CSC420H1 L0101, Fall 2018
Assignment 4

Name: JingXi, Hao
Student Number: 1000654188

Due Date: 7 November, 2018 10:59pm

1. In this exercise you will map out how to build a vision system. You want to build a robot equipped with stereo cameras to take the place of the 'ball boy/girl" during a tennis match. Your robot is modelled after WALL-E (link). Write psudeo-code calling algorithms covered in class to build the robot's visual system and fulfill its destiny. Your robot already knows the following commands to aid you:

   faceDirection(X,Y,Z),
   moveToLocation(X,Y,Z),
   grabObject(X,Y,Z),
   victoryDanceAtLocation(X,Y,Z).

   It also comes with camera calibration matrices K, [R|t].

   Check out a demo Video of a simpler model; note your robot should work during a tennis match, instead of these controlled conditions.

   (a) Brainstorm about the robot's world, what it will encounter, what data it will need for training, what challenges it will face, what its world (model) will look like, etc. Create a brainstorming diagram to include in your PDF.

   Solution:
   For simplicity, we need to make some assumptions for the robot's world.

   - Set the default position as origin where the robot returns to every time after collecting a ball. Therefore, the real-world coordinates of the other objects are able to be defined with respect to that origin.
   - The balls can only drop inside the tennis court area. So, the robot only need to move within the tennis court to pick up the balls.
   - Assume there are only tennis players, a net, tennis balls, and a umpire on the tennis court during a tennis match. Treat tennis players, net, and umpire as obstacles which the robot tries to avoid during its way to grab the balls.
   - The robot only picks up the balls and do not worry about feeding. Thus, no need to give the ball to the players.

   Then, we show the tasks the robot is going to encounter in this simplified world.

- Detect the tennis court which is the area where the robot should move within to pick up the balls
- Detect the balls and the obstacles (i.e. tennis players, net, and umpire)
- Find a path to reach the balls avoiding the obstacles
- Grab a ball and back to the default position

Then, we show how the robot is able to complete its tasks.

i. Place the robot at the default position. Scan the whole tennis court to get a panorama by using the stereo cameras and homography.
ii. Employ the edge detection on the image taken to find the tennis court and set the movement boundary.
iii. Train the data for the tennis players, the net, the tennis ball, and the umpire.
iv. Face any direction at the default position and take two images by using parallel stereo cameras. Run an object detector on image taken and define each object classes (tennis player, net, tennis ball, umpire). If a ball object is not found, change another view by rotation and retake the stereo images. If ball objects found, then proceed.
v. Compute the depth of each object detected by using disparity matrices and camera parameters. Then, compute their real-world position (X, Y, Z) with respect to the origin.
vi. Face the direction of any ball detected and move to its location avoiding any obstacles. Then, grab the ball and back to default position to do the victory dance.
vii. Repeat step iv. to step vi..

Then, we list the challenges it will face below.

- The ball that is playing by the tennis players also can be detected. The robot may confused about which ball should be grabbed.
- Since the tennis players are not still, so the robot may not avoid them properly when it is on the way to fetch the balls.
- It is possible that the ball is not grabbed successfully by the robot.
- Based on the position where the stereo images are taken, the balls may be covered or in the blind spot for robot. Therefore, the robot may not detect it.

Show the brainstorming diagram below.

