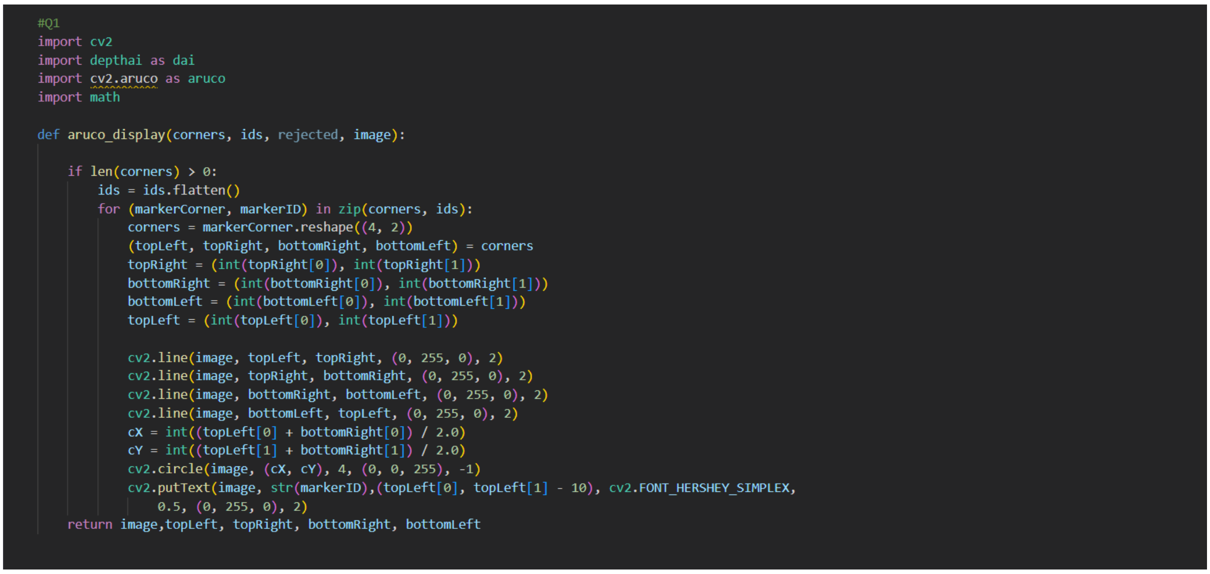
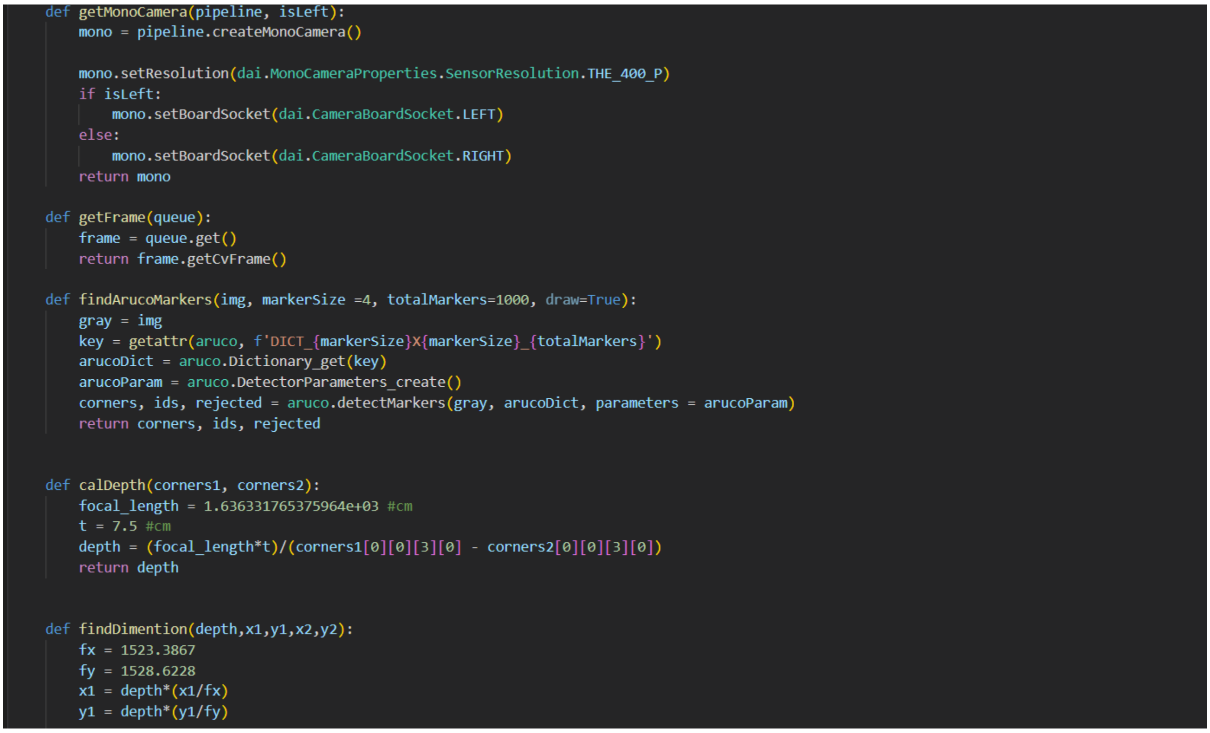
Assignment-4:

