```
#|-------------------------------------------------------------------------|#
#| Program and run 'Scara final 1.cydsn' on PSoC board to use this code.   |#
#|                                                                         |#
#| This code locates an object with a webcam. Based off of camera data, two|#
#| servo angles are calculated and transmitted to the PSoC board over UART.|#
#|-------------------------------------------------------------------------|#


#libraries
import serial
import numpy as np
import cv2
import random

##################################################
    #connect with PSoC
##################################################

ser=serial.Serial()
ser.baudrate=9600
ser.port='COM4'
ser.open()

i=bytearray([1])                    #initialize
ser.write(i)

startB=0
while (startB==0):                  #waiting for button to be pressed
    startB=bytearray(ser.read())    #read until button pressed

ser.close()
#button pressed, now camera setup -> scara is homing now

##################################################
    #camera setup
##################################################

cap=cv2.VideoCapture(1)     #camera
cm_to_pixelX=490.3/640.0    #conversion horizontal, mm to pixel
cm_to_pixelY=260.35/360.0   #conversion vertical, mm to pixel

#build homogenous transformation matrix for camera to scara base frame

#x rotation matrix: changed to y rot 180
R180_X=[[np.cos(np.pi),0,np.sin(np.pi)],[0,1,0],[-np.sin(np.pi),0,np.cos(np.pi)]]
Rad=(0.0/180)*np.pi                         #change theta to whatever it needs to be here
RZ=[[np.cos(Rad),-np.sin(Rad),0],[np.sin(Rad),np.cos(Rad),0],[0,0,1]] #z rotation matrix
R0_C=np.dot(R180_X,RZ)                      #camera to base frame rotation matrix
d0_C=[[218.53],[-91.173],[0]]               #displacement vector

H0_C=np.concatenate((R0_C,d0_C),1)          #combine rotation and dispacement vectors
H0_C=np.concatenate((H0_C,[[0,0,0,1]]),0)   #homogenous transformation matrix


#-------------------------------------------------------------
#initial values for inside loop
nanCount=0
frameCount=0
initFrame=0
#-------------------------------------------------------------

##################################################
    #infinite loop
##################################################

while(1):

```

```
68
69      ###################################################
70      #wait for user to set background
71      ###################################################
72      ser.open()
73      startB=0
74      while (startB==0):                  #waiting for button to be pressed
75          startB=bytearray(ser.read())    #read until button pressed
76
77      ser.close()
78
79
80      ###################################################
81      #set background
82      ###################################################
83      timer=0
84      while(1):
85
86          notReading,frame=cap.read()                             #camera starts reading
87          gray_image1=cv2.cvtColor(frame,cv2.COLOR_BGR2GRAY)  #conversion RGB to grayscale
88          #cv2.imshow('background',frame)
89          k=cv2.waitKey(100)
90          if k==27:                                               #escape key
91              break
92          if timer < 5:
93              timer=timer+1
94          else:                                                   #or on 5th frame
95              break
96
97
98      ###################################################
99      #locate object from camera
100     ###################################################
101         #Brief order of while loop:
102         #1. capture image
103         #2. subtract image from background image
104         #3. calculate center of brightness from subtraction to get x,y camera coordinates
105         #4. coordinate transformation, camera to scara base frame
106         #5. add error for final calculated position
107         #6. do the whole while loop again to get a 2nd set of x,y coordinates
108         #7. compare 2 sets of coordinates
109         #8. if difference is +/-0.8: counter is set to +1, if difference is greater:
            counter is set to 0
110         #9. if 10 frames in a row are in tolerance, the while loop ends
111         #10. x,y coordinates of final frame comparison is used as the objects position
112     while(1):
113
114         _,frame=cap.read()                                      #camera starts reading
115         gray_image2=cv2.cvtColor(frame,cv2.COLOR_BGR2GRAY)  #conversion RGB to grayscale
116         #cv2.imshow('foreground',gray_image2)
117
118
            Difference=np.absolute(np.matrix(np.int16(gray_image1))-np.matrix(np.int16(gray_i
            mage2)))   #subtract images
119         Difference[Difference>255]=255                          #for uint8
120         Difference=np.uint8(Difference)
121         cv2.imshow('difference',Difference)
122