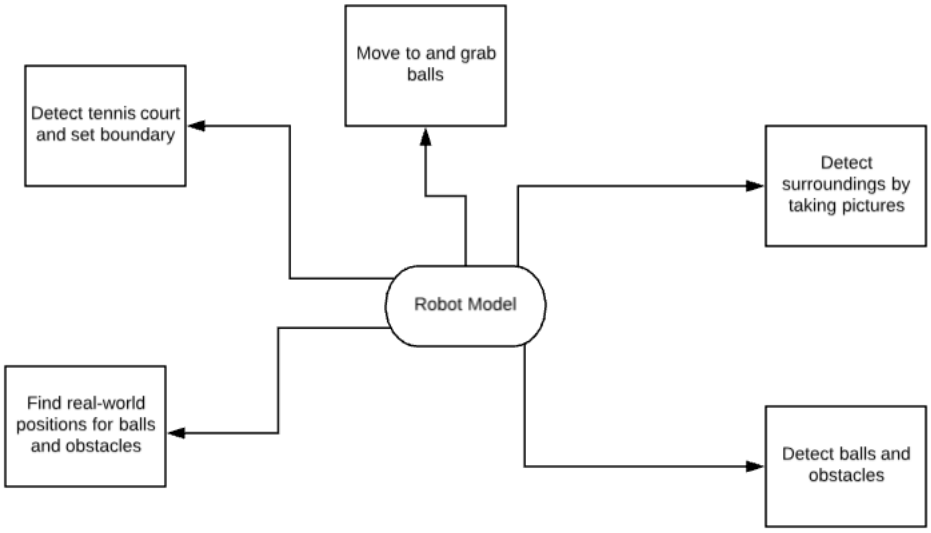

Figure 1: Brainstorming Diagram

(b) Create a flow chart linking together the main processing pipeline the robot will use. For simplicity, your robot should always return to a default location and do its victory dance there after grabbing a ball (It cannot dance on the field!). Make sure to describe what each function does in your framework. Note that you can add functions in your flowchart if you find that the ones given are not sufficient to do a full cycle.

Solution:

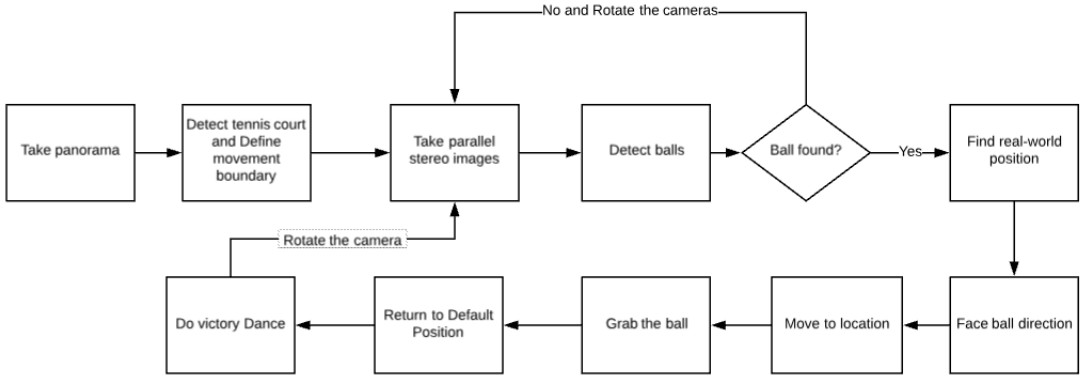

Figure 2: Flowchart

(c) Write psuedo-code to implement your robot, run algorithms from the course by name, e.g. **cannyEdgeDetection** and include relevant arguments. You will be marked on completeness and ingenuity (i.e. how we think your robot will work, the conditions it can handle, etc.).

Solution:

Show the psuedo-code of implementation of the robot below.

```
# take several panorama and use SIFT and homography to combine them as a panorama
panorama = takePanorama()

# run cannyEdgeDetection to find the tennis court and set the boundary
tennis_court = cannyEdgeDetection()
setBoundary()

# train classfier for multiple classes, here is four classes
classifer = trainClassifier()

while True:
    # take parallel stereo images at degree (total rotation is 360 degree, thus rotate inside
    # this range and start from 0)
    image_left, images_right = takeParallelImages(degree)

    # run object detector
    objects = detectObjects()

    # if ball found, then find 3D coordinate and move to it
    for object in objects:

        if object is ball:

            # find real-world posiiton with respect to the robot
            X, Y, Z = findBallPosition()

            # move to it
            faceDirection(X,Y,Z)
            moveToLocation(X,Y,Z)

            # grab it
            grabObject(X,Y,Z)

            # return to default position
            faceDirection(default_X,default_Y,default_Z)
            moveToLocation(default_X,default_Y,default_Z)

            # do victory dance
            victoryDanceAtLocation(X,Y,Z)

    # increase degree, if greater or equal to 360 degree, set it back to 0
    increaseDegree()
```

Figure 3: Psuedo-Code of Implementation of the Robot

(d) Your robot suddenly loses one of its eyes (lens)! Nevertheless, your robot should still be able to achieve its task. Describe in details what you would need to modify (software only) and why those changes are important to solve this issue. Finally, describe in what conditions the robot may still be unable to complete its task after losing an eye.

Solution:

In this case, the robot can rotate a little bit to get two stereo images taken by using only one stereo camera. Then, instead of using parallel stereo images to compute the depth in order to find real-world coordinates, the general algorithm to solve for depth by using two stereo images can be employed. Therefore, the robot still be able to fulfill its task.

