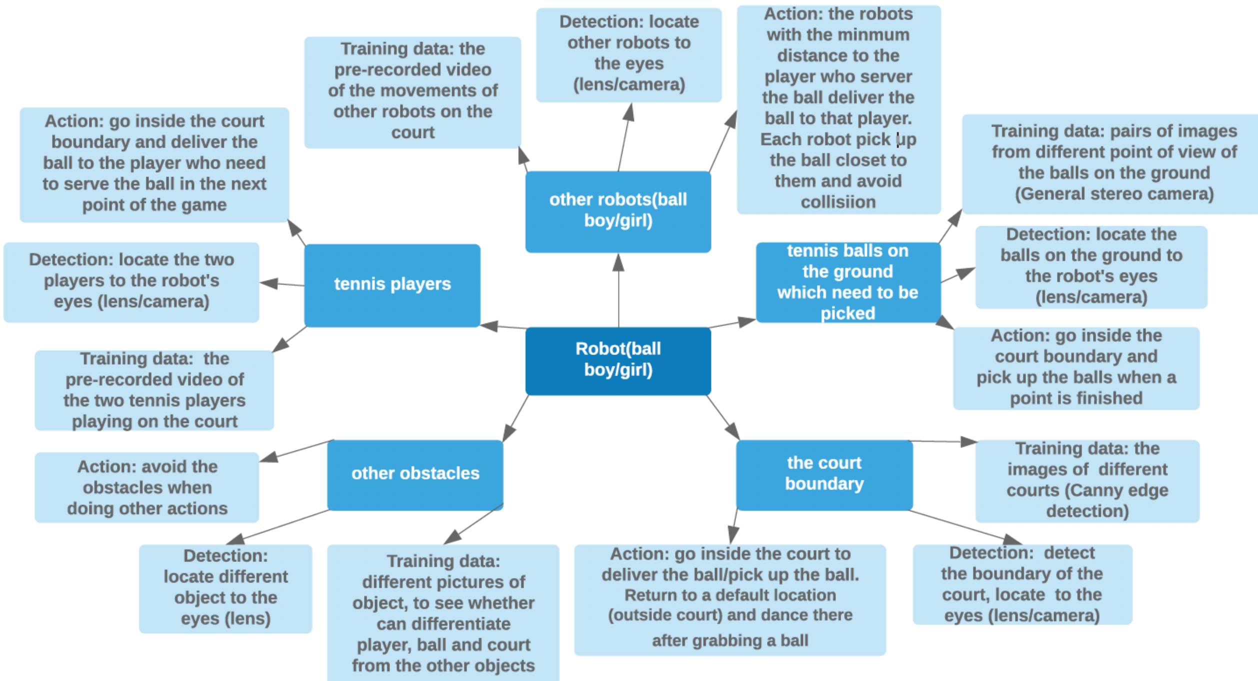
Name: Yuhan Shao
UTORid: shaoyuha
Student Number: 1002281327

CSC420 Assignment4

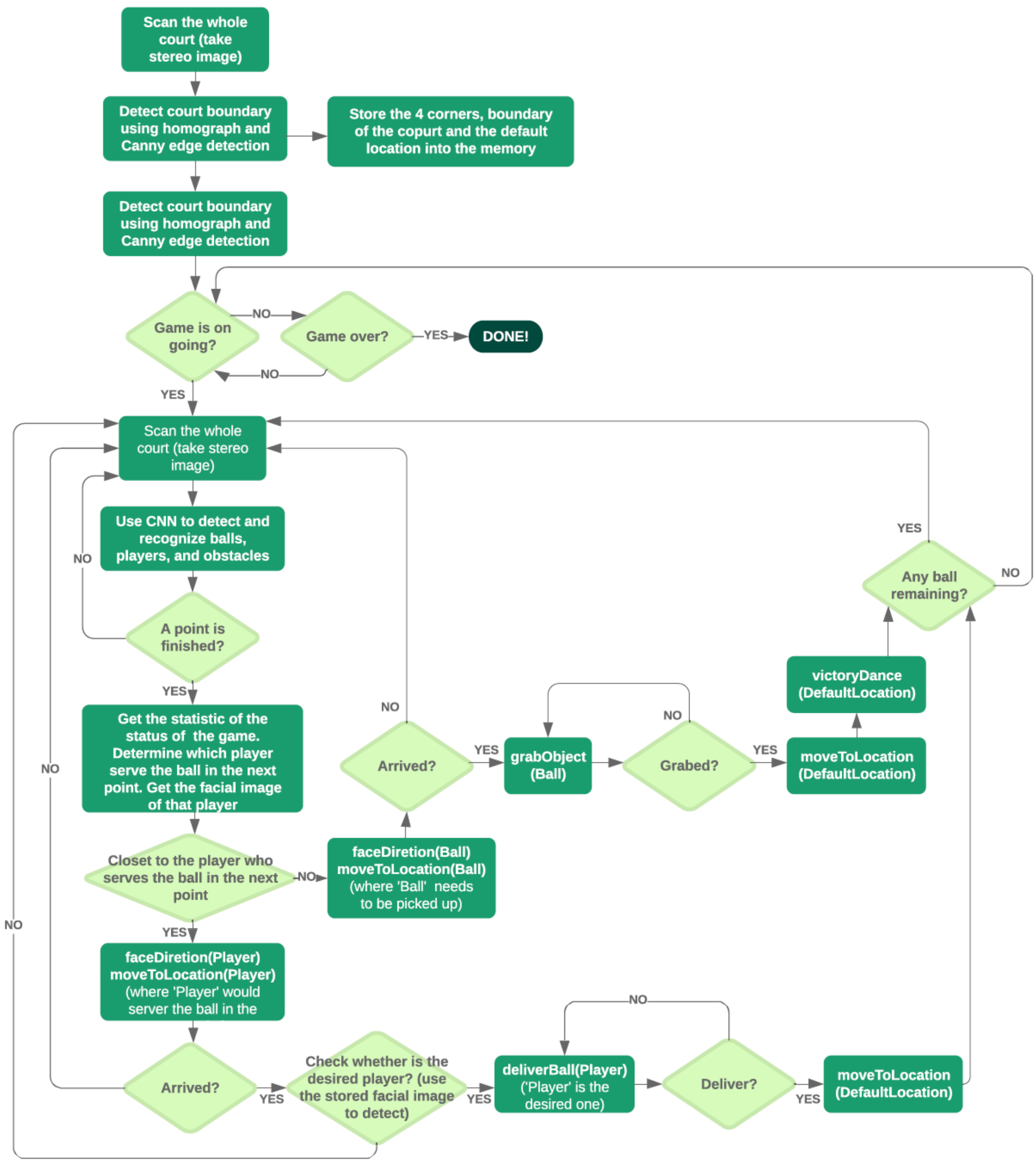
1. (a)