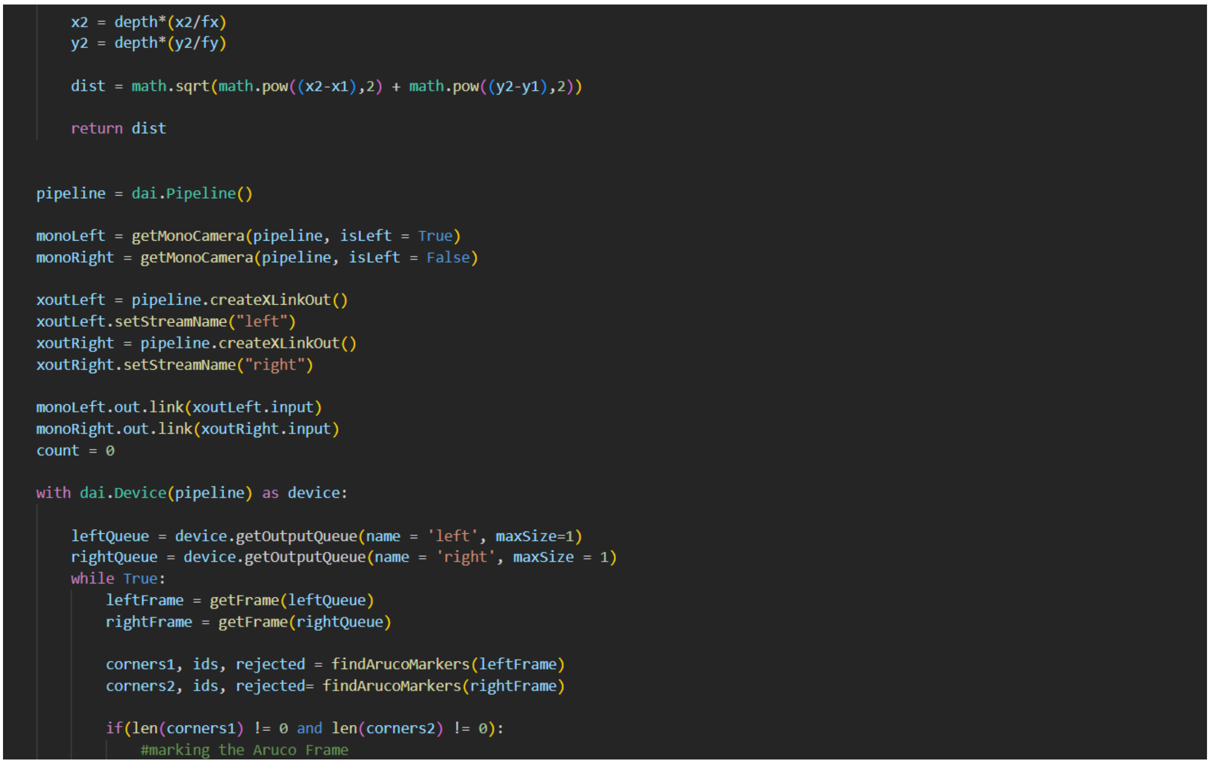
Q1:



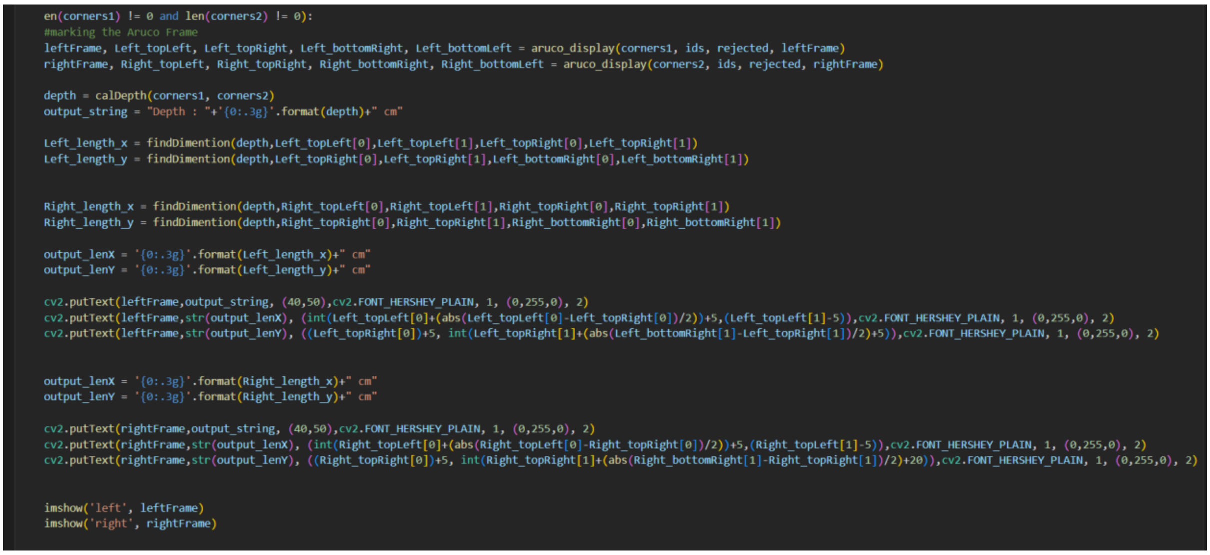
```
#Q1
import cv2
import depthai as dai
import cv2.aruco as aruco
import math

def aruco_display(corners, ids, rejected, image):

    if len(corners) > 0:
        ids = ids.flatten()
        for (markerCorner, markerID) in zip(corners, ids):
            corners = markerCorner.reshape((4, 2))
            (topLeft, topRight, bottomRight, bottomLeft) = corners
            topRight = (int(topRight[0]), int(topRight[1]))
            bottomRight = (int(bottomRight[0]), int(bottomRight[1]))
            bottomLeft = (int(bottomLeft[0]), int(bottomLeft[1]))
            topLeft = (int(topLeft[0]), int(topLeft[1]))

            cv2.line(image, topLeft, topRight, (0, 255, 0), 2)
            cv2.line(image, topRight, bottomRight, (0, 255, 0), 2)
            cv2.line(image, bottomRight, bottomLeft, (0, 255, 0), 2)
            cv2.line(image, bottomLeft, topLeft, (0, 255, 0), 2)
            cX = int((topLeft[0] + bottomRight[0]) / 2.0)
            cY = int((topLeft[1] + bottomRight[1]) / 2.0)
            cv2.circle(image, (cX, cY), 4, (0, 0, 255), -1)
            cv2.putText(image, str(markerID),(topLeft[0], topLeft[1] - 10), cv2.FONT_HERSHEY_SIMPLEX,
                0.5, (0, 255, 0), 2)
    return image,topLeft, topRight, bottomRight, bottomLeft
```



