CSC420 Assignment4

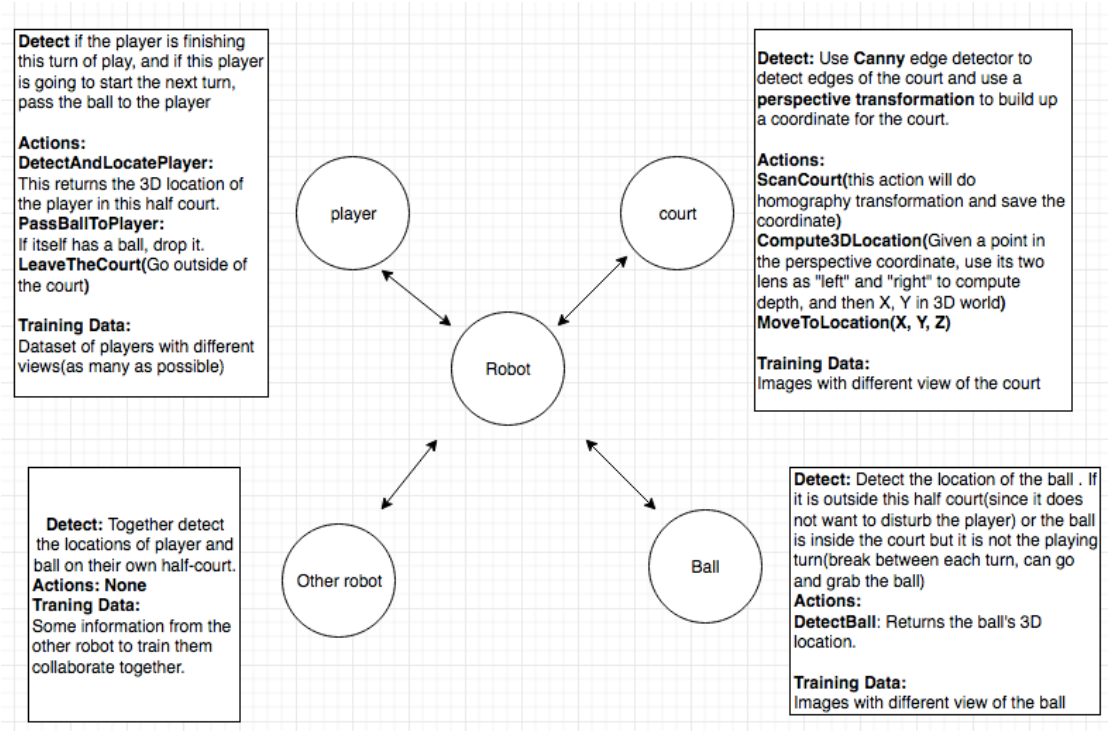
Name: Yilin Qu

Student number: 1002995734

Utorid: quyilin

Question1

(a)



Challenges:

- Two robots may not collaborate well, i.e., they might enter each other's half court
- It needs to avoid enter the court while the players are in game, so it is important but hard to know when the game is going on. (possible solution: Introduce a variable "Is_playing" to represent whether the game is on).
- It is also important to know who will serve on the next turn, so that the corresponding robot can pass the ball to that player while the other robot should wait.
- When delivering the ball to players, it might cause collision between robot and players.

(b)

Assumption:

- One robot only grab and pass the ball to the player on its own side.
- There are two ball-supplies on each sides of the court, labeled **supply1, supply2** in case that the robot grab no ball on its own half court but this player is going to serve.
- The supplies are to **deal with a special case:**
- If this robot grabbed the ball but the other player is going to serve: send the ball to supply
- If this robot did not grab the ball but this player is going to serve, it should grab a ball from supply and send it to the player.