- **Challenges:**

- Other robots: when robots pick up the ball when a point is finished, robots need to cooperate with each other and do not collide with each other.
- Tennis ball on the ground which need to be picked: the robot needs to know the object on the ground is the ball that need to be picked when a point is finished.
- The court boundary: need to use Canny edge detection and homograph to determine the boundary.
- Other obstacles: differentiate other objects with balls, player, court and other robots.
- Tennis players: need to know which player serves the ball in the next point of the game in order to deliver the ball to him.

(b)



Scan the whole court (take stereo image)
Detect court boundary using homograph and Canny edge detection
Store the 4 corners, boundary of the copurt and the default location into the memory
Detect court boundary using homograph and Canny edge detection
Game is on going?
NO
Game over?
YES
DONE!
NO
YES
Scan the whole court (take stereo image)
Use CNN to detect and recognize balls, players, and obstacles
NO
A point is finished?
YES
Get the statistic of the status of the game. Determine which player serve the ball in the next point. Get the facial image of that player
Closet to the player who serves the ball in the next point
NO
faceDiretion(Ball) moveToLocation(Ball) (where 'Ball' needs to be picked up)
Arrived?
NO
YES
grabObject (Ball)
Grabed?
NO
YES
moveToLocation (DefaultLocation)
victoryDance (DefaultLocation)
Any ball remaining?
YES
NO
YES
faceDiretion(Player) moveToLocation(Player) (where 'Player' would server the ball in the
Arrived?
NO
YES
Check whether is the desired player? (use the stored facial image to detect)
NO
YES
deliverBall(Player) ('Player' is the desired one)
Deliver?
NO
YES
moveToLocation (DefaultLocation)

(c) **CNN ALGO Idea**: https://arxiv.org/pdf/1506.01195.pdf

```
# helper function
CNNTraining(dataSet):
    for i in range(size(dataSet)):
        matches = CNN(dataSet.image[i])
        while matches != dataSet.matches:
            backPropagation()
```

```
# consider the default methods: faceDirection(X,Y,Z), moveToLocation(X,Y,Z),
# grabObject(X,Y,Z), victoryDanceAtLocation(X,Y,Z) as the methods of robot class
init robot class
init game class

robot.ballFeatures = CNNTraining(dataSetOfTennisBall)
robot.playersFeatures = CNNTraining(dataSetOfPlayers)
robot.courtBoundary = cannyEdgeDetection(currCourtImage)
```

```
TennisGame(robot, game):
    """
    Pass in the initialized robot and game objects.
    Play the tennis match.
    """
    while !game.gameOver:
        robot.takeStereoImage()
        balls, players, obstacles = robot.recognizeBallsLocation(),
                                    robot.recognizePlayersLocation(),
                                    robot.recognizeObsatclesLocation()
        numRemainingBalls = len(balls)

        if game.pointFinished && numRemainingBalls != 0:
            robot.getCurrentGameStatistic()
            # get the position of the player who serve the ball in the next point
            playerServeBall = robot.getPlayerServeBall()

            # determine whether the current robot is closest to playerServeBall among all other robots
            if robot.closestToPlayer(playerServeBall, robot.otherRobots)
                robot.faceDirection(playerServeBall)
                robot.moveToLocation(playerServeBall)
                # use the facial detection to check
                if robot.arrive(playerServeBall) && robot.checkCorrectPlayer():
                    if robot.deliverBall(playerServeBall):
                        # defaultLocation is somewhere outside of the court, i.e.: the start position and dance position of the robot
                        # here robot not grab the ball, therefore, not dance
                        robot.faceDirection(robot.defaultLocation)
                        robot.moveToLocation(robot.defaultLocation)

            else:
                # the robot grab one ball from the balls on the ground and dance
                ball = robot.ballToGrab(balls)
                robot.faceDirection(ball)
                robot.moveToLocation(ball)
                if robot.arrive(ball):
                    numRemainingBalls--
                    if robot.grabObject(ball):
                        robot.faceDirection(robot.defaultLocation)
                        robot.moveToLocation(robot.defaultLocation)
                        robot.victoryDanceAtLocation(robot.defaultLocation)
```

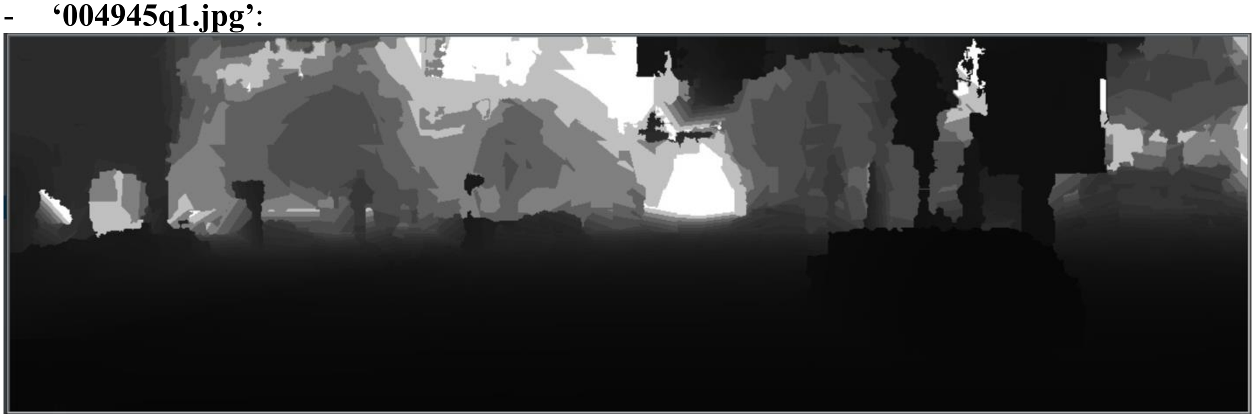
(d)