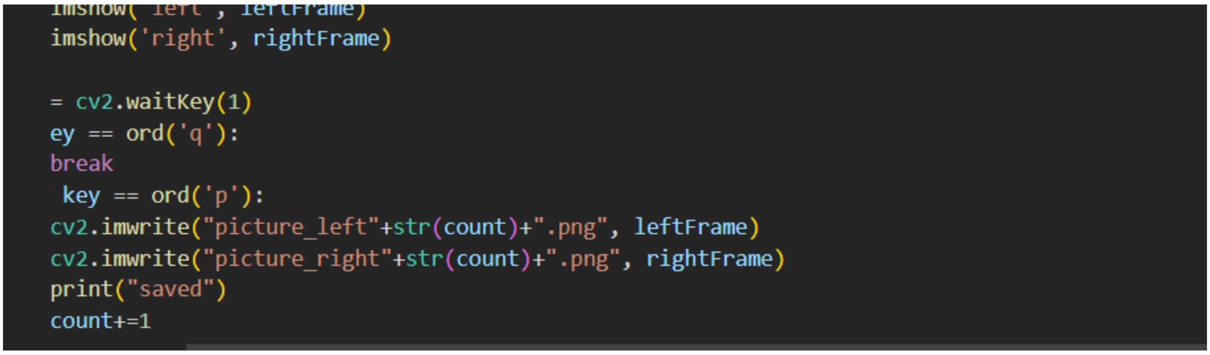
```
def getMonoCamera(pipeline, isLeft):
    mono = pipeline.createMonoCamera()

    mono.setResolution(dai.MonoCameraProperties.SensorResolution.THE_400_P)
    if isLeft:
        mono.setBoardSocket(dai.CameraBoardSocket.LEFT)
    else:
        mono.setBoardSocket(dai.CameraBoardSocket.RIGHT)
    return mono

def getFrame(queue):
    frame = queue.get()
    return frame.getCvFrame()

def findArucoMarkers(img, markerSize =4, totalMarkers=1000, draw=True):
    gray = img
    key = getattr(aruco, f'DICT_{markerSize}X{markerSize}_{totalMarkers}')
    arucoDict = aruco.Dictionary_get(key)
    arucoParam = aruco.DetectorParameters_create()
    corners, ids, rejected = aruco.detectMarkers(gray, arucoDict, parameters = arucoParam)
    return corners, ids, rejected


def calDepth(corners1, corners2):
    focal_length = 1.636331765375964e+03 #cm
    t = 7.5 #cm
    depth = (focal_length*t)/(corners1[0][0][3][0] - corners2[0][0][3][0])
    return depth


def findDimention(depth,x1,y1,x2,y2):
    fx = 1523.3867
    fy = 1528.6228
    x1 = depth*(x1/fx)
    y1 = depth*(y1/fy)
```



