATOR
a3=230;b3=230;XF3=-50;YF3=100;TH3F1=20;TH3F2=330
XE3=XF3+b3*m.cos((TH3F1+TH3F2)*m.pi/180)+a3*m.cos(TH3F1*m.pi/180);
YE3=YF3+b3*m.sin((TH3F1+TH3F2)*m.pi/180)+a3*m.sin(TH3F1*m.pi/180);
print(XE3,YE3)
                    ##THE END OF FORWARD
                 ###ROTATİON TASK CODE START HERE##
for x in range(6):
#INVERSE FOR FINAL POINT OF FİRST MANİPULATOR
   aa=230;bb=230  #links length
   AFM =-2*aa*xc11;
   BFM =-2*aa*yc11;
   CFM= xc11**2 +aa**2 + yc11**2 -bb**2;
   th10h=2*m.atan2((-BFM -m.sqrt(BFM**2 + AFM**2 - CFM**2)),(CFM -AFM))*(180/m.pi);
   th20h=(m.atan2(yc11-aa*m.sin(th10h*m.pi/180),xc11-aa*m.cos(th10h*m.pi/180))*180/m.pi)-th10h;
   #th10h ve th20h Pic'e yollanacak raspberryden !!!!
   while True:
      try:
         writeNumber(th10h,th20h)
         time.sleep(1)               #delay one second

      except KeyboardInterrupt:
         quit()


   #INVERSE kinematic FOR FINAL POINT OF SECOND MANİPULATOR
   aa=230;bb=230  #links length
   ASM =-2*aa*xc21;
   BSM =-2*aa*yc21;
   CSM= xc21**2 +aa**2 + yc21**2 -bb**2;
```

```
th11h=2*m.atan2((-BSM -m.sqrt(BSM**2 + ASM**2 - CSM**2)),(CSM -ASM))*(180/m.pi);
th21h=(m.atan2(yc11-aa*m.sin(th11h*m.pi/180),xc21-aa*m.cos(th11h*m.pi/180))*180/m.pi)-th11h;
#th11h ve th21h Pic'e yollanacak raspberryden
while True:
  try:
    writeNumber(th11h,th21h)
    time.sleep(1)          #delay one second

  except KeyboardInterrupt:
    quit()

#INVERSE FOR FINAL POINT OF THİRD MANİPULATOR
aa=230;bb=230  #links length
ATM =-2*aa*xc20;
BTM =-2*aa*yc20;
CTM= xc20**2 +aa**2 + yc20**2 -bb**2;
th12h=2*m.atan2((-BTM -m.sqrt(BTM**2 + ATM**2 - CTM**2)),(CTM -ATM))*(180/m.pi);
th22h=(m.atan2(yc20-aa*m.sin(th12h*m.pi/180),xc20-aa*m.cos(th12h*m.pi/180))*180/m.pi)-th12h;
#th12h ve th22h Pic'e yollanacak raspberryden !!!!
while True:
  try:
    writeNumber(th12h,th22h)
    time.sleep(1)          #delay one second