- **Need to modify (software):** the general flow chart does not need to change but: Need to change ‘robot.takeStereoImage()’ method of the robot class.
- **Why those changes are important to solve this issue:** Lose one of its lens, the robot cannot have the two (from 2 lens) stereo images from different point of views simultaneously. Instead, only use one eye (len) to take an image from different point of view sequentially. (i.e.: robot needs to change the face direction in order to change the point of view, and then take an image)
- **Conditions that the robot may still be unable to complete its task:** Problem comes when the object **moves** (especially when moving fast) during the time when robot changes the face direction and take the images sequentially. The robot would calculate wrong depth/distance to the object from the stereo images it takes. Consequently, robot may crash into the moving objects (e.g.: moving balls, players and obstacles)

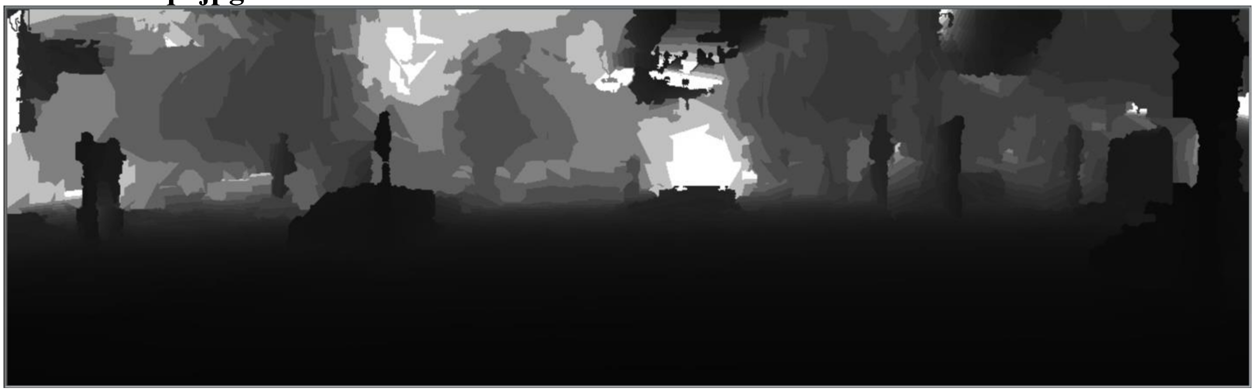
2. (a) **Code:**

```
import cv2 as cv
import numpy as np

# _ALL CALIB.txt from first 3 images
calib = {'f': 721.537700, 'px': 609.559300, 'py': 172.854000, 'baseline':
0.5327119288}

def calculate_depth(imgs, calib):
    disparity = cv.imread(imgs)
    depth = np.true_divide(calib['f'] * calib['baseline'], disparity)
    return depth


if __name__ == '__main__':
    cv.imwrite('004945q1.jpg', calculate_depth('004945_left_disparity.png', calib))
    cv.imwrite('004964q1.jpg', calculate_depth('004964_left_disparity.png', calib))
    cv.imwrite('005002q1.jpg', calculate_depth('005002_left_disparity.png', calib))
```

- **‘004945q1.jpg’**:



- **‘004964q1.jpg’:**



- **‘005002q1.jpg’:**

(b)

- Reuse the output ‘*output_dict*’ from the function ‘*run_inference_for_single_image(image, graph)*’
- Eliminate the data from the ‘*output_dict*’ according to the ‘*threshold*’ variable. Adjust the ‘*threshold*’ variable to order to keep approximate 3-5 ‘*num_detections*’ **(‘*threshold*’ = 0.5)**
- Use the following function to eliminate the dictionary

```
import numpy as np

threshold = 0.5

def eliminate_dict(output_dict):
    scores = output_dict['detection_scores']

    index = []
    num_detection = 0
    for i in range(output_dict['num_detections']):
        score = scores[i]
        if score > threshold:
            index.append(i)
            num_detection += 1

    detection_boxes = []
    detection_scores = []
    detection_classes = []
    eliminated_dict = {}

    for i in range(len(index)):
        detection_boxes.append(output_dict['detection_boxes'][index[i]])
        detection_scores.append(output_dict['detection_scores'][index[i]])
        detection_classes.append(output_dict['detection_classes'][index[i]])
```

```
    eliminated_dict['num_detection'] = num_detection
    eliminated_dict['detection_boxes'] = np.array(detection_boxes)
    eliminated_dict['detection_scores'] = np.array(detection_scores)
    eliminated_dict['detection_classes'] = np.array(detection_classes)

    return eliminated_dict
```

- Then, modify the main function in '*object_detection_tutorial.ipynb*' as below:

```
for image_path in TEST_IMAGE_PATHS:
  image = Image.open(image_path)
  # the array based representation of the image will be used later in order to prepare the
  # result image with boxes and labels on it.
  image_np = load_image_into_numpy_array(image)
  # Expand dimensions since the model expects images to have shape: [1, None, None, 3]
  image_np_expanded = np.expand_dims(image_np, axis=0)
  # Actual detection.
  output_dict = run_inference_for_single_image(image_np, detection_graph)

  # Add the elimination dictionary code
  output_dict = eliminate_dict(output_dict)
  print(output_dict)

  # Visualization of the results of a detection.
  vis_util.visualize_boxes_and_labels_on_image_array(
      image_np,
      output_dict['detection_boxes'],
      output_dict['detection_classes'],
      output_dict['detection_scores'],
      category_index,
      instance_masks=output_dict.get('detection_masks'),
      use_normalized_coordinates=True,
      line_thickness=8)
  plt.figure(figsize=IMAGE_SIZE)
  plt.imshow(image_np)
```