2. In this exercise you are given stereo pairs of images from the autonomous driving dataset KITTI (http://www.cvlibs.net/datasets/kitti/index.php). The images have been recorded with a car driving on the road. Your goal is to create a simple system that analyzes the road ahead of the driving car and describes the scene to the driver via text. Include your code to your solution document.

   - In this assignment the stereo results have been provided with the following stereo code: http://ttic.uchicago.edu/dmcallester/SPS/spsstereo.zip. You do not need to run this code (the output of this code is already provided with the assignment). For those interested in actually running the code yourselves, you may download it from the link above.
   - To do object detection, we propose using a simple object detector trained on the MS-COCO dataset with Tensorflow. Follow the tutorial here to install Tensorflow along with the model: Daniel Stang's tutorial (Make sure to move the models folder in your tensorflow folder). To use the object detector, you can run the Jupyter notebook **object_detection_tutorial.ipynb** mentioned in the tutorial. You will need to add images to the $test\_images$ folder as mentioned in the tutorial and save $detection\_classes$, $detection\_scores$ and $detection\_boxes$ from the object $ouput\_dict$ to do the rest of the assignment (in python or MATLAB).
     Note: for those who would prefer using a different object detector, you are free to do so and/or train your own. You will need to mention which you have used in your report. You do not need to include the detector in your submission.
   - In your data directory you are given a folder called **train** and **test**. You will need to compute all your results only for the **test** folder. You only need to process the **first three images** written in the **data/test/test.txt** file.
   - Only submit your code for the following questions.

   (a) The folder results under test contains stereo results from the algorithm called **spsstereo**. For each image there is a file with extension **_LEFT_DISPARITY.png** that contains the estimated disparity.

   For each image compute depth. In particular, compute $depth$ which is a $n \times m$ matrix, where n is the height and m is the width of the original image. The value $depth(i, j)$ should be the depth of the pixel (i, j). In your solution document, include a visualization of the depth matrices. In order to compute depth you'll need the camera parameters:

   In the $test/calib$ folder there are text files with extension **_ALLCALIB.txt** which contain all camera parameters.

   Note that the baseline is given in meters.

   Solution:
   Please see the detailed code implementation of the function, ***compute_depth_for_image()***, for this question in the file, ***a4.py***.

   For each image, we first read in and store the corresponding camera parameters from the text files with extension **_ALLCALIB.txt**. Then, we obtain the disparity matrix for each image by reading the corresponding disparity image. Therefore, based on the formula

$$Z = \frac{f \cdot T}{x_l - x_r} \tag{1}$$

, where f is the focal length, T is the baseline, and $x_l - x_r$ is the disparity, we are able to compute the depth of each pixel on the image. This implies that the depth matrix of each image can be computed by repeating the same process mentioned above for each pixel on the image. Note that the depth computed in meters since the baseline given is in meters.

The visualization of the depth matrices are shown below.