  except KeyboardInterrupt:
    quit()
if __name__ == '__main__':
  setServoAngle(servo3, 180) #180 degree suppose gripper is closing at that angle
```

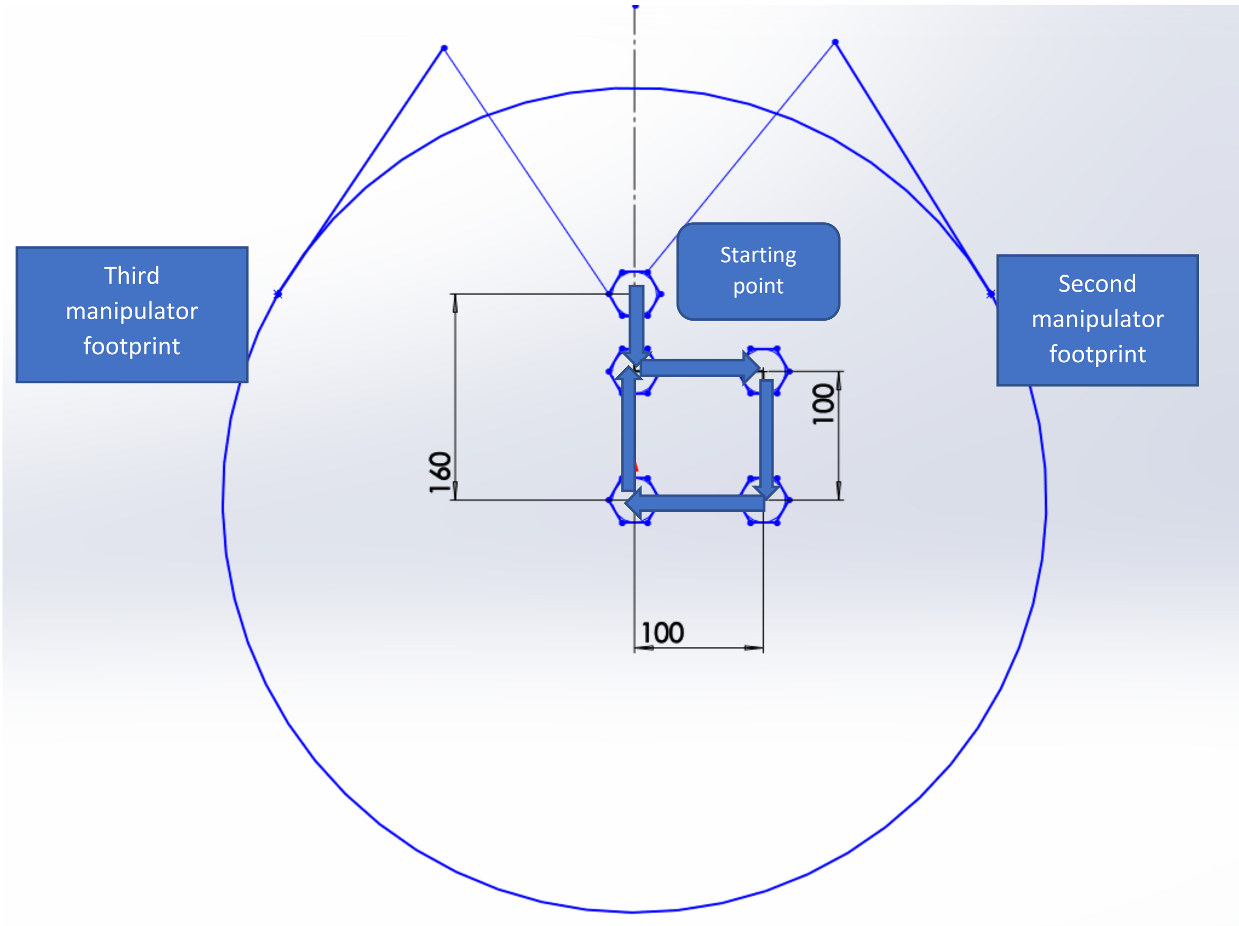
```
GPIO.cleanup()


#INVERSE FOR FINAL POINT OF FİRST MANİPULATOR GOİNG BACK
if __name__ == '__main__':
   setServoAngle(servo1, 0) #0 degree suppose gripper is opening at that angle
GPIO.cleanup()
aa=230;bb=230  #links length
AFM =-2*aa*xc10;
BFM =-2*aa*yc10;
CFM= xc10**2 +aa**2 + yc10**2 -bb**2;
th10h=2*m.atan2((-BFM -m.sqrt(BFM**2 + AFM**2 - CFM**2)),(CFM -AFM))*(180/m.pi);
th20h=(m.atan2(yc10-aa*m.sin(th10h*m.pi/180),xc10-aa*m.cos(th10h*m.pi/180))*180/m.pi)-th10h;
#th10h ve th20h Pic'e yollanacak raspberryden!!!!
while True:
   try:
      writeNumber(th10h,th20h)
      time.sleep(1)              #delay one second


   except KeyboardInterrupt:
      quit()
if __name__ == '__main__':
   setServoAngle(servo1, 180) #180 degree suppose gripper is closing at that angle
GPIO.cleanup()
#INVERSE FOR FINAL POINT OF SECOND MANİPULATOR GOİNG BACK
if __name__ == '__main__':
   setServoAngle(servo2, 0) #0 degree suppose gripper is opening at that angle