- **The printed eliminated '*output_dict*' output would as below:**

```
{'num_detection': 6, 'detection_boxes': array([[0.50004214, 0.6555561 , 0.8872184 , 0.8707886
 ],
       [0.44908354, 0.584361  , 0.5227398 , 0.6364023 ],
       [0.01041733, 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]], dtype=float32), 'detection_scores':
 array([0.9491954 , 0.903302  , 0.7825848 , 0.72892267, 0.66242313,
       0.57823133], dtype=float32), 'detection_classes': array([ 3,  3, 10,  3,  1,  3], dtyp
e=uint8)}
{'num_detection': 6, 'detection_boxes': array([[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 ]], dtype=float32), 'detection_scores':
 array([0.88989836, 0.7733079 , 0.7360068 , 0.72747695, 0.65513057,
       0.555762  ], dtype=float32), 'detection_classes': array([ 3,  3,  3,  3,  3, 10], dtyp
e=uint8)}
{'num_detection': 4, 'detection_boxes': array([[0.46915835, 0.39268324, 0.59815925, 0.5128745
4],
       [0.49698436, 0.16895944, 0.70170236, 0.31567362],
       [0.41786578, 0.8332899 , 0.6043134 , 0.986494  ],
       [0.0924626 , 0.814217  , 0.18619645, 0.8461287 ]], dtype=float32), 'detection_scores':
 array([0.88686144, 0.8019987 , 0.72281444, 0.50532067], dtype=float32), 'detection_classes':
 array([ 3,  3,  3, 10], dtype=uint8)}
```

(c)

- **Reuse the eliminated dictionary from the previous question.**

→ Store the dictionary data in the text file as the following format:

- datafile_dict1.txt

```
6
0.50004214, 0.6555561 , 0.8872184 , 0.8707886
0.44908354, 0.584361  , 0.5227398 , 0.6364023
0.01041733, 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
0.9491954
0.903302
0.7825848
0.72892267
0.66242313
0.57823133
3
3
10
3
1
3
```

- datafile_dict2.txt

```
6
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
0.88989836
0.7733079
0.7360068
0.72747695
0.65513057
0.555762
3
3
3
3
3
10
```

- datafile_dict3.txt

```
4
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
0.88686144
0.8019987
0.72281444
0.50532067
3
3
3
10
```

→ Then, read the data from the text file and store it as dictionary:

```
def read_data(filename):
    img_dict = {}
    file = open(filename, 'r')

    num_detection = int(file.readline())
    img_dict['num_detection'] = num_detection
    img_dict['detection_boxes'] = []
    img_dict['detection_scores'] = []
    img_dict['detection_classes'] = []

    # read box
    for i in range(num_detection):
        line_box = file.readline()
        box = line_box.split (',')
        for j in range(4):
            box[j] = float(box[j])
        img_dict['detection_boxes'].append(box)

    # read score
    for i in range(num_detection):
        line_score = file.readline()
        img_dict['detection_scores'].append(float(line_score))

    # read type
    for i in range(num_detection):
        line_type = file.readline()
        img_dict['detection_classes'].append(int(line_type))

    return img_dict


if __name__ == '__main__':
    img1_dict = read_data('datafile_dict1.txt')
    print(img1_dict)

    img2_dict = read_data('datafile_dict2.txt')
    print(img2_dict)

    img3_dict = read_data('datafile_dict3.txt')
    print(img3_dict)
```

■ **Output:**

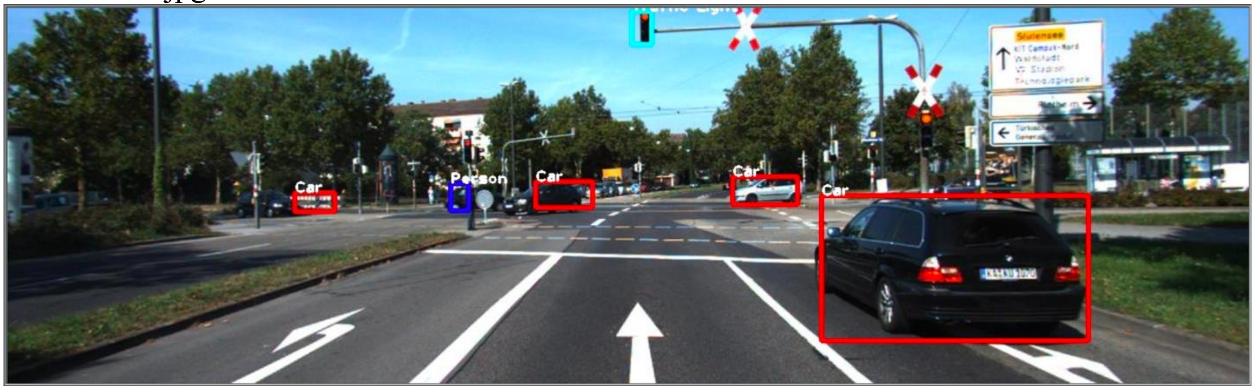
```
{'num_detection': 6, 'detection_boxes': [[0.50004214, 0.6555561, 0.8872184, 0.8707886], [0.44908354, 0.584361, 0.5227398, 0.6364023], [0.01041733, 0.5026116, 0.09855365, 0.5190967], [0.46349224, 0.42532513, 0.5317025, 0.47155157], [0.4670957, 0.35683563, 0.53998923, 0.3719673], [0.49316233, 0.23185101, 0.54320043, 0.26347476]], 'detection_scores': [0.9491954, 0.903302, 0.7825848, 0.72892267, 0.66242313, 0.57823133], 'detection_classes': [3, 3, 10, 3, 1, 3]}

{'num_detection': 6, 'detection_boxes': [[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]], 'detection_scores': [0.88989836, 0.7733079, 0.7360068, 0.72747695, 0.65513057, 0.555762], 'detection_classes': [3, 3, 3, 3, 3, 10]}

{'num_detection': 4, 'detection_boxes': [[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]], 'detection_scores': [0.88686144, 0.8019987, 0.72281444, 0.50532067], 'detection_classes': [3, 3, 3, 10]}
```