Functions for robot class:

- **moveToLocation(X, Y, Z)**
- **takeImage(X, Y, Z):** take a stereo image with camera located at (X, Y, Z)
- **wait():** simply do nothing and wait to its next move instruction
- **faceLocation(X, Y, Z)**
- **grabBall(X, Y, Z)**
- **victoryDanceAtLocation(X, Y, Z)**
- **ScanCourt(cannyEdgeDetection, calculateCenterOfMass):** Scan the whole court, take stereo images, return the bounding boxes and their centers of mass, and label them with either "ball", "player" or "supply".
- **compute3dLocation(object):** given the coordinate of the object(player, ball or supply), compute the 3D location as in Question2
- **DetectBall():** following **ScanCourt**, detect the ball and return the coordinate location of the ball. Use either left or right image captured by its lens to locate the ball's coordinate.
- **DetectPlayer():** similar as DetectBall, but this time it will focus on the player.
- **PassBallToPlayer():** Drop the ball at its current location, usually following after it "moveToLocation(Player's location)"
- **LeaveTheCourt():** Find the shortest path from its current location to one of the edge of this half court but it should avoid the player, otherwise the player might be angry and kick the robot out!

GameStates: since we need some representation of the game, suppose:

- **ballInThisCourt:** returns if ball is inside this half court
- **Gameover:** return if this game is totally over
- **isPlaying:** returns if the players are gaming on the court
- **otherSide:** returns the other side's direction, opposite to its own.
- **thisPlayerServing:** returns if the player on this half court is going to serve on the next point.

Flowchart: (notice the cases are more detailed in section (c))

(c)

Some algorithm's descriptions and input arguments are listed in section 2(b)

```
import CNN
import robot
import gameState as game


def system():
    while not game.GameOver:
        while game.isPlaying:
            robot.wait()
        robot.scanCourt()
        objects = robot.detect(CNN)
        location = robot.compute3dLocation(objects["ball"])
        if game.ballInsideThisCourt:
            robot.moveToLocation(location)
            robot.grabBall(location)
            if game.thisPlayerServing:
                # if this player is going to serve, perfect.
                # send the ball to the player
                # ------ scan detect compute and move to location
                robot.scanCourt()
                objects = robot.detect(CNN)
                location = robot.compute3dLocation(objects["player"])
                robot.moveToLocation(location)
                # ------ scan detect compute and move to location
                robot.leaveCourt()
                robot.faceLocation(game.otherSide)
                robot.victoryDanceAtLocation(robot.currentLocation)
                robot.wait()
            else:
                # grabbed the ball but player not serving, send it to supply
                # ------ scan detect compute and move to location
                robot.scanCourt()
                objects = robot.detect(CNN)
                location = robot.compute3dLocation(objects["supply"])
```

```
            robot.moveToLocation(location)
            # ------ scan detect compute and move to location
            robot.leaveCourt()
            robot.faceLocation(game.otherSide)
            robot.victoryDanceAtLocation(robot.currentLocation)
            robot.wait()
    else:
        if game.thisPlayerServing:
            # the player is going to serve but no ball detected
            # go to the supply and grab a ball
            # send the ball to player and leave the court
            robot.scanCourt()
            objects = robot.detect(CNN)
            location = robot.compute3dLocation(objects["supply"])
            robot.moveToLocation(location)
            robot.grabBall(location)

            robot.scanCourt()
            objects = robot.detect(CNN)
            location = robot.compute3dLocation(objects["player"])
            robot.moveToLocation(location)
            robot.grabBall(location)

            robot.leaveCourt()
            robot.faceLocation(game.otherSide)
            robot.victoryDanceAtLocation(robot.currentLocation)
            robot.wait()
        else:
            # ball is not in this half court, just wait
            robot.wait()
```

(d)

Functions need to modify:

Since it lost one of the lens, need to modify “takeImage” and “compute3dLocation” since they first take stereo images and use each pair of them to compute 3d location of an object. The modification is, before the robot do compute3dLocation, it should add one action called “horizontalRapidMove”, which will let the robot to move a short distance(as baseline) along horizontal line very quickly, at the beginning and the end of this movement it will take two stereo images(like the parallel stereo cameras) as the “left image” and “right image” before. Finally the robot will compute 3d locations based on this pair of new images. However there might be an error if the whole scene is dynamic, but we only focus on when the ball is still and able to be picked up, and when the player is standing still and requesting a ball. Under this condition the new robot system should work fine.