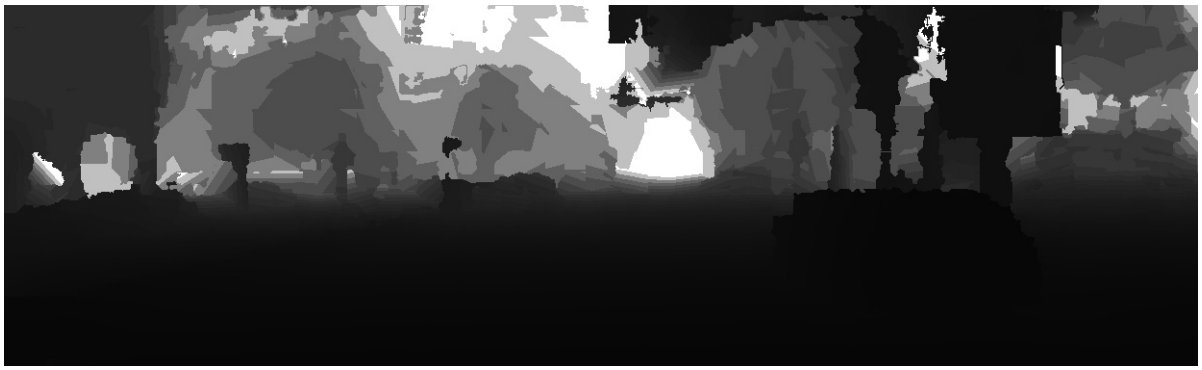

Figure 4: Visualization of Depth Matrix for 004945.jpg

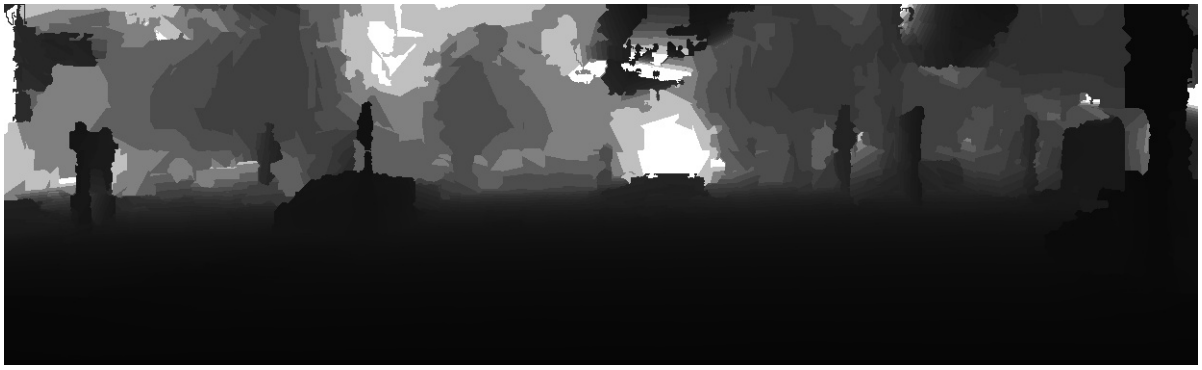

Figure 5: Visualization of Depth Matrix for 004964.jpg

Figure 6: Visualization of Depth Matrix for 005002.jpg

(b) For all test images run the object detector and keep the results for car, person, bicycle and traffic light. Store the detections in the **results** folder. Storing is important as you will need them later and it takes time to compute.

The rows in array *detection_boxes* represents a rectangle (called a bounding box) around what the detector thinks is an object. The bounding box is represented with two corner points, the top left corner (x_{left}, y_{top}) and the bottom right corner (x_{right}, y_{bottom}). The values are by default normalized by the size of the image. Each row has the following information: [x_{left}, y_{top}, x_{right}, y_{bottom}]. The object *detection_scores* reflects the how much the detector believes that there is a specific *detection_classes* object inside the bounding box. In this assignment, you only need to use the following classes: 1 (person), 2 (bicycle) ,3 (car) and 10 (traffic_light). You will need to use a threshold to only keep strong bounding box proposals. Describe what threshold you have used in your method.

Solution:

Please see the detailed code implementation of the function, ***get_and_store_detection_info()***, for this question in the file, ***a4.py***.

For this question, we employ an object detector in order to collect the detections for each image. Then, we set the threshold to be 0.35 and filter out the detection that has the greater or equal detection score in comparison with the threshold and is among one of the four classes (person, bicycle, car, and traffic light). Then, use pickle to store the selected detections in the **results** folder.

(c) In your solution document, include a visualization of the first three images with all the car, person, bicycle and traffic light detections you have kept. Mark the car detections with red, person with blue, cyclist with green and traffic light with cyan rectangles. Inside each rectangle (preferably the top left corner) also write the label, be it a car, person, cyclist or traffic light. The visualization should look similar to the one shown in the notebook.

Solution:

Please see the detailed code implementation of the function, ***draw_bounding_boxes()***, for this question in the file, ***a4.py***.

Show the visualization of the first three images with all the car, person, bicycle, and traffic light detections kept below.

