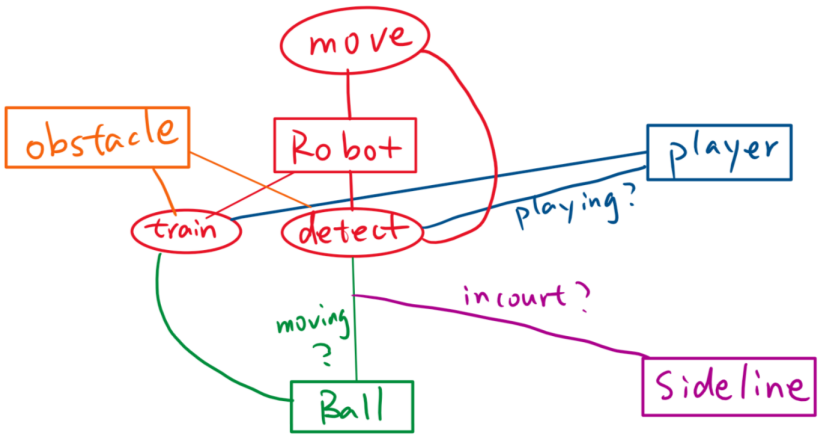
csc420 assignment 4

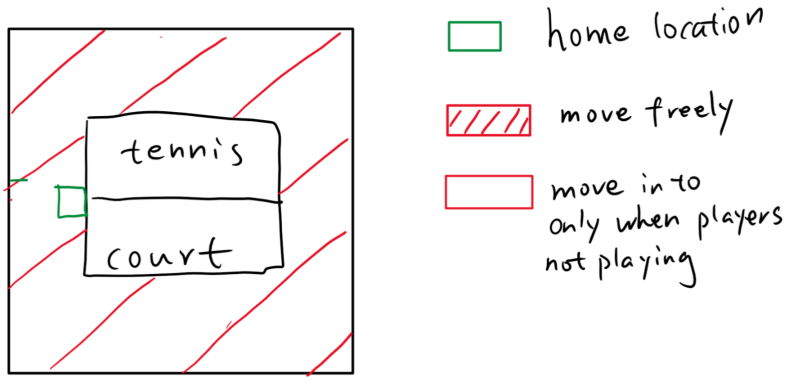
Guanchen Zhang

November 2018

1 Question 1

(a)





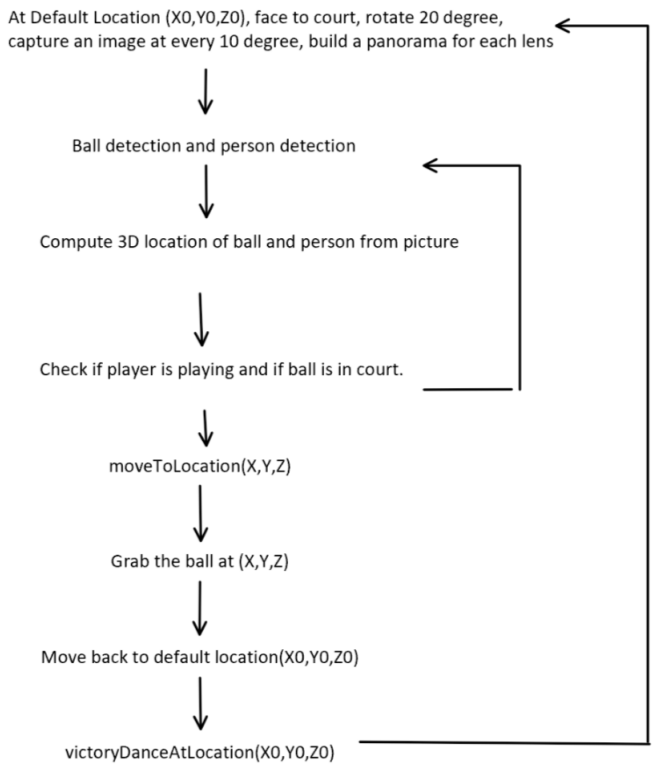
In the world of a robot, the robot is in a tennis court which consists sideline. The area inside the sideline is court in which players may be playing tennis and the area outside the sideline may contain some obstacles such as chairs and bags. A net separates a court by half. Tennis ball may be in the court or off court.

When the robot is doing its job, it may encounter obstacles off court and players playing in court. It may also be blocked from the nets so that it needs to be in the position where it can view the full court. Therefore, the ideal home location is at the position where ref is sitting because it can decide on which side the ball is.