Why is it important:

The robot can detect and locate object in the real world’s 3d coordinate based on the stereo images on both left and right. If it lost one of the lens, we will only have one stereo image each time, so that it is not sufficient to calculate 3d location of objects since it requires both “left” and “right” images. These changes above will present as in a very short time the robot will get two stereo images from left-to-right locations, just as with two lens on its left and right, therefore it will be able to continue its calculation on 3d locations.

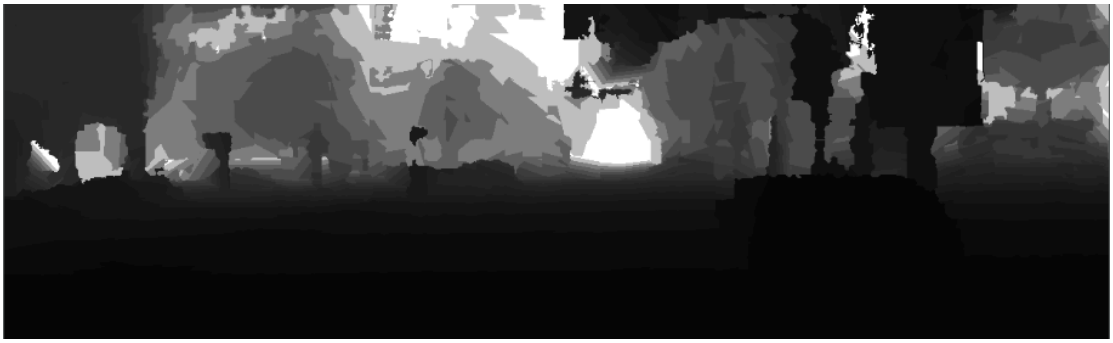
Conditions that make it still not possible to work:

As described before, if the scene is dynamic, i.e., if the player is always walking randomly and requiring the ball at the same time, the robot might compute a wrong 3d location since the two stereo images do not represent the same scene. Same issue will occur if the ball is always bouncing around for some reasons.

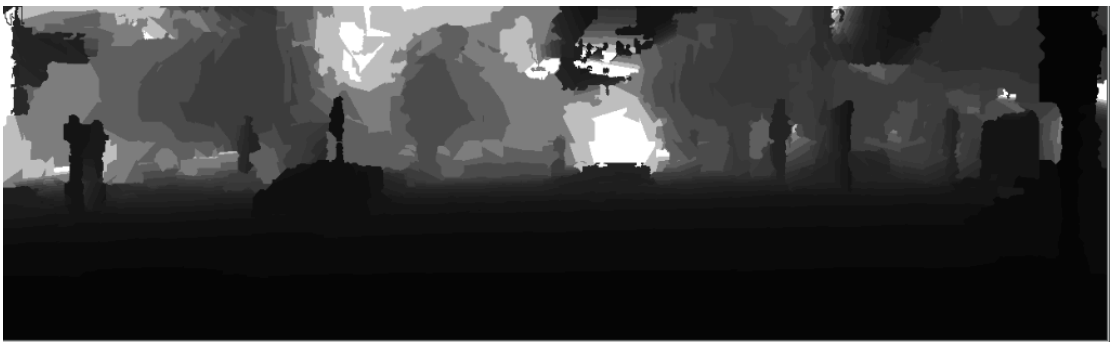
Question2

(a)

For 004945.png:



For 004964.png:



For 005002.png:

Code to compute depth Z:

```
def q2_a():
    # first image
    D1 = cv.imread("data/test/results/004945_left_disparity.png")
    f1 = 721.537700
    px1 = 609.559300
    py1 = 172.854000
    baseline1 = 0.5327119288 * 1000


    result1 = np.zeros(D1.shape)
    result1 = np.divide((f1 * baseline1), D1) / 1000


    cv.imwrite("4945_result.png", result1)
```

```
    # second image
    D2 = cv.imread("data/test/results/004964_left_disparity.png")
    f2 = 721.537700
    px2 = 609.559300
    py2 = 172.854000
    baseline2 = 0.5327119288 * 1000

    result2 = np.zeros(D1.shape)
    result2 = np.divide((f2 * baseline2), D2) / 1000

    cv.imwrite("4964_result.png", result2)

    # third image
    D3 = cv.imread("data/test/results/005002_left_disparity.png")
    f3 = 721.537700
    px3 = 609.559300
    py3 = 172.854000
    baseline3 = 0.5327119288 * 1000

    result3 = np.zeros(D3.shape)
    result3 = np.divide((f3 * baseline3), D3) / 1000

    cv.imwrite("5002_result.png", result3)
    return D1, D2, D3
```

(b)

Using a threshold = 0.48

This threshold will keep at least one traffic light per image, since it can be verified visually.

Code for thresholding detections:

```
result = {}
        result["num_detections"] = 0
        result["detection_scores"] = []
        result["detection_classes"] = []
        result["detection_boxes"] = []
        threshold = 0.48
        for i in range(output_dict["num_detections"]):
                if output_dict["detection_scores"][i] >= threshold:
```

```
                result["num_detections"] += 1
                result["detection_scores"].append(output_dict["detection_scores"][i])
                result["detection_classes"].append(output_dict["detection_classes"][i])
                result["detection_boxes"].append([output_dict["detection_boxes"][i][j] for j in
range(len(output_dict["detection_boxes"][i]))])
```

Giving the following result

q2b_4945.txt:

```
7
[0.9491954, 0.903302, 0.7825848, 0.72892267, 0.66242313, 0.57823133, 0.49721998]
[3, 3, 10, 3, 1, 3, 1]
[[0.50004214, 0.6555561, 0.8872184, 0.8707886], [0.44908354, 0.584361, 0.5227398,
0.6364023], [0.010417335, 0.5026116, 0.09855365, 0.5190967], [0.46349224, 0.42532513,
0.5317025, 0.47155157], [0.4670957, 0.35683563, 0.53998923, 0.3719673], [0.49316233,
0.23185101, 0.54320043, 0.26347476], [0.47287303, 0.39115834, 0.5437165, 0.40961748]]
```

q2b_4964.txt:

```
6
[0.88989836, 0.7733079, 0.7360068, 0.72747695, 0.65513057, 0.555762]
[3, 3, 3, 3, 3, 10]
[[0.47048816, 0.44235417, 0.5088938, 0.474377], [0.46128154, 0.5218015, 0.55010664,
0.58383596], [0.4213005, 0.7209805, 0.5365165, 0.80251265], [0.47370738, 0.49366155,
0.520934, 0.5202959], [0.5039439, 0.15278855, 0.5627457, 0.21511334], [0.28877726,
0.29553753, 0.3853735, 0.3158021]]
```

q2b_5002.txt:

```
4
[0.88686144, 0.8019987, 0.72281444, 0.50532067]
[3, 3, 3, 10]
[[0.46915835, 0.39268324, 0.59815925, 0.51287454], [0.49698436, 0.16895944,
0.70170236, 0.31567362], [0.41786578, 0.8332899, 0.6043134, 0.986494], [0.0924626,
0.814217, 0.18619645, 0.8461287]]
```

(c)

Preprocess the data:

The format here is a little bit messy due to the copy-paste operation in pycharm

```
def process_data(filename):
    d = {}
    fileIO = open(filename, "r")
    count = 0
    num_detection = int(fileIO.readline().strip())
    scores_list = [float(x) for x in fileIO.readline().strip("[]\n").split(",")]
    classes = [int(x) for x in fileIO.readline().strip("[]\n ").split(",")]
    boxes = fileIO.readline().strip("\n)")[1:-1].split("),")
    detection_boxes = []
    for item in boxes:
        item_list = item.strip()[1:].split(", ")
        coor_list = [float(x) for x in item_list]
        detection_boxes.append(coor_list)
    d["num_detections"] = num_detection
    d["detection_boxes"] = detection_boxes
    d["detection_scores"] = scores_list
    d["detection_classes"] = classes
    fileIO.close()
    return d

detection1, detection2, detection3 = process_data("data/test/q2b_4945.txt"),
process_data("data/test/q2b_4964.txt"), process_data("data/test/q2b_5002.txt")
img1 = cv.imread("data/test/left/004945.jpg")
img2 = cv.imread("data/test/left/004964.jpg")
img3 = cv.imread("data/test/left/005002.jpg")
```

Helper function for drawing(Using cv2.rectangle and cv2.putText):

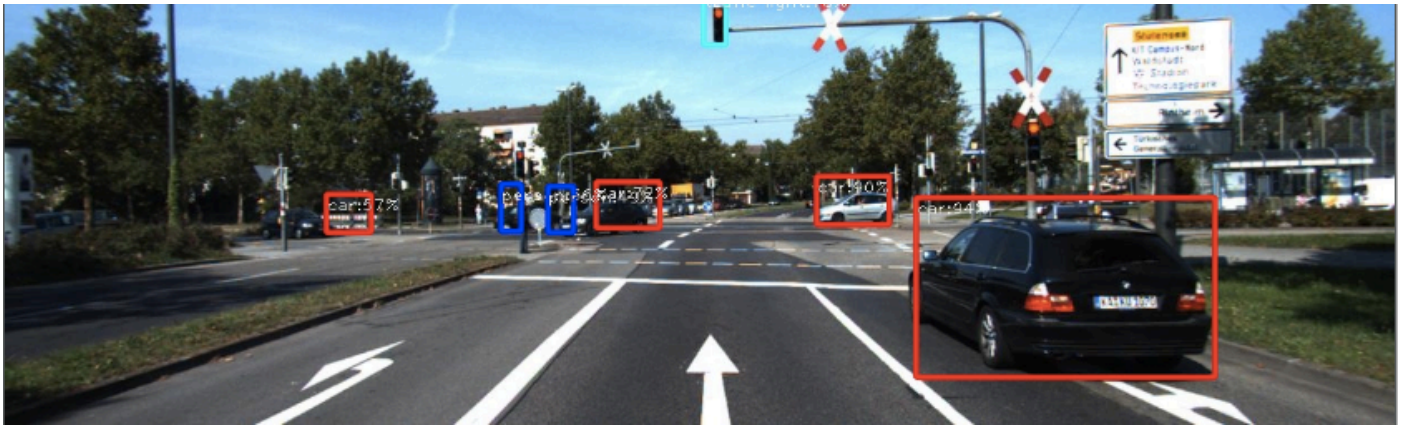
```
def draw(img, pos, classes, confidence):
    left = int(pos[0]*img.shape[0])
    top = int(pos[1]*img.shape[1])
    right = int(pos[2]*img.shape[0])
```

```
    bot = int(pos[3]*img.shape[1])

    color = (0, 0, 0)
    text = ""

    if classes == 1:
        # person
        color = (255, 0, 0)
        text = "person:" + str(int(confidence*100)) + "%"
    if classes == 2:
        # bic
        color = (0, 255, 0)
        text = "bicycle:" + str(int(confidence*100)) + "%"
    if classes == 3:
        # car
        color = (0, 0, 255)
        text = "car:" + str(int(confidence*100)) + "%"
    if classes == 10:
        # traffic light
        color = (255, 255, 0)
        text = "trafic light:" + str(int(confidence*100)) + "%"
    cv.rectangle(img, (top, left-15), (bot, right), color=color, thickness=3)
    cv.putText(img, text, org=(top, left), color=(255, 255, 255), fontScale=1, fontFace=1)
```