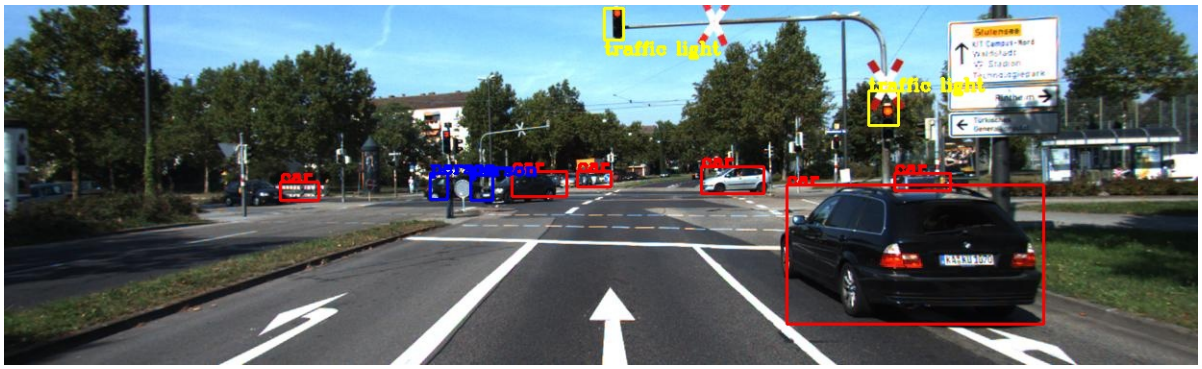

Figure 7: Visualization of Detections for 004945.jpg

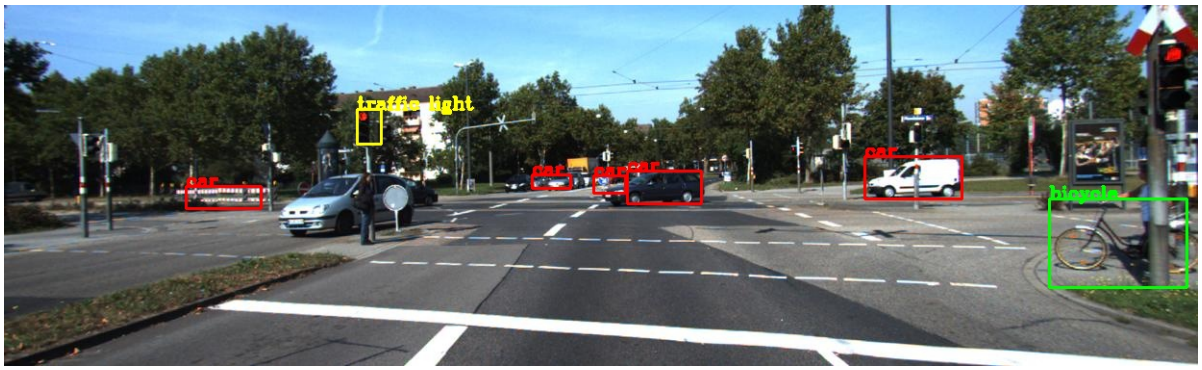

Figure 8: Visualization of Detections for 004964.jpg

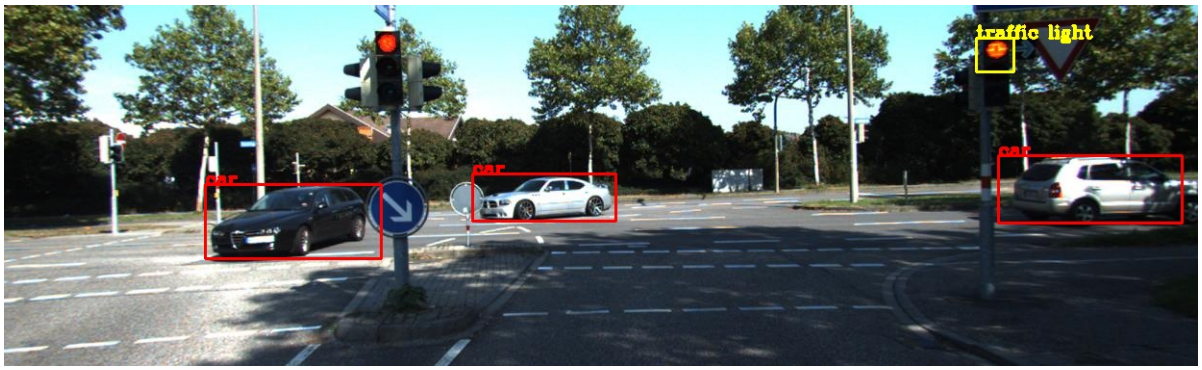

Figure 9: Visualization of Detections for 005002.jpg

(d) Compute the 3D location (centre of mass) of each detected object. How will you do that? Store your results as you'll need them later. Hint: Use depth information inside each detection's bounding box.

Solution:

Please see the detailed code implementation of the function, ***compute_center_of_mass()***, for this question in the file, ***a4.py***.

The center of mass is the average position of all the parts of the system, weighted according to their masses. In this question, the depth of each pixel on image is treated as the weight of that pixel. We are able to find the center of mass for the components along x-axis by using the formula

$$com_x = \frac{w_1 x_1 + w_2 x_2 + \dots}{w_1 + w_2 + \dots} \tag{2}$$

Similarly, the center of mass along y-axis can be computed by using the formula

$$com_y = \frac{w_1 y_1 + w_2 y_2 + \dots}{w_1 + w_2 + \dots} \tag{3}$$

Thus, for each bounding box (detected object), loop over the position coordinates inside the bounding box and apply formula (2) and (3). Then, we are able to find x and y coordinates of the center of mass for that detected object. Then, the corresponding depth is our third component in the 3D coordinate of the center of mass for that detected object. Therefore, we are able to compute the 3D location of the center of mass of each bounding box, which is the 3D location of the center of mass of each detected object.