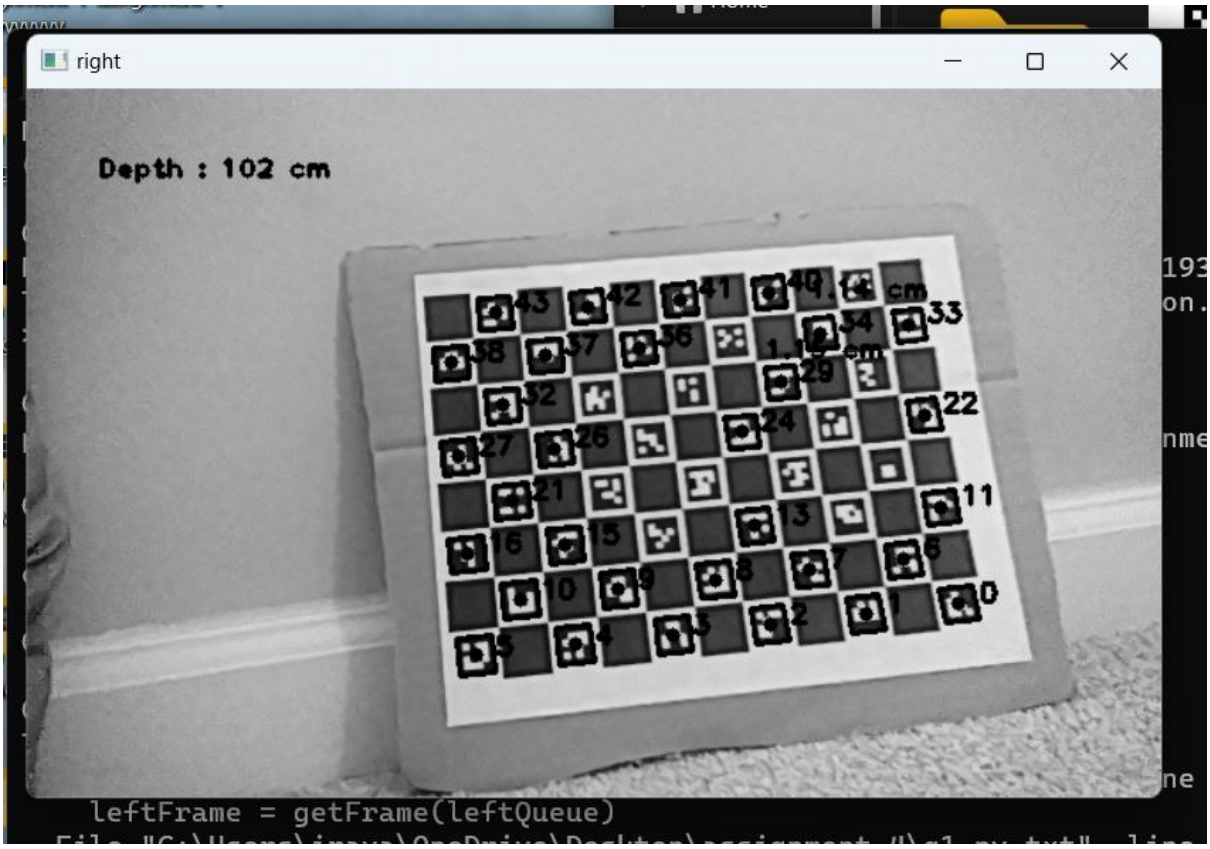
```
    x2 = depth*(x2/fx)
    y2 = depth*(y2/fy)

    dist = math.sqrt(math.pow((x2-x1),2) + math.pow((y2-y1),2))

    return dist


pipeline = dai.Pipeline()

monoLeft = getMonoCamera(pipeline, isLeft = True)
monoRight = getMonoCamera(pipeline, isLeft = False)

xoutLeft = pipeline.createXLinkOut()
xoutLeft.setStreamName("left")
xoutRight = pipeline.createXLinkOut()
xoutRight.setStreamName("right")

monoLeft.out.link(xoutLeft.input)
monoRight.out.link(xoutRight.input)
count = 0

with dai.Device(pipeline) as device:

    leftQueue = device.getOutputQueue(name = 'left', maxSize=1)
    rightQueue = device.getOutputQueue(name = 'right', maxSize = 1)
    while True:
        leftFrame = getFrame(leftQueue)
        rightFrame = getFrame(rightQueue)

        corners1, ids, rejected = findArucoMarkers(leftFrame)
        corners2, ids, rejected= findArucoMarkers(rightFrame)

        if(len(corners1) != 0 and len(corners2) != 0):
            #marking the Aruco Frame
```



```
en(corners1) != 0 and len(corners2) != 0):
#marking the Aruco Frame
leftFrame, Left_topLeft, Left_topRight, Left_bottomRight, Left_bottomLeft = aruco_display(corners1, ids, rejected, leftFrame)
rightFrame, Right_topLeft, Right_topRight, Right_bottomRight, Right_bottomLeft = aruco_display(corners2, ids, rejected, rightFrame)

depth = calDepth(corners1, corners2)
output_string = "Depth : "+'{0:.3g}'.format(depth)+" cm"

Left_length_x = findDimention(depth,Left_topLeft[0],Left_topLeft[1],Left_topRight[0],Left_topRight[1])
Left_length_y = findDimention(depth,Left_topRight[0],Left_topRight[1],Left_bottomRight[0],Left_bottomRight[1])


Right_length_x = findDimention(depth,Right_topLeft[0],Right_topLeft[1],Right_topRight[0],Right_topRight[1])
Right_length_y = findDimention(depth,Right_topRight[0],Right_topRight[1],Right_bottomRight[0],Right_bottomRight[1])

output_lenX = '{0:.3g}'.format(Left_length_x)+" cm"
output_lenY = '{0:.3g}'.format(Left_length_y)+" cm"

cv2.putText(leftFrame,output_string, (40,50),cv2.FONT_HERSHEY_PLAIN, 1, (0,255,0), 2)
cv2.putText(leftFrame,str(output_lenX), (int(Left_topLeft[0]+(abs(Left_topLeft[0]-Left_topRight[0])/2))+5,(Left_topLeft[1]-5)),cv2.FONT_HERSHEY_PLAIN, 1, (0,255,0), 2)
cv2.putText(leftFrame,str(output_lenY), ((Left_topRight[0])+5, int(Left_topRight[1]+(abs(Left_bottomRight[1]-Left_topRight[1])/2)+5)),cv2.FONT_HERSHEY_PLAIN, 1, (0,255,0), 2)


output_lenX = '{0:.3g}'.format(Right_length_x)+" cm"
output_lenY = '{0:.3g}'.format(Right_length_y)+" cm"

cv2.putText(rightFrame,output_string, (40,50),cv2.FONT_HERSHEY_PLAIN, 1, (0,255,0), 2)
cv2.putText(rightFrame,str(output_lenX), (int(Right_topLeft[0]+(abs(Right_topLeft[0]-Right_topRight[0])/2))+5,(Right_topLeft[1]-5)),cv2.FONT_HERSHEY_PLAIN, 1, (0,255,0), 2)
cv2.putText(rightFrame,str(output_lenY), ((Right_topRight[0])+5, int(Right_topRight[1]+(abs(Right_bottomRight[1]-Right_topRight[1])/2)+20)),cv2.FONT_HERSHEY_PLAIN, 1, (0,255,0), 2)


imshow('left', leftFrame)
imshow('right', rightFrame)
```