GPIO.cleanup()
```

```
aa=230;bb=230  #links length
ASM =-2*aa*xx2c;
BSM =-2*aa*yy2c;
CSM= xx2c**2 +aa**2 + yy2c**2 -bb**2;
th11h=2*m.atan2((-BSM -m.sqrt(BSM**2 + ASM**2 - CSM**2)),(CSM -ASM))*(180/m.pi);
th21h=(m.atan2(yy2c-aa*m.sin(th11h*m.pi/180),xx2c-aa*m.cos(th11h*m.pi/180))*180/m.pi)-th11h;
#th11h ve th21h Pic'e yollanacak raspberryden !!!!
while True:
   try:
      writeNumber(th11h,th21h)
      time.sleep(1)                #delay one second

   except KeyboardInterrupt:
      quit()
if __name__ == '__main__':
   setServoAngle(servo2, 180) #180 degree suppose gripper is closing at that angle

GPIO.cleanup()

#INVERSE FOR FINAL POINT OF THİRD MANİPULATOR
if __name__ == '__main__':
   setServoAngle(servo3, 0) #0 degree suppose gripper is opening at that angle

GPIO.cleanup()
aa=230;bb=230  #links length
ATM =-2*aa*XE3;
BTM =-2*aa*YE3;
CTM= XE3**2 +aa**2 + YE3**2 -bb**2;
th12h=2*m.atan2((-BTM -m.sqrt(BTM**2 + ATM**2 - CTM**2)),(CTM -ATM))*(180/m.pi);
```

```
th22h=(m.atan2(YE3-aa*m.sin(th12h*m.pi/180),XE3-aa*m.cos(th12h*m.pi/180))*180/m.pi)-th12h;
#th12h ve th22h Pic'e yollanacak raspberryden !!!!!
while True:
    try:
        writeNumber(th12h,th22h)
        time.sleep(1)           #delay one second

    except KeyboardInterrupt:
        quit()
```

## 4-Drawing square task(pure translation)

In this task we used just second and third  manipulator without any attaching and detaching operation so that it can go segment by segment until we reach to the final position of platform center .This task can be configure and apply into laser cutting job  which we chose it to be our main task configuration.




#Note:to move indentations or dedent press ctr+] or ctr+[

#just second and third manipulator will be implemented

```
import numpy as np
import sympy
import math as m
import matplotlib.pyplot as plt
from pylab import *
import time
import serial
def f(xg21,xg23,yg21,yg23,xg31,xg33,yg31,yg33): #DEFING function for instantly chanegable variables
                                ######################second manipulator code start here
  tf21=10;tf22=10  #time period definition
  t=np.arange(0,10,0.5);
```

```
#x2=80;y2=200 #second station of end effector
a2=230;b2=230  #links length
#algorithm to obtain via point and end effector positions
x22, y22 = sympy.symbols("x22 y22", real=True)  #to solve system equation
x21=-(277.13-xg21);    x23=-(277.13-xg23);     y21=-(160-yg21);       y23=-(160-yg23);#position w.r.t
manipulator itself
WW2=m.sqrt((y23-y21)**2+(x23-x21)**2)
THE2=30      ##bending angle (it can be change)
QQ21=(x22-x21)**2+(y22-y21)**2  ##first circle equ
QQ22=(x22-x23)**2+(y22-y23)**2  ##second circle equ
L2=(WW2/2)/(m.cos(THE2*m.pi/180))  ##midpoint of distance
CC21=L2**2  ##radius of 1st circle
CC22=L2**2  ##radius for 2nd circle
eq21 = sympy.Eq(QQ21, CC21)
eq22 = sympy.Eq(QQ22, CC22)
solution2=sympy.solve([eq21, eq22])  ##system equations solution
print(solution2)
S21=solution2[0]
S22=solution2[1]
VI2=[list(S21.values()),list(S22.values())]##values for both solution
##first solution
x22=VI2[0][0]
y22=VI2[0][1]
######via point has finish###
###INVERSE K FOR INTIAL POINT
A2O =-2*a2*x21;
B2O =-2*a2*y21;
C2O = x21**2 +a2**2 + y21**2 -b2**2;
th210=2*m.atan2((-B2O -m.sqrt(B2O**2 + A2O**2 - C2O**2)),(C2O - A2O))*(180/m.pi);
```