→ In the following question's code, consider the dictionary that we constructed from the data in the text file as the global variable. (i.e.: *img1_dict*, *img2_dict*, *img3_dict***)**

→ Now, consider the following code:

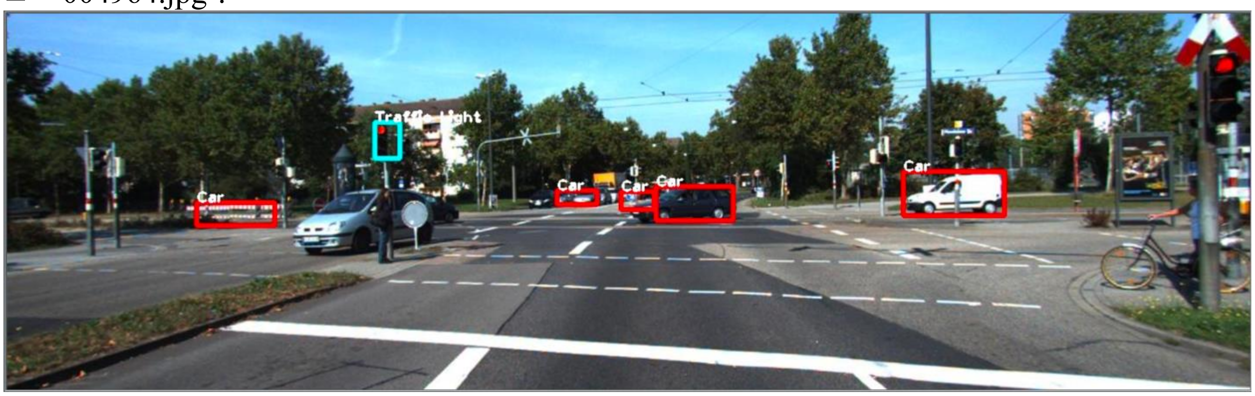
```
import cv2 as cv


img1_dict = {'num_detection': 6, 'detection_boxes': [[0.50004214, 0.6555561 , 0.8872184 , 0.8707886 ],
       [0.44908354, 0.584361  , 0.5227398 , 0.6364023 ],
       [0.01041733, 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]],
        'detection_scores': [0.9491954 , 0.903302  , 0.7825848 , 0.72892267, 0.66242313, 0.57823133],
        'detection_classes': [ 3,  3, 10,  3,  1,  3]}
img2_dict = {'num_detection': 6, 'detection_boxes': [[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 ]],
        'detection_scores': [0.88989836, 0.7733079 , 0.7360068 , 0.72747695, 0.65513057, 0.555762  ],
        'detection_classes': [ 3,  3,  3,  3,  3, 10]}
img3_dict = {'num_detection': 4, 'detection_boxes': [[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 ]],
        'detection_scores': [0.88686144, 0.8019987 , 0.72281444, 0.50532067],
        'detection_classes': [ 3,  3,  3, 10]}


def visualize(img, img_dict):
    img = cv.imread(img)
    height, width = img.shape[0], img.shape[1]
    for i in range(img_dict['num_detection']):
        box = img_dict['detection_boxes'][i]
        pt1 = (int(box[1] * width), int(box[0] * height))
        pt2 = (int(box[3] * width), int(box[2] * height))
        classes = img_dict['detection_classes'][i]

        if classes == 1:
            # person, blue (b, g, r)
            cv.rectangle(img, pt1, pt2, color=(255, 0, 0), thickness=3)
            cv.putText(img, 'Person', org=pt1, color=(255, 255, 255), fontScale=1, fontFace=1,
thickness=2)
        elif classes == 2:
            # bicycle, green
            cv.rectangle(img, pt1, pt2, color=(0, 255, 0), thickness=3)
            cv.putText ( img, 'Bicycle', org=pt1, color=(255, 255, 255), fontScale=1, fontFace=1,
thickness=2)
        elif classes == 3:
            # car, red
            cv.rectangle(img, pt1, pt2, color=(0, 0, 255), thickness=3)
            cv.putText(img, 'Car', org=pt1, color=(255, 255, 255), fontScale=1, fontFace=1,
thickness=2)
        elif classes == 10:
            # traffic_light, cyan
            cv.rectangle(img, pt1, pt2, color=(255, 255, 0), thickness=3)
            cv.putText(img, 'Traffic Light', org=pt1, color=(255, 255, 255), fontScale=1, fontFace=1,
thickness=2)
    return img


if __name__ == '__main__':
    cv.imwrite('004945q2_3.jpg', visualize('004945.jpg', img1_dict))
    cv.imwrite('004964q2_3.jpg', visualize('004964.jpg', img2_dict))
    cv.imwrite('005002q2_3.jpg', visualize('005002.jpg', img3_dict))
```

- **Output of the 3 images:**
- ‘004945.jpg’:

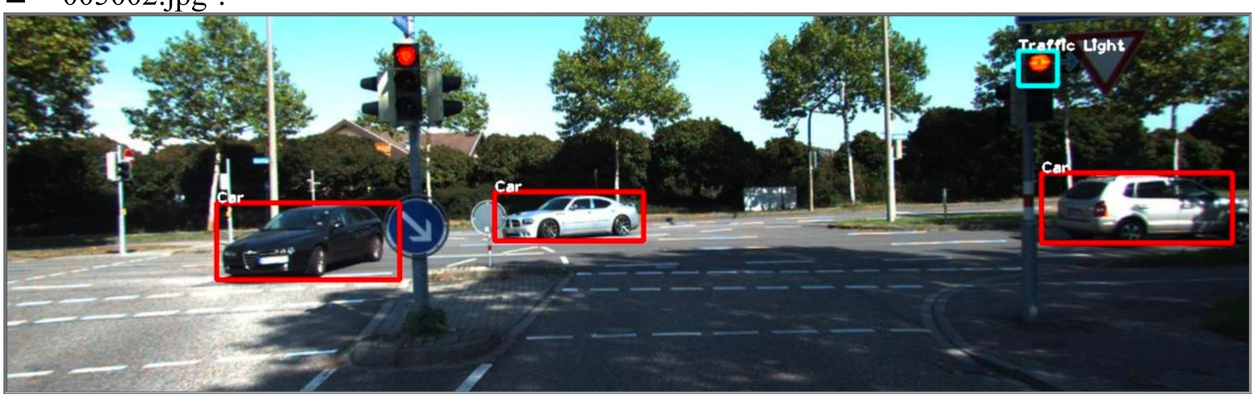


(**Note**: **‘traffic light’ text has been written on the top, but slightly out of image boundary**)

- ‘004964.jpg’:



- ‘005002.jpg’:



(d)

Compute the 3D location (center of mass) of each detected object:

- For the first 3 images, we have the dictionary of data for the objects on the image to be detected (i.e.: person, bicycle, car, traffic light).
- For each pixel inside the boundary box, first, calculate the (x, y, z) of each pixel in the real world, store as (z, x, y) in a list, then sort the list according to the z (depth) value. Then, get the **MEAN depth** (i.e.: the middle depth/z value of the sorted list of tuple) among all pixel, which is considered as the center of mass.

- **Code:**

```
import cv2 as cv
import numpy as np

img1_dict = {'num_detection': 6, 'detection_boxes': [[0.50004214, 0.6555561 , 0.8872184 ,
0.8707886 ],
       [0.44908354, 0.584361  , 0.5227398 , 0.6364023 ],
       [0.01041733, 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]],
        'detection_scores': [0.9491954 , 0.903302  , 0.7825848 , 0.72892267, 0.66242313,
0.57823133],
        'detection_classes': [ 3,  3, 10,  3,  1,  3]}
img2_dict = {'num_detection': 6, 'detection_boxes': [[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 ]],
        'detection_scores': [0.88989836, 0.7733079 , 0.7360068 , 0.72747695, 0.65513057,
0.555762  ],
        'detection_classes': [ 3,  3,  3,  3,  3, 10]}
img3_dict = {'num_detection': 4, 'detection_boxes': [[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 ]],
        'detection_scores': [0.88686144, 0.8019987 , 0.72281444, 0.50532067],
        'detection_classes': [ 3,  3,  3, 10]}
calib = {'f': 721.537700, 'px': 609.559300, 'py': 172.854000, 'baseline': 0.5327119288}

def calculate_depth(img):
    disparity = cv.imread(img)
    depth = np.true_divide(calib['f'] * calib['baseline'], disparity)
    return depth

def center(ZDepth, img, img_dict):
    img = cv.imread(img)
    height, width = img.shape[0], img.shape[1]
    box_depth = []

    for i in range(img_dict['num_detection']):
        depth = []
        box = img_dict['detection_boxes'][i]
        pt1 = (int(box[1] * width), int(box[0] * height))
        pt2 = (int(box[3] * width), int(box[2] * height))

        for row in range(pt1[1], pt2[1]):
            for column in range(pt1[0], pt2[0]):
                z = ZDepth[row, column, 0]
                x = np.true_divide((column - calib['px']) * z, calib['f'])
                y = np.true_divide((row - calib['py']) * z, calib['f'])
                depth.append((z, x, y))
        depth.sort()
        box_depth.append(depth[len(depth) // 2])  # (z, x, y)
    print(box_depth)
    return box_depth

if __name__ == '__main__':
    ZDepth1 = calculate_depth('004945_left_disparity.png')
    ZDepth2 = calculate_depth('004964_left_disparity.png' )
    ZDepth3 = calculate_depth('005002_left_disparity.png' )

    center(ZDepth1, '004945_left_disparity.png', img1_dict)
    center(ZDepth2, '004964_left_disparity.png', img2_dict)
    center(ZDepth3, '005002_left_disparity.png', img3_dict)
```

- **Output: (the '*box_depth*' of the 3 images, list of (z, x, y) tuple)**

```
[(7.252296978658787, 3.381622155166456, 0.4135653777812225), (48.046467483614464, 11.61583021977777, 1.075145850300599), (20.230091572048195, 0.8815176810432712, -4.0893771401681684), (42.70797109654619, -3.4069473025975374, 0.4229732714671992), (42.70797109654619, -9.089207876464204, 1.1924460575116436), (76.87434797378314, -33.51389828499557, 1.5071485889609586)]

[(76.87434797378314, -4.214742200995568, 0.4417247313609585), (34.9428854426287, 3.3144706641292867, 0.3460690402913448), (38.43717398689157, 18.934750087102216, 0.48721833008047927), (64.06195664481929, 1.1045515487703592, 0.7232452286674654), (54.91024855270224, -26.449855010768257, 2.522467084572113), (16.711814776909378, -5.293750676007732, -1.3168175652171827)]

[(25.624782657927714, -0.8722021381985228, 0.5734111201603195), (16.015489161204822, -6.338368562074076, 1.2462351647668664), (19.218586993445786, 13.888884052591107, 0.003888797080239642), (8.355907388454689, 4.950059994448307, -1.2606048760346782)]
```

(e)