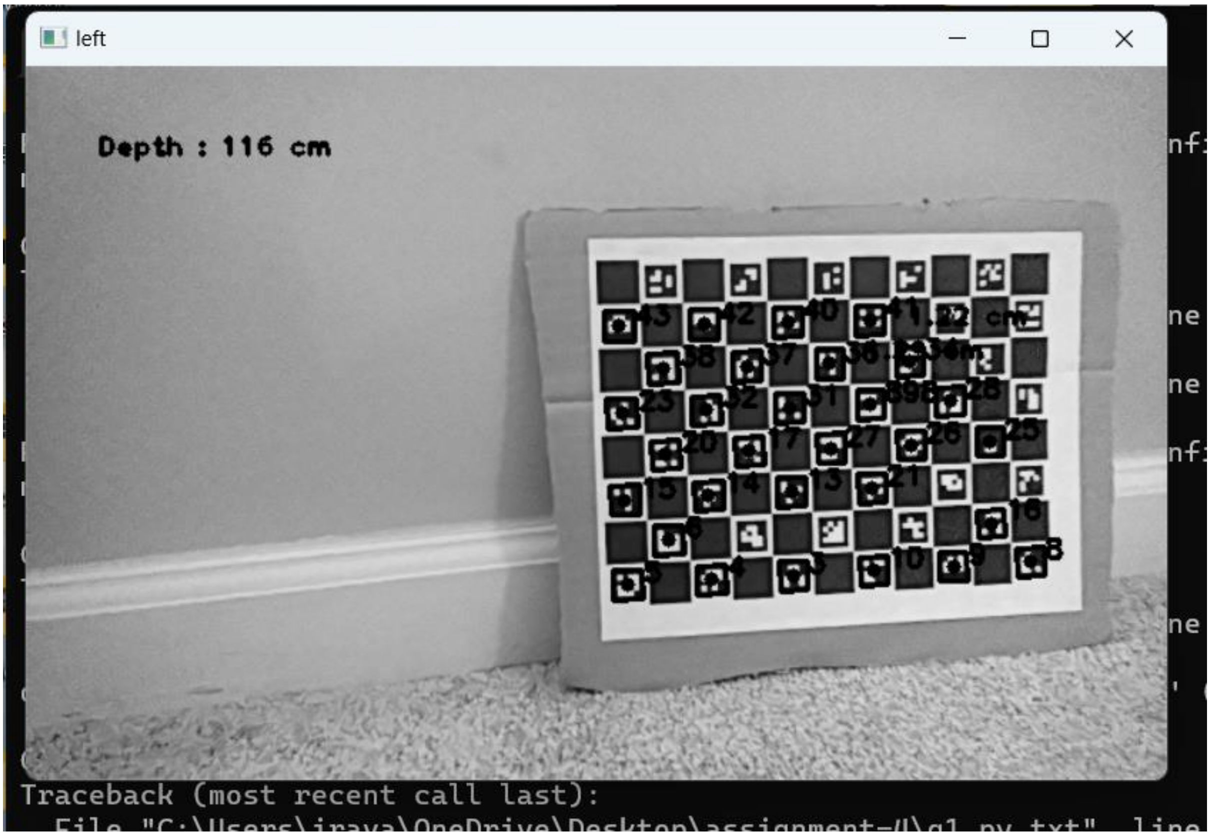


```
imshow('left', leftFrame)
imshow('right', rightFrame)

= cv2.waitKey(1)
ey == ord('q'):
break
 key == ord('p'):
cv2.imwrite("picture_left"+str(count)+".png", leftFrame)
cv2.imwrite("picture_right"+str(count)+".png", rightFrame)
print("saved")
count+=1
```



```
Depth : 102 cm
```

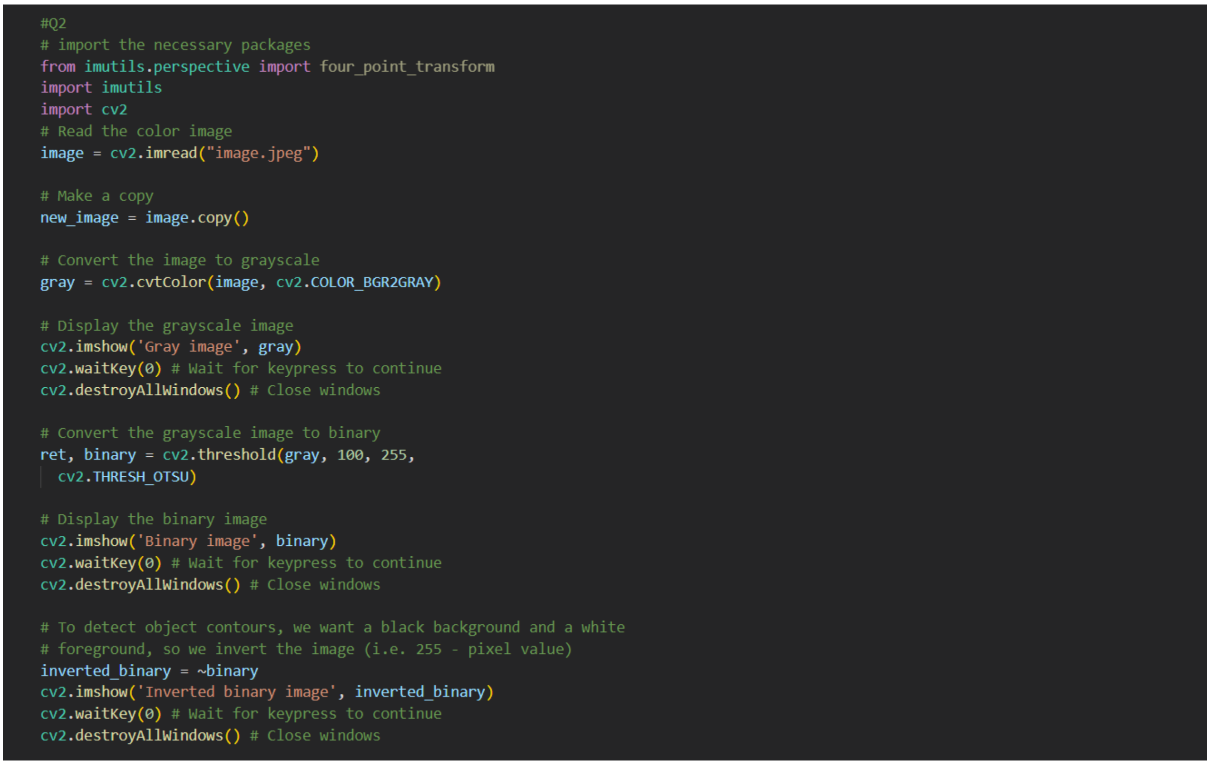
```
    leftFrame = getFrame(leftQueue)
  File "C:\Users\irava\OneDrive\Desktop\assignment-4\q1.py.txt", line
```



