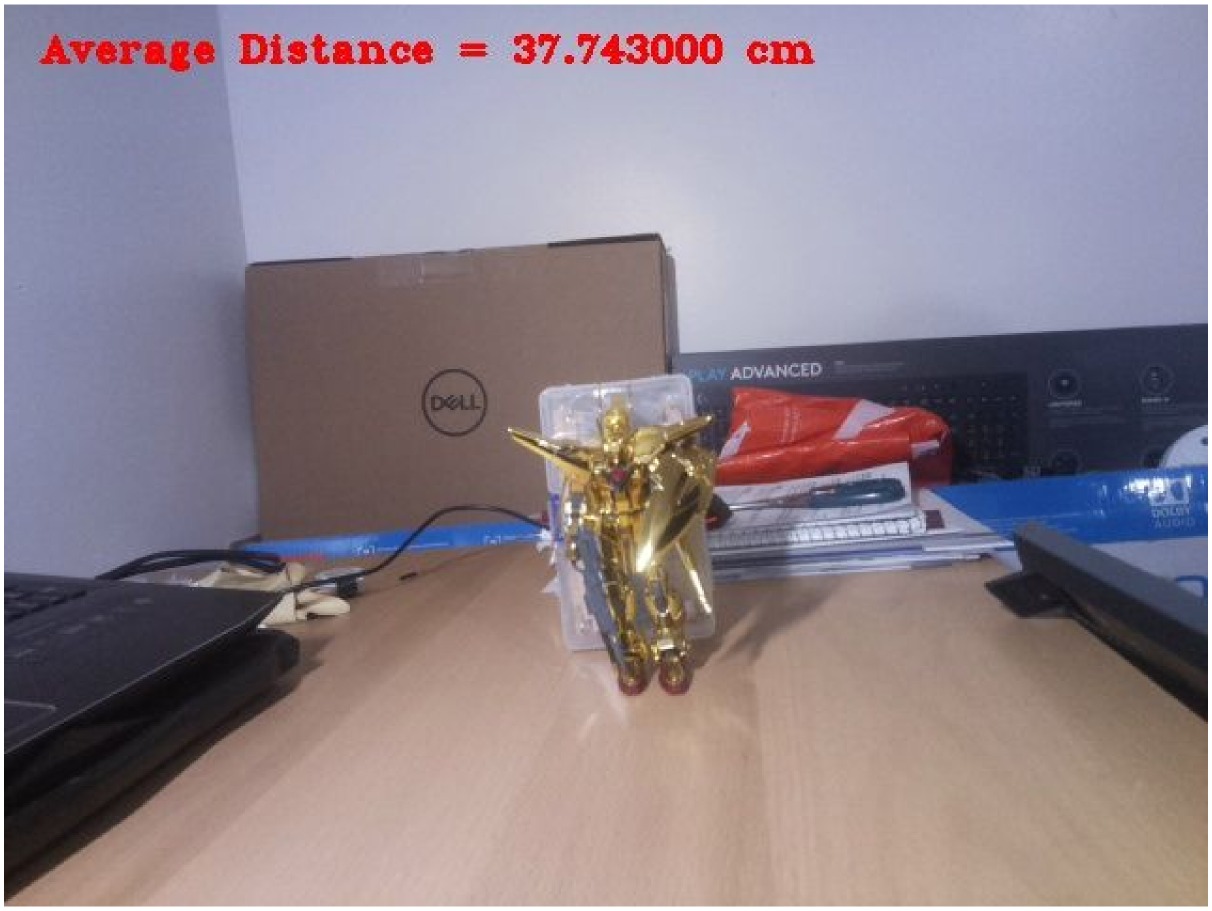
809T Homework 4

Charan Karthikeyan Parthasarathy Vasanthi

This assignment is split into two parts,one for finding the distance between the Ultrasonic Sensor and an object placed at a distance and calculate the distance between them. The Second is to read an arrow and find its orientation by rotating them. The video link for the second question is attached below. Please do provide feedback on the video.
Video 1:
https://youtu.be/NNCDum7YehY
Video 2:
https://youtu.be/eRoRnqdipDM

Part 1:

The first part of the homework involves in finding the distance between the Ultrasonic Sensor and the Object. The Object used in this image is a hobby figurine placed at a distance of 37-39 cm. The image showing the values of the distance calculated with the figurine is shown below.