123         BW=Difference   #set threshold
124         BW[BW<=100]=0
125         BW[BW>100]=1
126
127         ##X column center of brightness location
128
129         column_sums=np.matrix(np.sum(BW,0))       #step 1:sum columns
130         #step 2: 1st build matrix with numbers [1,2,....,639,640]
131         column_numbers=np.matrix(np.arange(640))
```

```
        #step 2: 2nd multiply matrices
        column_mult=np.multiply(column_sums,column_numbers)
        total=np.sum(column_mult)                   #step 3: sum multiplied matrix
        total_total=np.sum(np.sum(BW))              #step 4: sum of original matrix
        if total_total>0:                           #eliminates division by 0
            column_location=total/total_total       #step 5: divide
        else:
            column_location=random.random()*640     #random column
        X_Location=column_location*cm_to_pixelX     #convert


        ##Y row center of brightness

        row_sums=np.matrix(np.sum(BW,1))          #step 1:sum rows
        #step 1: take transpose of matrix so 360 coumns and 1 row
        row_sums=row_sums.transpose()
        #step 2: 1st build matrix with numbers [1,2,....,359,360]
        row_numbers=np.matrix(np.arange(360))
        row_mult=np.multiply(row_sums,row_numbers) #step 2: 2nd multiply matrices
        total=np.sum(row_mult)                      #step 3: sum multiplied matrix
        total_total=np.sum(np.sum(BW))              #step 4: sum of original matrix
        if total_total>0:                           #eliminates division by 0
            row_location=total/total_total          #step 5: divide
        else:
            row_location=random.random()*360        #random row
        Y_Location=row_location*cm_to_pixelY        #convert

        #coordinate transformation: use hmt to find x,y coordinates in the base frame
        PC= [[X_Location],[Y_Location],[0],[1]]      #point found by camera
        P0=np.dot(H0_C,PC)                          #point in base frame

        X0=P0[0]                                    #x location in the base frame
        Y0=P0[1]                                    #y location in the base frame

        nanCheckX=np.isnan(X0)                      #check for nan x
        nanCheckY=np.isnan(Y0)                      #check for nan y


        #error correction-based off angle of base frame orgin to calculated x,y values
        #                   error found through testing and using averages
        #                   needs improvement

        X0c=np.copy(P0[0])
        Y0c=np.copy(P0[1])
        angleR=np.arctan(Y0c/X0c)*180/np.pi     #angle of base frame to x,y position
        if X0c<0:
            angleR=angleR+180

        if angleR<=15.0:                        #add error to calculated values
            X0c=X0c-5.4
            Y0c=Y0c+6.8
        if angleR>15.0 and angleR<=45.0:
            X0c=X0c-6.7
            Y0c=Y0c+5.3
        if angleR>45.0 and angleR<=75.0:
            X0c=X0c-6.9
            Y0c=Y0c+5.6
        if angleR>75.0 and angleR<=90.0:
            X0c=X0c-4.1
            Y0c=Y0c+1.3
        if angleR>90.0 and angleR<=105.0:
            X0c=X0c+3.7
            Y0c=Y0c+0.9
        if angleR>105.0 and angleR<=135.0:
            X0c=X0c+3.4
            Y0c=Y0c+0.0
        if angleR>135.0 and angleR<=165.0:
```

```
            X0c=X0c+13.1
            Y0c=Y0c+2.1
        if angleR>165.0 and angleR<=180.0:
            X0c=X0c+15.2
            Y0c=Y0c+6.7
        if angleR>180.0:
            X0c=X0c+9.7
            Y0c=Y0c+6.4

        #compare coordinates from previous frame, set counter accordingly
        if nanCheckX == True or nanCheckY == True:  #nan value resets count
            frameCount=0
        elif initFrame==0:                          #initial frame
            X0c1=0.0                                #previous x coordinate
            Y0c1=0.0                                #previous y coordinate
            X0c2=X0c                                #current x coordinate
            Y0c2=Y0c                                #current y coordinate
            initFrame=1                             #never resets
        else:
            X0c1=np.abs(X0c2)                       #previous x coordinate
            Y0c1=np.abs(Y0c2)                       #previous y coordinate
            X0c2=np.abs(X0c)                        #current x coordinate
            Y0c2=np.abs(Y0c)                        #current y coordinate
            xdif=np.abs(X0c2-X0c1)                  #x difference
            ydif=np.abs(Y0c2-Y0c1)                  #y difference
            if xdif<0.8 and ydif<0.8:               #tolerance
                frameCount+=1                       #in tolerance
            else:
                frameCount=0                        #out of tolerance


        k=cv2.waitKey(100)
        if k==27:                                   #escape key
            break