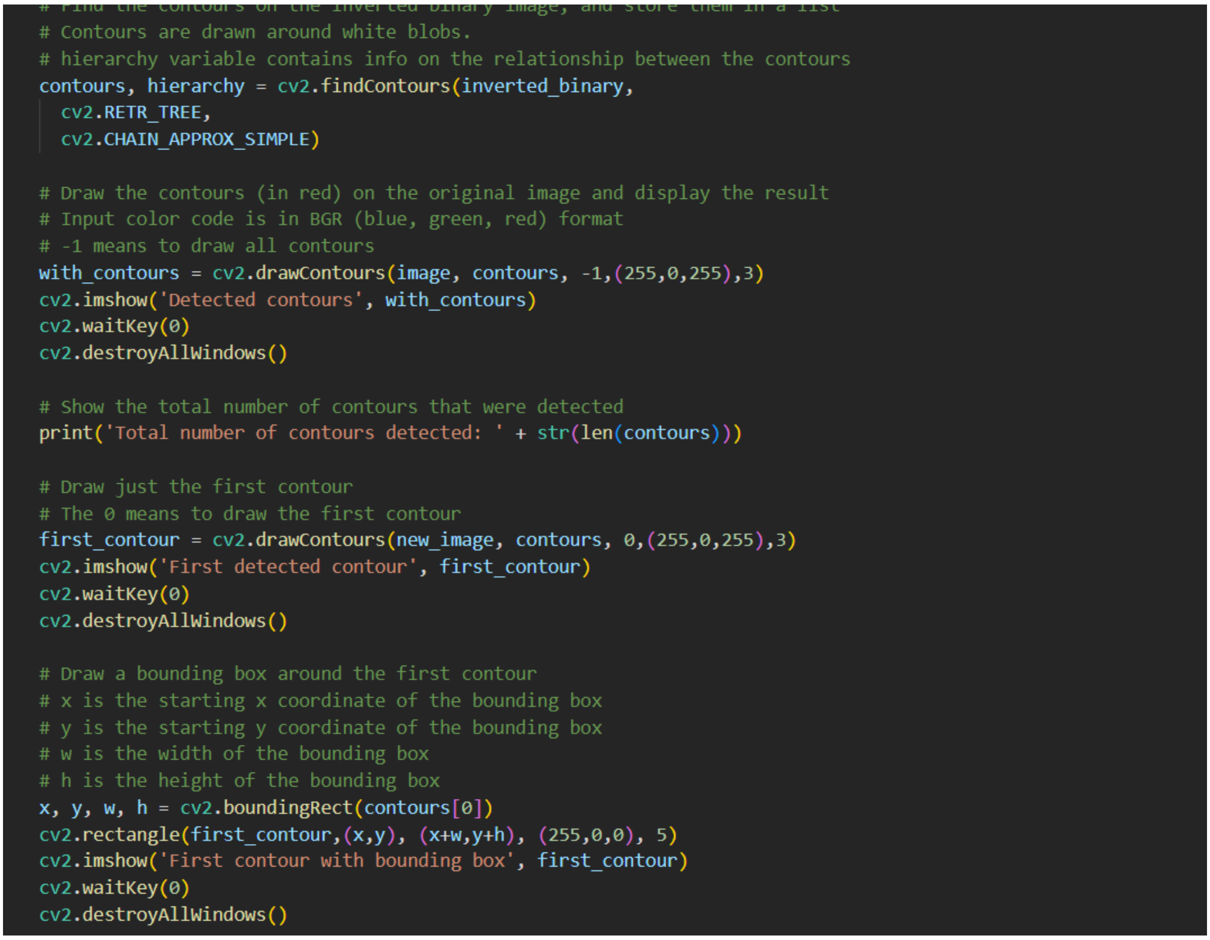
```
Depth : 116 cm
```

```
Traceback (most recent call last):
  File "C:\Users\irava\OneDrive\Desktop\assignment-4\q1.py.txt", line
```