```
th220=(m.atan2(y21-a2*m.sin(th210*m.pi/180),x21-a2*m.cos(th210*m.pi/180))*180/m.pi)-th210;

#INVERSE K FOR via POINT

AM2=-2*a2*x22;

BM2=-2*a2*y22;

CM2=x22**2 +a2**2 + y22**2 -b2**2;

th211=2*m.atan2((-BM2 -m.sqrt(BM2**2 + AM2**2 - CM2**2)),(CM2 - AM2))*(180/m.pi);

th221=(m.atan2(y22-a2*m.sin(th211*m.pi/180),x22-a2*m.cos(th211*m.pi/180))*180/m.pi)-th211;

#INVERSE K FOR FINAL POINT

AF2=-2*a2*x23;

BF2=-2*a2*y23;

CF2=x23**2 +a2**2 +y23**2-b2**2;

th212=2*m.atan2((-BF2 -m.sqrt(BF2**2 + AF2**2 - CF2**2)),(CF2 -AF2))*(180/m.pi);

th222=(m.atan2(y23-a2*m.sin(th212*m.pi/180),x23-a2*m.cos(th212*m.pi/180))*180/m.pi)-th212;

#8 by 8 matrices FOR FIRST ACTUATOR

B21=np.array([[th210],[th211],[th211],[th212],[0],[0],[0],[0]]) #values of abgles obtained from inverse
kinematic 1.st actuator

A21=np.array([[1 ,0 ,0, 0, 0 ,0, 0, 0],[1 ,tf21, tf21*tf21, tf21*tf21*tf21, 0, 0, 0 ,0],

    [0, 0 ,0 ,0 ,1 ,0, 0, 0 ],[0 ,0 ,0 ,0, 1, tf22, tf22*tf22, tf22*tf22*tf22],

    [0 ,1 ,0 ,0, 0 ,0, 0, 0 ],[0, 0, 0, 0, 0, 1, 2*tf22, 3*tf22*tf22],

    [0 ,1 ,2*tf21+3*tf21*tf21, 0, 0, -1, 0 ,0,] ,[0, 0, 2, 6*tf21, 0 ,0 ,-2, 0] ])

A21N= np.linalg.inv(A21)

C21=np.matmul(A21N,B21)

#print(th10,th11,th12)

t21s1=C21[0,0]+C21[1,0]*t+C21[2,0]*t**2+C21[3,0]*t**3; #first segment equation for 1.st act

t21s2=C21[4,0]+C21[5,0]*t+C21[6,0]*t**2+C21[7,0]*t**3; #second segment equation  for 1st act

#8 by 8 matrices FOR second ACTUATOR

B22=np.array([[th220],[th221],[th221],[th222],[0],[0],[0],[0]]) #values of angles obtained from inverse
kinematic 1.st actuator

A22=np.array([[1 ,0 ,0, 0, 0 ,0, 0, 0],[1 ,tf21, tf21*tf21, tf21*tf21*tf21, 0, 0, 0 ,0],
```