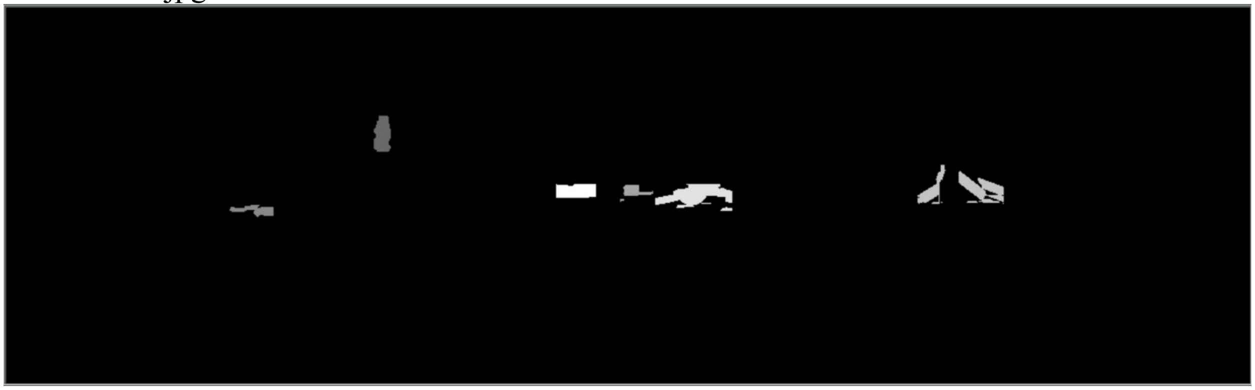
- **Code:**

```
import cv2 as cv
import numpy as np


img1_dict = {'num_detection': 6, 'detection_boxes': [[0.50004214, 0.6555561 , 0.8872184 ,
0.8707886 ],
       [0.44908354, 0.584361  , 0.5227398 , 0.6364023 ],
       [0.01041733, 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]],
        'detection_scores': [0.9491954 , 0.903302  , 0.7825848 , 0.72892267, 0.66242313,
0.57823133],
        'detection_classes': [ 3,  3, 10,  3,  1,  3]}
img2_dict = {'num_detection': 6, 'detection_boxes': [[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 ]],
        'detection_scores': [0.88989836, 0.7733079 , 0.7360068 , 0.72747695, 0.65513057,
0.555762  ],
        'detection_classes': [ 3,  3,  3,  3,  3, 10]}
img3_dict = {'num_detection': 4, 'detection_boxes': [[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 ]],
        'detection_scores': [0.88686144, 0.8019987 , 0.72281444, 0.50532067],
        'detection_classes': [ 3,  3,  3, 10]}
calib = {'f': 721.537700, 'px': 609.559300, 'py': 172.854000, 'baseline': 0.5327119288}


def calculate_depth(img):
    disparity = cv.imread(img)
    depth = np.true_divide(calib['f'] * calib['baseline'], disparity)
    return depth


def center(ZDepth, img, img_dict):
    img = cv.imread(img)
```

```
    height, width = img.shape[0], img.shape[1]
    box_depth = []

    for i in range(img_dict['num_detection']):
        depth = []
        box = img_dict['detection_boxes'][i]
        pt1 = (int(box[1] * width), int(box[0] * height))
        pt2 = (int(box[3] * width), int(box[2] * height))

        for row in range(pt1[1], pt2[1]):
            for column in range(pt1[0], pt2[0]):
                z = ZDepth[row, column, 0]
                x = np.true_divide((column - calib['px']) * z, calib['f'])
                y = np.true_divide((row - calib['py']) * z, calib['f'])
                depth.append((z, x, y))
        depth.sort()
        box_depth.append(depth[len(depth) // 2])  # (z, x, y)

    print(box_depth)
    return box_depth


def segmentation(img, ZDepth, box_depth, img_dict):
    img = cv.imread(img)
    height, width = img.shape[0], img.shape[1]
    segmentation = np.zeros((height, width, 3))
    num_detection = img_dict['num_detection']

    for i in range(num_detection):
        box = img_dict['detection_boxes'][i]
        pt1 = (int(box[1] * width), int(box[0] * height))
        pt2 = (int(box[3] * width), int(box[2] * height))

        for row in range(pt1[1], pt2[1]):
            for column in range(pt1[0], pt2[0]):
                z = ZDepth[row, column, 0]
                x = np.true_divide((column - calib['px']) * z, calib['f'] )
                y = np.true_divide((row - calib['py']) * z, calib['f'] )
                if (z - box_depth[i][0]) ** 2 + (x - box_depth[i][1]) ** 2 + (y -
box_depth[i][2]) ** 2 <= 9:
                    segmentation[row, column] = 255 – 30 * i

    return segmentation


if __name__ == '__main__':
    ZDepth1 = calculate_depth('004945_left_disparity.png')
    ZDepth2 = calculate_depth('004964_left_disparity.png' )
    ZDepth3 = calculate_depth('005002_left_disparity.png' )

    box_depth1 = center(ZDepth1, '004945_left_disparity.png', img1_dict)
    box_depth2 = center(ZDepth2, '004964_left_disparity.png', img2_dict)
    box_depth3 = center(ZDepth3, '005002_left_disparity.png', img3_dict)

    cv.imwrite('004945seg.jpg', segmentation('004945.jpg', ZDepth1, box_depth1, img1_dict))
    cv.imwrite('004964seg.jpg', segmentation('004964.jpg', ZDepth2, box_depth2, img2_dict))
    cv.imwrite('005002seg.jpg', segmentation('005002.jpg', ZDepth3, box_depth3, img3_dict))
```

- **Output:**
- ‘004945.jpg’:

- ‘004964.jpg’:



- ‘005002.jpg’:

(f)