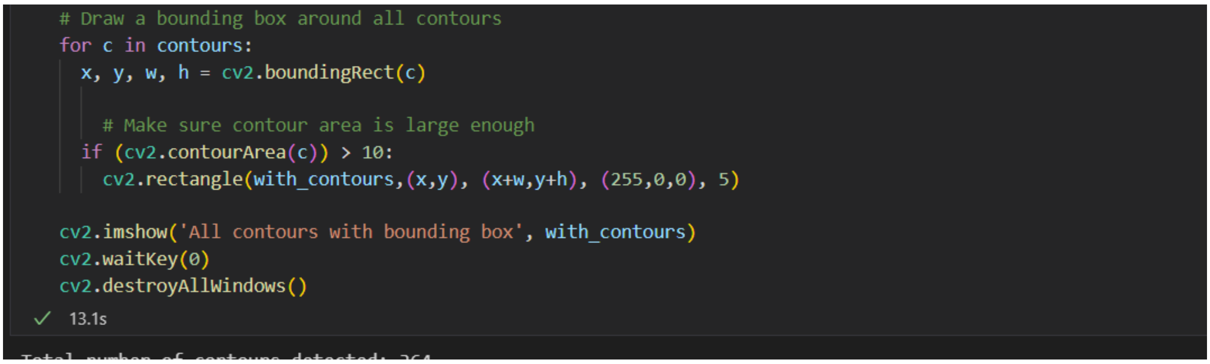
Q2:



```
#Q2
# import the necessary packages
from imutils.perspective import four_point_transform
import imutils
import cv2
# Read the color image
image = cv2.imread("image.jpeg")

# Make a copy
new_image = image.copy()

# Convert the image to grayscale
gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

# Display the grayscale image
cv2.imshow('Gray image', gray)
cv2.waitKey(0) # Wait for keypress to continue
cv2.destroyAllWindows() # Close windows

# Convert the grayscale image to binary
ret, binary = cv2.threshold(gray, 100, 255,
  cv2.THRESH_OTSU)

# Display the binary image
cv2.imshow('Binary image', binary)
cv2.waitKey(0) # Wait for keypress to continue
cv2.destroyAllWindows() # Close windows

# To detect object contours, we want a black background and a white
# foreground, so we invert the image (i.e. 255 - pixel value)
inverted_binary = ~binary
cv2.imshow('Inverted binary image', inverted_binary)
cv2.waitKey(0) # Wait for keypress to continue
cv2.destroyAllWindows() # Close windows
```



```
# Find the contours on the inverted binary image, and store them in a list
# Contours are drawn around white blobs.
# hierarchy variable contains info on the relationship between the contours
contours, hierarchy = cv2.findContours(inverted_binary,
  cv2.RETR_TREE,
  cv2.CHAIN_APPROX_SIMPLE)

# Draw the contours (in red) on the original image and display the result
# Input color code is in BGR (blue, green, red) format
# -1 means to draw all contours
with_contours = cv2.drawContours(image, contours, -1,(255,0,255),3)
cv2.imshow('Detected contours', with_contours)
cv2.waitKey(0)
cv2.destroyAllWindows()

# Show the total number of contours that were detected
print('Total number of contours detected: ' + str(len(contours)))

# Draw just the first contour
# The 0 means to draw the first contour
first_contour = cv2.drawContours(new_image, contours, 0,(255,0,255),3)
cv2.imshow('First detected contour', first_contour)
cv2.waitKey(0)
cv2.destroyAllWindows()

# Draw a bounding box around the first contour
# x is the starting x coordinate of the bounding box
# y is the starting y coordinate of the bounding box
# w is the width of the bounding box
# h is the height of the bounding box
x, y, w, h = cv2.boundingRect(contours[0])
cv2.rectangle(first_contour,(x,y), (x+w,y+h), (255,0,0), 5)
cv2.imshow('First contour with bounding box', first_contour)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

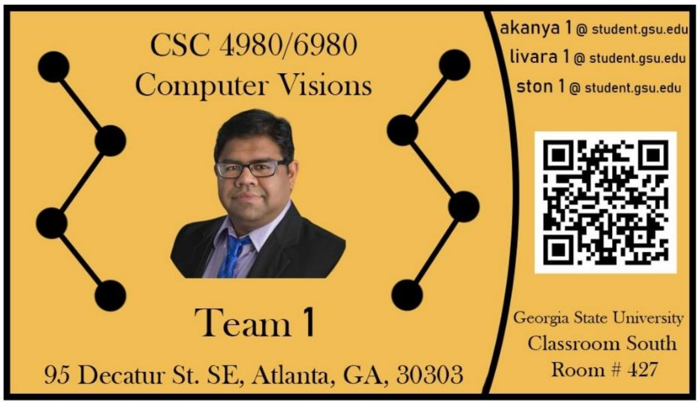


```
# Draw a bounding box around all contours
for c in contours:
  x, y, w, h = cv2.boundingRect(c)

    # Make sure contour area is large enough
  if (cv2.contourArea(c)) > 10:
    cv2.rectangle(with_contours,(x,y), (x+w,y+h), (255,0,0), 5)

cv2.imshow('All contours with bounding box', with_contours)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

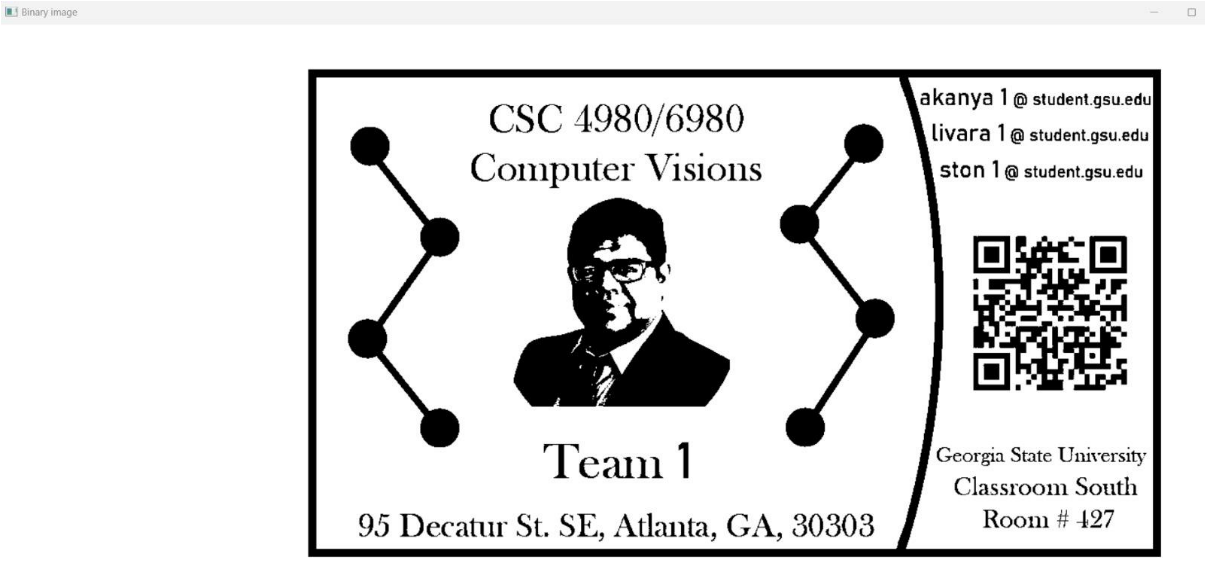
✓ 13.1s

```
Total number of contours detected: 364
```