```
   [0, 0 ,0 ,0 ,1 ,0, 0, 0 ],[0 ,0 ,0 ,0, 1, tf22, tf22*tf22, tf22*tf22*tf22],
   [0 ,1 ,0 ,0, 0 ,0, 0, 0 ],[0, 0, 0, 0, 0, 1, 2*tf22, 3*tf22*tf22],
   [0 ,1 ,2*tf21+3*tf21*tf21, 0, 0, -1, 0 ,0,] ,[0, 0, 2, 6*tf21, 0 ,0 ,-2, 0] ])
A2N=np.linalg.inv(A22)
C22=np.matmul(A2N,B22)
#print(th20,th21,th22)
t22s1=C22[0,0]+C22[1,0]*t+C22[2,0]*t**2+C22[3,0]*t**3; #first segment equation for 1.st act
t22s2=C22[4,0]+C22[5,0]*t+C22[6,0]*t**2+C22[7,0]*t**3; #second segment equation  for  1.st act
tt21=[];tt22=[];tt23=[];tt24=[]  #empty lists for thetas erray
for aq21 in t21s1*100:
   bb21=('%0.0f'%aq21)
   #print(bb1)
   tt21.append(int(bb21))
   #tt1.append(int(bb21)/100)
   #time.sleep(1)
tm21s1=tt21  #list to be send for first segment of first motor
print(tm21s1)
for aq22 in t21s2*100:


   bb22=('%0.0f'%aq22)
   #print(bb22)
   tt22.append(int(bb22))
   #time.sleep(1)
tm21s2=tt22  #list to be send for second segment of first motor
print(tm21s2)
for aq23 in t22s1*100:
   bb23=('%0.0f'%aq23)
   #print(bb23)
   tt23.append(int(bb23))
```

```
        #time.sleep(1)
    tm22s1=tt23  #list to be send for first segment of second motor
    print(tm22s1)
    for aq24 in t22s2*100:
        bb24=('%0.0f'%aq24)
        ##print(bb4)
        tt24.append(int(bb24))
        #time.sleep(1)
    tm22s2=tt24  #list to be send for second segment of second motor
    print(tm22s2)
    print('Second manipulator data')
                 ##second manipulator code end point
                                          #####################thid manipulator code start here
    tf31=10;tf32=10  #time period definition
    t=np.arange(0,10,0.5);
    #x2=80;y2=200 #second station of end effector
    a3=230;b3=230  #links length
    #algorithm to obtain via point start
    x32, y32 = sympy.symbols("x32 y32", real=True)  #to solve system equation
    x31=-(-277.13-xg31);      x33=-(-277.13-xg33);      y31=-(160-yg31);      y33=-(160-yg33);#first and last
station of end effector
    WW3=m.sqrt((y33-y31)**2+(x33-x31)**2)
    THE3=30      ##bending angle (it can be change)
    QQ31=(x32-x31)**2+(y32-y31)**2  ##first circle equ
    QQ32=(x32-x33)**2+(y32-y33)**2  ##second circle equ
    L3=(WW3/2)/(m.cos(THE3*m.pi/180))  ##midpoint of distance
    CC31=L3**2  ##radius of 1st circle
    CC32=L3**2  ##radius for 2nd circle
    eq31 = sympy.Eq(QQ31, CC31)
```

```
eq32 = sympy.Eq(QQ32, CC32)
solution3=sympy.solve([eq31, eq32])  ##system equations solution
print(solution3)
S31=solution3[0]
S32=solution3[1]
VI3=[list(S31.values()),list(S32.values())]##values for both solution
##first solution
x32=VI3[0][0]
y32=VI3[0][1]
######via point has finish###
###INVERSE K FOR INTIAL POINT
A3O =-2*a3*x31;
B3O =-2*a3*y31;
C3O = x31**2 +a3**2 + y31**2 -b3**2;
th310=2*m.atan2((-B3O -m.sqrt(B3O**2 + A3O**2 - C3O**2)),(C3O - A3O))*(180/m.pi);
th320=(m.atan2(y31-a3*m.sin(th310*m.pi/180),x31-a3*m.cos(th310*m.pi/180))*180/m.pi)-th310;
#INVERSE K FOR via POINT
AM3=-2*a3*x32;
BM3=-2*a3*y32;
CM3=x32**2 +a3**2 + y32**2 -b3**2;
th311=2*m.atan2((-BM3 -m.sqrt(BM3**2 + AM3**2 - CM3**2)),(CM3 - AM3))*(180/m.pi);
th321=(m.atan2(y32-a3*m.sin(th311*m.pi/180),x32-a3*m.cos(th311*m.pi/180))*180/m.pi)-th311;
#INVERSE K FOR FINAL POINT
AF3=-2*a3*x33;
BF3=-2*a3*y33;
CF3=x33**2 +a3**2 +y33**2-b3**2;
th312=2*m.atan2((-BF3 -m.sqrt(BF3**2 + AF3**2 - CF3**2)),(CF3 -AF3))*(180/m.pi);
th322=(m.atan2(y33-a3*m.sin(th312*m.pi/180),x33-a3*m.cos(th312*m.pi/180))*180/m.pi)-th312;
#8 by 8 matrices FOR FIRST ACTUATOR
```