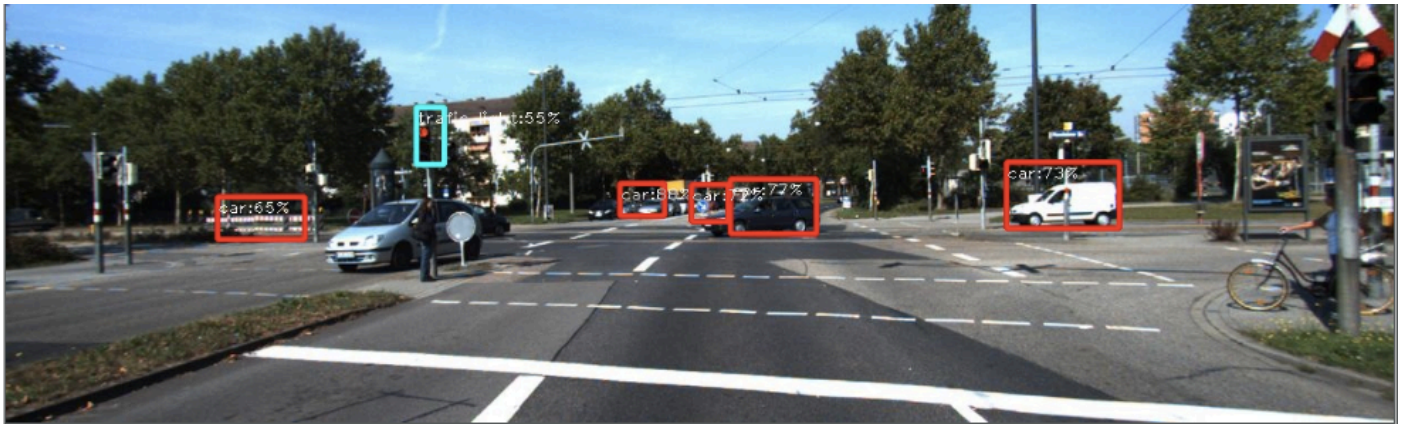
```
def q2_c():
    for i in range(detection1["num_detections"]):
        draw(img1, detection1["detection_boxes"][i], detection1["detection_classes"][i],
detection1["detection_scores"][i])
    cv.imwrite("q2_c_4945.jpg", img1)

    for i in range(detection2["num_detections"]):
        draw(img2, detection2["detection_boxes"][i], detection2["detection_classes"][i],
detection2["detection_scores"][i])
    cv.imwrite("q2_c_4964.jpg", img2)

    for i in range(detection3["num_detections"]):
```

```
        draw(img3, detection3["detection_boxes"][i], detection3["detection_classes"][i],
detection3["detection_scores"][i])
    cv.imwrite("q2_c_5002.jpg", img3)
```