- **Code:**

```
import cv2 as cv
import numpy as np


img1_dict = {'num_detection': 6, 'detection_boxes': [[0.50004214, 0.6555561 , 0.8872184 ,
0.8707886 ],
       [0.44908354, 0.584361  , 0.5227398 , 0.6364023 ],
       [0.01041733, 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]],
        'detection_scores': [0.9491954 , 0.903302  , 0.7825848 , 0.72892267, 0.66242313,
0.57823133],
```

```
        'detection_classes': [ 3,  3, 10,  3,  1,  3]}
img2_dict = {'num_detection': 6, 'detection_boxes': [[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 ]],
        'detection_scores': [0.88989836, 0.7733079 , 0.7360068 , 0.72747695, 0.65513057,
0.555762  ],
        'detection_classes': [ 3,  3,  3,  3,  3, 10]}
img3_dict = {'num_detection': 4, 'detection_boxes': [[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 ]],
        'detection_scores': [0.88686144, 0.8019987 , 0.72281444, 0.50532067],
        'detection_classes': [ 3,  3,  3, 10]}
calib = {'f': 721.537700, 'px': 609.559300, 'py': 172.854000, 'baseline': 0.5327119288}


def num_object(img_dict):
    count_person, count_bicycle, count_car, count_traffic_light = 0, 0, 0, 0
    type_dict = {}

    for i in range(img_dict['num_detection']):
        classes = img_dict['detection_classes'][i]

        if classes == 1:
            # person
            count_person += 1
            if 1 in type_dict:
                type_dict[1].append(i)
            else:
                type_dict[1] = [i]
        elif classes == 2:
            # bicycle
            count_bicycle += 1
            if 2 in type_dict:
                type_dict[2].append(i)
            else:
                type_dict[2] = [i]
        elif classes == 3:
            # car
            count_car += 1
            if 3 in type_dict:
                type_dict[3].append(i)
            else:
                type_dict[3] = [i]
        elif classes == 10:
            # traffic_light
            count_traffic_light += 1
            if 10 in type_dict:
                type_dict[4].append(i)
            else:
                type_dict[4] = [i]

    str1 = 'There is(are) {} person in the scene; {} bicycle(s) in the scene; {} car(s) in the
scene.'.format(
        count_person, count_bicycle, count_car)
    print(str1)

    if count_traffic_light != 0:
        str2 = 'There is(are) {} traffic light nearby.'.format(count_traffic_light)
        print(str2)

    return type_dict
```

```python
def calculate_depth(img):   # input is left_disparity
    disparity = cv.imread(img)
    depth = np.true_divide(calib['f'] * calib['baseline'], disparity)
    return depth


def center(ZDepth, img, img_dict):
    """
    Return list of (z, x, y) of the center point of the object inside the box.
    """
    img = cv.imread(img)
    height, width = img.shape[0], img.shape[1]
    box_depth = []

    for i in range(img_dict['num_detection']):
        depth = []
        box = img_dict['detection_boxes'][i]
        pt1 = (int(box[1] * width), int(box[0] * height))
        pt2 = (int(box[3] * width), int(box[2] * height))

        for row in range(pt1[1], pt2[1]):
            for column in range(pt1[0], pt2[0]):
                z = ZDepth[row, column, 0]
                x = np.true_divide((column - calib['px']) * z, calib['f'])
                y = np.true_divide((row - calib['py']) * z, calib['f'])
                depth.append((z, x, y))
        depth.sort()
        box_depth.append(depth[len(depth) // 2])  # (z, x, y)

    # print(box_depth)
    return box_depth


def find_closest(img_dict, box_depth, type_dict):
    """
    Find the closest object among all. box_depth is list of (z, x, y)
    """
    classes = type_dict.keys()

    for types in classes:
        types_index = type_dict[types]  # list of index
        min_index_distance = (0, np.inf)
        for i in types_index:
            pt = box_depth[i]
            distance = (pt[0] ** 2 + pt[1] ** 2 + pt[2] ** 2) ** (1 / 2)
            if distance < min_index_distance[1]:
                min_index_distance = (i, distance)
        X = box_depth[min_index_distance[0]][1]
        print_helper(min_index_distance[0], X, min_index_distance[1], img_dict)


def print_helper(index, X, min_distance, img_dict):
    if X >= 0:
        txt = 'to your right'
    else:
        txt = 'to your left'

    types = img_dict['detection_classes'][index]
    if types == 1:
        # person
        label = 'person'
    elif types == 2:
        # bicycle
        label = 'bicycle'
    elif types == 3:
        # car
        label = 'car'
    elif types == 10:
        # traffic light
```

```
        label = 'traffic light'

    str = 'There is a {} {} meters {}.\n It is {} meters away from you.\n'.format(label, abs(X),
txt, min_distance)
    print(str)


if __name__ == '__main__':
    # image1
    print('============ image1 ============')
    ZDepth1 = calculate_depth('004945_left_disparity.png')
    box_depth1 = center(ZDepth1, '004945_left_disparity.png', img1_dict)

    type_dict1 = num_object(img1_dict)
    find_closest(img1_dict, box_depth1, type_dict1)

    # image2
    print('============ image2 ============')
    ZDepth2 = calculate_depth('004964_left_disparity.png')
    box_depth2 = center(ZDepth2, '004964_left_disparity.png', img2_dict)

    type_dict2 = num_object(img2_dict)
    find_closest(img2_dict, box_depth2, type_dict2)

    # image3
    print('============ image3 ============')
    ZDepth3 = calculate_depth('005002_left_disparity.png')
    box_depth3 = center(ZDepth3, '005002_left_disparity.png', img3_dict)

    type_dict3 = num_object(img3_dict)
    find_closest(img3_dict, box_depth3, type_dict3)
```

- **Output:**

```
============ image1 ============
There is(are) 1 person in the scene; 0 bicycle(s) in the scene; 4 car(s) in the scene.
There is(are) 1 traffic light nearby.
There is a car 3.381622155166456 meters to your right.
 It is 8.012628544284036 meters away from you

There is a traffic light 0.8815176810432712 meters to your right.
 It is 20.658090033446395 meters away from you

There is a person 9.089207876464204 meters to your left.
 It is 43.68073285334254 meters away from you

============ image2 ============
There is(are) 0 person in the scene; 0 bicycle(s) in the scene; 5 car(s) in the scene.
There is(are) 1 traffic light nearby.
There is a car 3.3144706641292867 meters to your right.
 It is 35.10143476584247 meters away from you

There is a traffic light 5.293750676007732 meters to your left.
 It is 17.579606305532653 meters away from you

============ image3 ============
There is(are) 0 person in the scene; 0 bicycle(s) in the scene; 3 car(s) in the scene.
There is(are) 1 traffic light nearby.
There is a car 6.338368562074076 meters to your left.
 It is 17.26916069724466 meters away from you

There is a traffic light 4.950059994448307 meters to your right.
 It is 9.793539037883681 meters away from you
```