```
B31=np.array([[th310],[th311],[th311],[th312],[0],[0],[0],[0]]) #values of abgles obtained from inverse kinematic 1.st actuator
A31=np.array([[1 ,0 ,0, 0, 0 ,0, 0, 0],[1 ,tf31, tf31*tf31, tf31*tf31*tf31, 0, 0, 0 ,0],
    [0, 0 ,0 ,0 ,1 ,0, 0, 0 ],[0 ,0 ,0 ,0, 1, tf32, tf32*tf32, tf32*tf32*tf32],
    [0 ,1 ,0 ,0, 0 ,0, 0, 0 ],[0, 0, 0, 0, 0, 1, 2*tf32, 3*tf32*tf32],
    [0 ,1 ,2*tf31+3*tf31*tf31, 0, 0, -1, 0 ,0,] ,[0, 0, 2, 6*tf31, 0 ,0 ,-2, 0] ])
A31N= np.linalg.inv(A31)
C31=np.matmul(A31N,B31)
#print(th30,th31,th12)
t31s1=C31[0,0]+C31[1,0]*t+C31[2,0]*t**2+C31[3,0]*t**3; #first segment equation for 1.st act
t31s2=C31[4,0]+C31[5,0]*t+C31[6,0]*t**2+C31[7,0]*t**3; #second segment equation  for 1st act
#8 by 8 matrices FOR second ACTUATOR
B32=np.array([[th320],[th321],[th321],[th322],[0],[0],[0],[0]]) #values of angles obtained from inverse kinematic 1.st actuator
A32=np.array([[1 ,0 ,0, 0, 0 ,0, 0, 0],[1 ,tf31, tf31*tf31, tf31*tf31*tf31, 0, 0, 0 ,0],
    [0, 0 ,0 ,0 ,1 ,0, 0, 0 ],[0 ,0 ,0 ,0, 1, tf32, tf32*tf32, tf32*tf32*tf32],
    [0 ,1 ,0 ,0, 0 ,0, 0, 0 ],[0, 0, 0, 0, 0, 1, 2*tf32, 3*tf32*tf32],
    [0 ,1 ,2*tf31+3*tf31*tf31, 0, 0, -1, 0 ,0,] ,[0, 0, 2, 6*tf31, 0 ,0 ,-2, 0] ])
A3N=np.linalg.inv(A32)
C32=np.matmul(A3N,B32)
#print(th20,th21,th22)
t32s1=C32[0,0]+C32[1,0]*t+C32[2,0]*t**2+C32[3,0]*t**3; #first segment equation for 1.st act
t32s2=C32[4,0]+C32[5,0]*t+C32[6,0]*t**2+C32[7,0]*t**3; #second segment equation  for
tt31=[];tt32=[];tt33=[];tt34=[]  #empty lists for thetas array
for aq31 in t31s1*100:
    bb31=('%0.0f'%aq31)
    #print(bb3)
    tt31.append(int(bb31))
    #time.sleep(1)
```