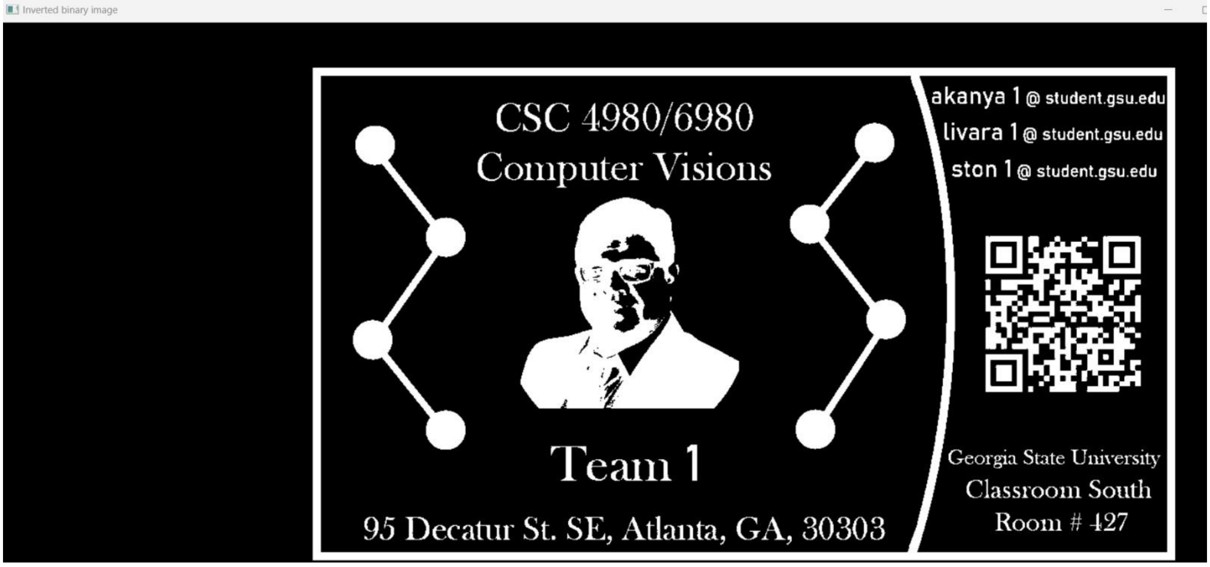
Image:



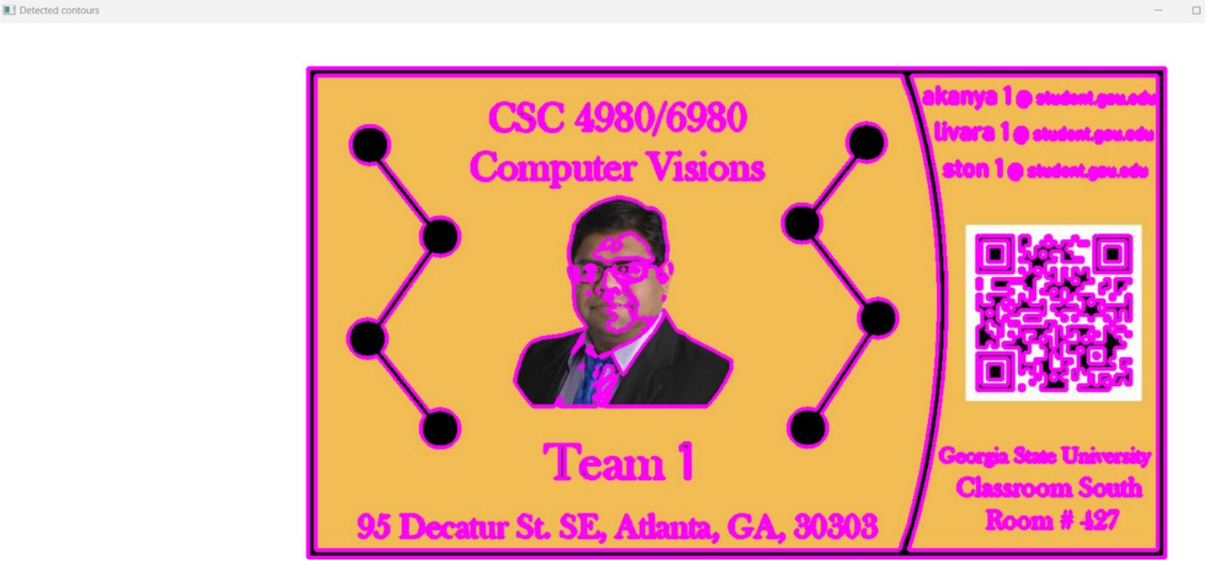
Binary Output:



Inverted Output:



Detect contours Output:


Detected contours
CSC 4980/6980
Computer Visions
Team 1
95 Decatur St. SE, Atlanta, GA, 30303
akanya 1 @ student.gsu.edu
livara 1 @ student.gsu.edu
ston 1 @ student.gsu.edu
Georgia State University
Classroom South
Room # 427