(e) We will now perform a simple segmentation of each detected object. By segmentation, we mean trying to find all pixels inside each detection's bounding box that belong to the object. You can do this easily as follows: for each detection you have computed the 3D location of the object. Find all pixels inside each bounding box that are at most 3 meters away from the computed 3D location. Create a segmentation image. This image (a matrix) has the same number of columns and rows as the original RGB image. Initialize the matrix with all zeros. Then for the i-th detection (bounding box), assign a value i to all pixels that you found to belong to detection i. In your solution document, include a visualization of the segmentation image (for the first three test images).

Solution:

Please see the detailed code implementation of the function, ***find_segmentation()***, for this question in the file, ***a4.py***.

Show the visualization of the segmentation images for the first three test images below.

Figure 10: Visualization of Segmentation Image for 004945.jpg

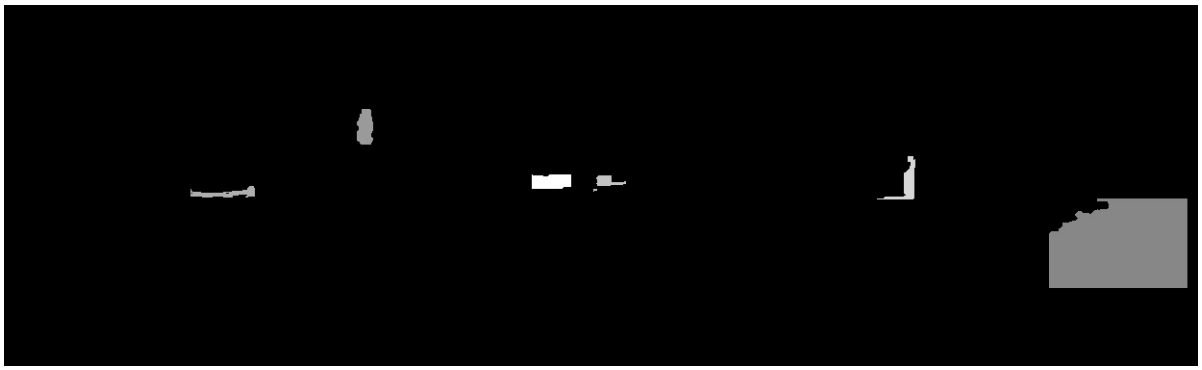

Figure 11: Visualization of Segmentation Image for 004964.jpg

Figure 12: Visualization of Segmentation Image for 005002.jpg

(f) Create a textual description of the scene. Try to make it as informative as possible. Convey more important facts before the less important ones. Think what you, as a driver, would like to know first. Think of the camera centre as the driver. In your description, generate at least a sentence about how many cars, how many people and cyclists are in the scene and if there is a traffic light

nearby. Generate also a sentence that tells the driver which object is the closest to him and where it is. For example, let's say you have an object with location (X, Y, Z) in 3D and label = car and you want to generate a sentence about where it is. You could do something like:

d = norm([X, Y, Z]); % this is the distance of object to driver
IF X $\geq$ 0, txt = "to your right"; else txt = "to your left"; end;
fprintf("There is a %s %0.1f meters %s \n", label, X, txt);
fprintf("It is %0.1 meters away from you \n", d);

Even if you didn't solve the tasks in the rest of Q2, you can still get points for this exercise by writing some pseudo code like the one above. Clearly explain what the piece of code is supposed to do.

Solution:
Please see the detailed code implementation of the function, ***make_description()***, for this question in the file, ***a4.py***.

Show the textual descriptions for each scene below.

```
In the image of 004945.jpg, there are 2 people, 0 bicycles, 6 cars, and 2 traffic lights.
There is a car 3.2 meters to your right
It is 7.7 meters away from you

In the image of 004964.jpg, there are 0 people, 1 bicycles, 5 cars, and 1 traffic lights.
There is a bicycle 8.0 meters to your right
It is 13.4 meters away from you

In the image of 005002.jpg, there are 0 people, 0 bicycles, 3 cars, and 1 traffic lights.
There is a traffic light 4.8 meters to your right
It is 9.8 meters away from you
```

Figure 13: Visualization of Textual Descriptions for Each Image