```
    tm31s1=tt31  #list to be send for first segment of first motor
    print(tm31s1)
    for aq32 in t31s2*100:

        bb32=('%0.0f'%aq32)
        #print(bb22)
        tt32.append(int(bb32))
        #time.sleep(1)
    tm31s2=tt32  #list to be send for second segment of first motor
    print(tm31s2)
    for aq33 in t32s1*100:
        bb33=('%0.0f'%aq33)
        #print(bb23)
        tt33.append(int(bb33))
        #time.sleep(1)
    tm32s1=tt33  #list to be send for first segment of second motor
    print(tm32s1)
    for aq34 in t32s2*100:
        bb34=('%0.0f'%aq34)
        ##print(bb4)
        tt34.append(int(bb34))
        #time.sleep(1)
    tm32s2=tt34  #list to be send for second segment of second motor
    print(tm32s2)
    print('Third manipulator data')
    #print("verification done")


for i in [0,1,2,3,4]:  ##slicing operation into five segment
```

```
    if i==0: #first segment will be run from stationary point of end effector to first corner of square
        xg21=60*m.cos(60*m.pi/180);  xg23=60*m.cos(60*m.pi/180);    yg21=160+60*m.sin(60*m.pi/180);
yg23=100+60*m.sin(60*m.pi/180);  ##position w.r.t global frame(2nd manipulator)
        xg31=-60;             xg33=-60;       yg31=160;         yg33=100;  ##position w.r.t global frame (third
manipulator)
        f(xg21,xg23,yg21,yg23,xg31,xg33,yg31,yg33)##calling function to evaluatee suitable condition
        print("stationary  line^^^^^^^ ")
        print("------------------------------------------------------------------------------------------------------------------------
-------- ")
        time.sleep(1)
    if i==1:#second segment will be run from first to second corner of square
        xg21=60*m.cos(60*m.pi/180);  xg23=130;    yg21=100+60*m.sin(60*m.pi/180);
yg23=100+60*m.sin(60*m.pi/180);##position w.r.t global frame(2nd manipulator)
        xg31=-60;             xg33=40;       yg31=100;         yg33=100;##position w.r.t global frame (third
manipulator)
        f(xg21,xg23,yg21,yg23,xg31,xg33,yg31,yg33)##calling function to evaluatee suitable condition
        print("square first  line CCW^^^^^^^ ")
        print("------------------------------------------------------------------------------------------------------------------------
-------- ")
        time.sleep(1)
    if i==2:#third segment will be run from second to third corner of square
         xg21=130;  xg23=130;   yg21=151.96; yg23=51.96 ##position w.r.t global frame(2nd manipulator)
         xg31=40;  xg33=40;   yg31=100;         yg33=0;##position w.r.t global frame (third manipulator)
         f(xg21,xg23,yg21,yg23,xg31,xg33,yg31,yg33)##calling function to evaluatee suitable condition
         print("square second  line CCW^^^^^^^ ")
         print("-----------------------------------------------------------------------------------------------------------------------
----------- ")
         time.sleep(1)

    if i==3:  #fourth  segment will be run from third to fourth corner of square
        xg21=100+60*m.cos(60*m.pi/180);  xg23=30;    yg21=60*m.sin(60*m.pi/180);
yg23=60*m.sin(60*m.pi/180);##position w.r.t global frame(2nd manipulator)
```

```
        xg31=40;              xg33=-60;       yg31=0;         yg33=0;##position w.r.t global frame (third
manipulator)

        f(xg21,xg23,yg21,yg23,xg31,xg33,yg31,yg33)##calling function to evaluatee suitable condition

        print("square third  line CCW^^^^^^^ ")

        print("--------------------------------------------------------------------------------------------------------------------------
------------- ")

        time.sleep(1)

    if i==4:#fifth segment will be run from fourth to first corner of square

        xg21=60*m.cos(60*m.pi/180);  xg23=60*m.cos(60*m.pi/180);    yg21=60*m.sin(60*m.pi/180);
yg23=100+60*m.sin(60*m.pi/180);##position w.r.t global frame(2nd manipulator)

        xg31=-60;             xg33=-60;       yg31=0;         yg33=100;##position w.r.t global frame (third
manipulator)

        f(xg21,xg23,yg21,yg23,xg31,xg33,yg31,yg33)##calling function to evaluatee suitable condition

        print("square fourth  line CCW ^^^^^^^^^^")

        print("--------------------------------------------------------------------------------------------------------------------------
----------- ")

        time.sleep(1)
```