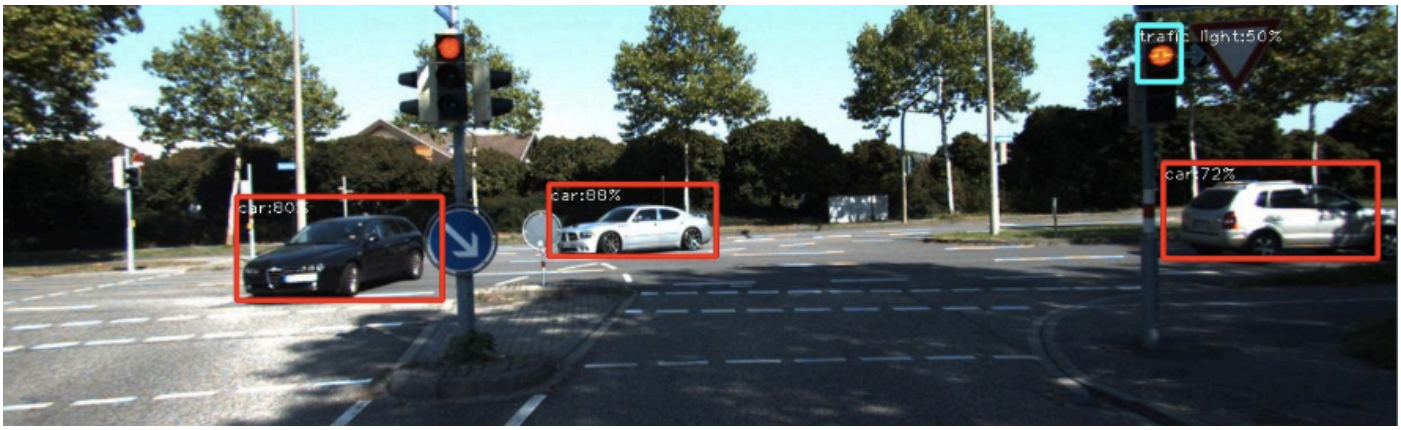
Visualization for 004945.png:



Visualization for 004964.png:



Visualization for 005002.png:



(d)

First I computed the depth information as Z, then I used Z to compute the corresponding X and Y with the following formula:

$$X = (x - px) * z / f$$

$$Y = (y - py) * z / f$$

Then for each bounding box, I iterated over all the pixels inside it and compute their 3D locations. To come up with the center of mass, I kept sorted all their locations with respect to their depths (Z value) and take the median of them to be the center of mass.

Algorithm below:

```
def calculate_center_of_mass(detection, D):
    # q2_d()
    centers = []
    for i in range(detection["num_detections"]):
        pos = detection["detection_boxes"][i]
        left = int(pos[0] * img1.shape[0])
        top = int(pos[1] * img1.shape[1])
        right = int(pos[2] * img1.shape[0])
        bot = int(pos[3] * img1.shape[1])
        z = np.divide((f * baseline), D)
        M, N = abs(left - right), abs(top - bot)
        Zs = []
        for row in range(left, right):
            for col in range(top, bot):
                X = np.divide((col-px)*z[row, col, 0], f)
                Y = np.divide((row-py)*z[row, col, 0], f)
                Zs.append((z[row, col, 0], X, Y))
        Zs.sort()
        median = Zs[len(Zs)//2]
        centers.append((median[1], median[2], median[0]))
    print(centers)
    return center
```

(e)

Function that calculates 3D locations for all pixels:

```
def calculate_3D_location(img, D):
    X, Y, Z = np.zeros(img.shape), np.zeros(img.shape), np.zeros(img.shape)
    z = np.divide((f * baseline), D)
    for row in range(img.shape[0]):
        for col in range(img.shape[1]):
            X[row, col] = np.divide((col - px) * z[row, col, 0], f)
            Y[row, col] = np.divide((row - py) * z[row, col, 0], f)
            Z[row, col] = z[row, col, 0]
    return X, Y, Z
```

Q2 (e) function:

```
def q2_e(detection, D, img, out):
    X, Y, Z = calculate_3D_location(img, D)
    center = calculate_center_of_mass(detection, D)
    segment = np.zeros((img.shape[0], img.shape[1], 3))
    for i in range(detection["num_detections"]):
        pos = detection["detection_boxes"][i]
        left = int(np.round(pos[0] * img.shape[0]))
        top = int(np.round(pos[1] * img.shape[1]))
        right = int(np.round(pos[2] * img.shape[0]))
        bot = int(np.round(pos[3] * img.shape[1]))
        z = np.true_divide((f * baseline), D)
        z[np.where(z == np.inf)] = 5000
        for row in range(left, right):
            for col in range(top, bot):
                color = ((i+1)*30, 25*(i+1), 125)
                if (
                            (center[i][0] - X[row, col, 0])**2 +
                            (center[i][1] - Y[row, col, 0])**2 +
                            (center[i][2] - z[row, col, 0])**2
                )**0.5 <= 3:
                    segment[row, col] = color
    cv.imwrite(out, segment)
```