Part 2:

The Second part involves in detecting a green arrow and find its orientation. This has been achieved by

1. Converting the frame into HSV and applying gaussian blur and masking the image with the Upper and Lower bounds of the color to detect, in our case which is green.
2. We then find the corner points from the masked image and plot them.
3. Then we find the corner point of the arrow head and the arrow tail from the array of corners, by finding the distance between each point.
4. The midpoint between the points at the arrow tail is calculated.
5. The slope between the arrowhead corner and the calculated midpoint is taken, and based on the angle of slope the orientation of the arrow is displayed and shown on the image.

Code:

The code for detecting and finding the orientation of the arrow is given below.

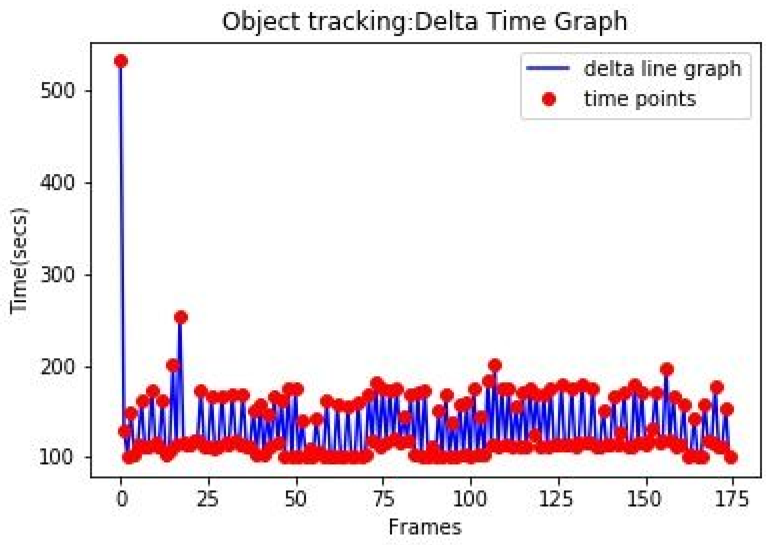
```
from picamera.array import PiRGBArray
from picamera import PiCamera
import time
import cv2
import datetime
import numpy as np
import imutils
import math

#Function for finding the orientation based on the
def direction_text(image,deg):

    if  -135< deg < -45:
        put_text(image,deg,"Up(North)")
    elif -180< deg< -125:
        put_text(image,deg,"Left(North West)")
    elif 145< deg< 180:
        put_text(image,deg,"Left(West)")
    elif 125 < deg <145:
        put_text(image,deg,"Left(South West)")
    elif 45 <deg< 145:
        put_text(image,deg,"Down(South)")
    elif 5< deg < 45:
        put_text(image,deg,"Right(South East)")
    elif 5 < deg <-5:
        put_text(image,deg,"Right(East)")
    else:
        put_text(image,deg,"Right(North East)")
    return image
```