        if frameCount>9:                            #counter made it to 10
            break

    cv2.destroyAllWindows()

    ###################################################
    #inverse kinematics to find servo angles
    ###################################################

    X=X0c
    Y=Y0c
    a2=75   #linkage lengths (mm)
    a4=71

    #first calculate inverse kinematics as elbow down configuration
    r1= np.sqrt((X*X)+(Y*Y))    #equation 1
    Q=1                         #out of range variable, 1 is out of range, 0 is in range
    q=0                         #another out of range variable
    if r1<103.276 or r1>146.0:  #r1 has to be within this range
        q=1                     #set as out of range
        r1=105.0                #make r1 in range

    phi1=np.arccos(((a2*a2)+(r1*r1)-(a4*a4))/(2*a2*r1)) #equation 2
    phi1d=phi1*180/np.pi                                #convert to degrees

    if X==0:                    #make sure equation 3 doesn't divide by 0
        X=0.000001
    phi2= np.arctan(Y/X)        #equation 3
    phi2d=phi2*180/np.pi        #convert to degrees

    phi3=np.arccos(((a2*a2)+(a4*a4)-(r1*r1))/(2*a2*a4)) #equation 5
    phi3d=phi3*180/np.pi                                #convert to degrees

```

```
266         T1=(phi2-phi1)*180/np.pi    #equation 4, T1 is the first servo angle, elbow down
267         if X<0:
268             T1=T1+180
269 
270         T2=(np.pi-phi3)*180/np.pi   #equation 6, T2 is the second servo angle, elbow down
271 
272 
273 
274         if T1>180 or T1<0 or T2>90 or T2<-90:       #check if servo angles are out of range
275             T1=(phi2+phi1)*180/np.pi                #equation 4, elbow up configuration
276             if X<0:
277                 T1=T1+180
278             T2=(-np.pi+phi3)*180/np.pi              #equation 6, elbow up configuration
279             if T1>180 or T1<0 or T2>90 or T2<-90:   #check if in range elbow up configuration
280                 Q=1                                 #not in range
281             else:
282                 Q=0                                 #in range elbow up configuration
283         else:
284             Q=0                                     #in range elbow down configuration
285 
286 
287         if Q>0 or q>0:                              #if any out of range variables were set
            to 1
288             T1=90.0;                                #set home angles
289             T2=0.0
290 
291         if nanCheckX == True or nanCheckY == True:  #if nan
292             T1=90.0                                 #set home angles
293             T2=0.0
294 
295 
296         ####################################################
297         #calculate PWM compare values from servo angles
298         ####################################################
299         min_comp1=1500
300         max_comp1=7270
301         min_angle1=0
302         max_angle1=180
303         Angle1=T1
304         Compare1 =
            int(((max_comp1-min_comp1)/(max_angle1-min_angle1))*(Angle1-min_angle1)+min_comp1)
            #2 point form, servo 1
305 
306         min_comp2=1500
307         max_comp2=7270
308         min_angle2=-90
309         max_angle2=90
310         Angle2=T2
311         Compare2 =
            int(((max_comp2-min_comp2)/(max_angle2-min_angle2))*(Angle2-min_angle2)+min_comp2)
            ##2 point form, servo 2
312 
313 
314         ####################################################
315         #prepare 8 bytes to send over UART
316         ####################################################
317 
318         k=[0,0,0,0,0,0,0,0]             #list of bytes
319 
320         k[0]=int(Compare1/256)          #compare value for servo 1, stored in 2 bytes
321         k[1]=int(Compare1-(256*k[0]))
322 
323         k[2]=int(Compare2/256)          #compare value for servo 2, stored in 2 bytes
324         k[3]=int(Compare2-(256*k[2]))
325 
326         if X0c<0:                       #X position for display, stored in 2 bytes
327             k[4]=100                    #first byte states if number is positive or negative
```

```
328         else:
329             k[4]=1
330         k[5]=np.abs(int(X0c))           #second byte is X position in mm
331 
332         if Y0c<0:                       #Y position for display, stored in 2 bytes
333             k[6]=100                    #first byte states if number is positive or negative
334         else:
335             k[6]=1
336         k[7]=np.abs(int(Y0c))           #second byte is Y position in mm
337 
338 
339         ####################################################
340         #send values over UART
341         ####################################################
342         ser.open()
343 
344         j=0
345         while (j<8):
346             i=bytearray([k[j]])
347             ser.write(i)
348             j=j+1
349 
350 
351         ser.close()
352 
```