Visualization for 004945.png:

Visualization for 004964.png:

Visualization for 005002.png:

(f)

```
def q2_f(detection, D):
    center = calculate_center_of_mass(detection, D)
    dis_dict = {"car": [np.inf, -1], "person": [np.inf, -1], "bicycle": [np.inf, -1]}
    num_cars, num_bics, num_people = 0, 0, 0
    is_traffic = False
    for i in range(detection["num_detections"]):
```

```
        X, Y, Z = center[i][0], center[i][1], center[i][2]
        distance = (X**2 + Y**2 + Z**2)**0.5
        if detection["detection_classes"][i] == 1:
            num_people += 1
            if distance < dis_dict["person"][0]:
                dis_dict["person"][0] = distance
                dis_dict["person"][1] = i
        if detection["detection_classes"][i] == 2:
            num_bics += 1
            if distance < dis_dict["bicycle"][0]:
                dis_dict["bicycle"][0] = distance
                dis_dict["bicycle"][1] = i
        if detection["detection_classes"][i] == 3:
            num_cars += 1
            if distance < dis_dict["car"][0]:
                dis_dict["car"][0] = distance
                dis_dict["car"][1] = i
        if detection["detection_classes"][i] == 10:
            label = "traffic light"
            dis_dict[label] = [distance, i]
            is_traffic = True
    print("There is(are) " + str(num_cars) + " car(s) nearby.")
    print("There is(are) " + str(num_people) + " persons nearby.")
    print("There is(are) " + str(num_bics) + " bicycle(s) nearby.")
    if is_traffic:
        print("Traffic light(s) near by~ Be careful!")
    if num_cars > 0:
        X = center[dis_dict["car"][1]][0]
        if X >= 0:
            text = "to your right."
        else:
            text = "to your left."
        print("The closest car is " + str(abs(X)) + " " + text)
        print("It is " + str(dis_dict["car"][0]) + " away from you.")
    if num_bics > 0:
        X = center[dis_dict["bicycle"][1]][0]
```

```
        if X >= 0:
            text = "to your right."
        else:
            text = "to your left."
        print("The closest bicycle is " + str(abs(X)) + " " + text)
        print("It is " + str(dis_dict["bicycle"][0]) + " away from you.")
    if num_people > 0:
        X = center[dis_dict["person"][1]][0]
        if X >= 0:
            text = "to your right."
        else:
            text = "to your left."
        print("The closest person is " + str(abs(X)) + " " + text)
        print("It is " + str(dis_dict["person"][0]) + " away from you.")
```

The output information given is:

----------Info for image 004945----------

There is(are) 4 car(s) nearby.

There is(are) 2 persons nearby.

There is(are) 0 bicycle(s) nearby.

Traffic light(s) near by~ Be careful!

The closest car is 3.2208034596796633 meters to your right.

It is 7.950685342912583 meters away from you.

The closest person is 5.063635125179803 meters to your left.

It is 35.31199525446571 meters away from you.

----------Info for image 004945----------

----------Info for image 004964----------

There is(are) 5 car(s) nearby.

There is(are) 0 persons nearby.

There is(are) 0 bicycle(s) nearby.

Traffic light(s) near by~ Be careful!

The closest car is 3.1691855926383776 meters to your right.

It is 35.08974792215562 meters away from you.

----------Info for image 004964----------

----------Info for image 005002----------

There is(are) 3 car(s) nearby.

There is(are) 0 persons nearby.

There is(are) 0 bicycle(s) nearby.

Traffic light(s) near by~ Be careful!

The closest car is 6.360564892440744 meters to your left.

It is 17.277319841616652 meters away from you.

----------Info for image 005002----------