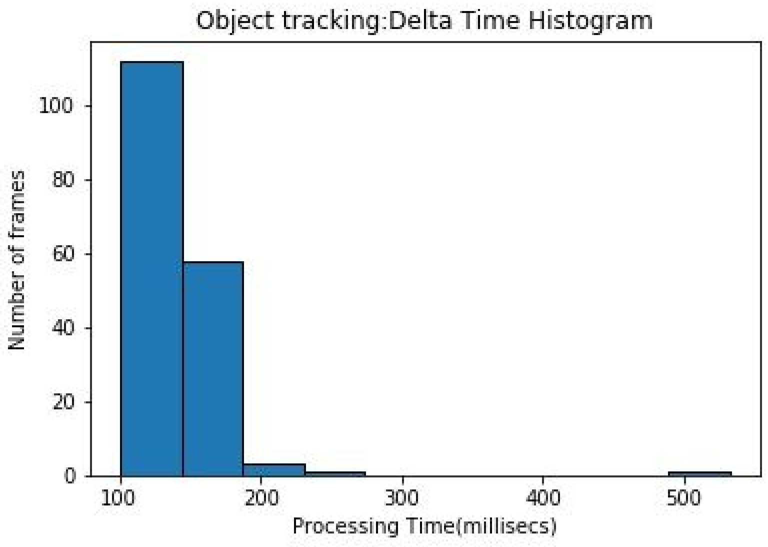
```
def put_text(image,deg,direc):
        font = cv2.FONT_HERSHEY_COMPLEX_SMALL
        clr = (0, 0, 255)
        cv2.putText(image, " Direction = %s and Slope = %f"%(direc,deg), (20, 30), font, 1, (clr),
        return image

camera = PiCamera()
camera.resolution = (640,480)
camera.framerate = 60
rawCapture = PiRGBArray(camera,size=(640,480))
fourcc = cv2.VideoWriter_fourcc(*'XVID')
out = cv2.VideoWriter("video_out.avi",fourcc,4,(640,480))
start_time = time.time()
num_arr = []
file = open("hw4_data.txt","w+")
time_thresh = 100
for frame in camera.capture_continuous(rawCapture,format="bgr",use_video_port=False):
        start_time_ = datetime.datetime.now()
        image = frame.array
        #lower_green= np.array ([40,200,220])
        lower_green = np.array([40,90,100])
        upper_green = np.array([100,255,190])
        #Color Conversion and Mask conversion
        hsv = cv2.cvtColor(image,cv2.COLOR_BGR2HSV)
        mask= cv2.inRange(hsv,lower_green,upper_green)
        #Apply Gaussian Blur to the image
        blur = cv2.GaussianBlur(mask,(9,9),0)
        result=cv2.bitwise_and(hsv, hsv, mask=mask)
        #Get Features from the image
        corners = cv2.goodFeaturesToTrack(blur,5,0.03,20)
        #print(corners)
        corners = np.int32(corners)
        #Plot the detected Corners in the Image
        for i in corners:
                x,y = i.ravel()
                cv2.circle(image,(x,y),4,(0,0,255),-1)

        points=np.array(corners).reshape(5,2)
        num_arr=(0,1,2,3,4)
        rest=[]
        #Find the distance between the detected points
        for i in num_arr:
                for j in range(i+1,len(num_arr)):
                 dist = np.linalg.norm(points[i]-points[j])
                 rest.append([dist,i,c])
        rest.sort()
        rest.reverse()
        temp_arr = []
        temp_arr.append(rest[0][1])
        temp_arr.append(rest[0][2])
        temp_arr.append(rest[1][1])
        temp_arr.append(rest[1][2])

        #Find the head point in the array of points
        head=-1
        for i in temp_arr:
                if temp_arr.count(i)>1:
                 head=i
                 break
        temp_arr.remove(head)
        temp_arr.remove(head)

        #Find the mid point between the arrow end points
        midpoint = (points[temp[0]] + points[temp[1]])/2
        #calculate the slope of the midopoint and the arrow head point
        dir = points[head] - midpoint
        rad = math.atan2(dir[1], dir[0])
        deg = math.degrees(rad)

        #Plot the Midpoint
        cv2.circle(image,(int(midpoint[0]),int(midpoint[1])),4,(0,0,255),-1)
        #Overlay text on the frame with the direction and slope
        image = direction_text(image,obstacle_gridgrees)

        cv2.imshow("frame",image)
        key = cv2.waitKey(1) & 0xFF
        rawCapture.truncate(0)
        if key== ord("q"):
                break
        time_out = time.time()-start_time
        if time_out >= time_thresh:
                break
        #Time and Write Functions
        stop_time = datetime.datetime.now()
        now = stop_time-start_time_
```

```
        out_values = str(now.total_seconds())+"\n"
        file.write(out_values)
        print("The time out is",time_out)
        out.write(image)
```

Graphs:

The Time timing graphs and histogram to show the difference in time or the time taken to process each frame is shown below.





Conclusion:

From the graphs above we can see that the delta time of the first frame spikes and comes to an oscillating time between 0.2 and 0.1 seconds a frames. This is also very evident when cross-referenced with the histogram graph. From this we can conclude that although there are frames that take some time to process a majority of the frames are within the operational time limits for the given orientation problem with most of them ranging between 0.1 to 0.2 seconds.