Since the robot is not able to adjust its view by zoom in or zoom out, it is hard to detect object that is off its view. For example, the player is closed to the robot and robot can't detect him because it only sees a partial image of player. Also it is hard for robot to reschedule its routine to avoid obstacle. Another problem is that when the player is playing in court, the robot cannot run into the court because it will interrupt players, but it can move freely to pick up balls off court.

To achieve its role, the robot need to be trained with person playing tennis, tennis and some obstacle that may be in these scenarios.

(b)


At Default Location (X0,Y0,Z0), face to court, rotate 20 degree,
capture an image at every 10 degree, build a panorama for each lens
Ball detection and person detection
Compute 3D location of ball and person from picture
Check if player is playing and if ball is in court.
moveToLocation(X,Y,Z)
Grab the ball at (X,Y,Z)
Move back to default location(X0,Y0,Z0)
victoryDanceAtLocation(X0,Y0,Z0)

(c)

```
def detection(object,image1,image2):
    dection_dict = detect object using tensorflow
    filtered_dict = filterDectionWithTopScore(dection_dict)
    depth = getDepth(camera_calibration,image1,image2)
    (X,Y,Z) = cacluate3DLocation(filtered_dict,depth)
    return (X,Y,Z), filtered_dict["score"]

#go to ball and pickup
def pickUp(X,Y,Z):
    faceToLocation(X,Y,Z)
    moveToLocation(X,Y,Z)
    grabObject(X,Y,Z)

#return to home and dance
def returnHome(X,Y,Z):
    faceToLocation(X,Y,Z)
    moveToLocation(X,Y,Z)
    victoryDanceAtLocation(X,Y,Z)

# take 2 pano photo
def takePanoPhoto():
    for every 20 degree:
        rotate robot
        len1 capture image1, append to image1_list
        len2 capture image2, append to image2_list
    image1_pano = getPanorama(image1_list)
    image2_pano = getPanorama(image2_list)

# check if the ball is in court
def checkInCourt(X_sideline,Y_sideline,Z_sideline,X_ball,Y_ball,Z_ball):
    distance_sideline = norm(X_sideline,Y_sideline,Z_sideline)
    distance_ball = norm(X_ball,Y_ball,Z_ball)
    return distance_ball >= distance_sideline


def main():
```

```
def main():
    while 1:
        image1_pano, image2_pano = takePanoPhoto()

        sideline1, sideline2 = cannyEdgeDetection(image1_pano), cannyEdgeDetection(image2_pano)
        X_sideline,Y_sideline = getXY(sideline1,sideline2)
        Z_sideline = getDepth(camera_calibration,sideline1,sideline2)

        (X_ball,Y_ball,Z_ball),score_ball = detection("ball",image1_pano,image2_pano)
        (X_obstacle,Y_obstacle,Z_obstacle), score_obstacle= detection("obstacle",image1_pano,image2_pano)
        (X_player,Y_player,Z_player), score_player = detection("player",image1_pano,image2_pano)

        if score_ball >= 0.5:

            if checkInCourt(X_sideline,Y_sideline,Z_sideline,X_ball,Y_ball,Z_ball) and score_player >= 0.5:
                continue

            if (X_obstacle,Y_obstacle,Z_obstacle) in same direction (X_ball,Y_ball,Z_ball) and score_obstacle >= 0.5:
                continue


            pickup(X_ball,Y_ball,Z_ball)

            image1_2_pano, image2_2_pano= takePanoPhoto()
            (X_home,Y_home,Z_home) = detection("player",image1_2_pano, image2_2_pano)

            returnHome(X_home,Y_home,Z_home)
```

(d)
It is possible that the robot can still be able to achieve its task when it suddenly loses one of its lens. After losing an eye, we are unable to calculate the depth of the image so we are unable to get the 3D world location of the object because the robot has 2 eyes and the 3D location is calculated from the 2 eyes at that timestamp. However, we can still calculate the depth with general stereo cameras given that we know the $[R|t]$ from last shot, so we need to know the extrinsic calibration matrix which describes the camera's location in the world, and what direction it's pointing.

To accommodate this accident, we need to store the most recent images captured from both lens and the most recent location of the robot as well as the direction it faced. Therefore we can calculate the $[R|t]$ between current location and last location of the robot. If the robot stops moving, we can still get 2 images by rotate it for a specific angle. Using the 2 images captured by the same lens but different timestamp and the $[R|t]$ between 2 timestamp, we are then able to get depth.

However, it is not always that this method works. For example, if the robot moves a long distance that the object is not visible in one of the images, the algorithm might not find the correspondence match so that depth is impossible to calculate. Another problem with that method is that only static object can

be detect because at different timestamp.

2 Question 2

(a)

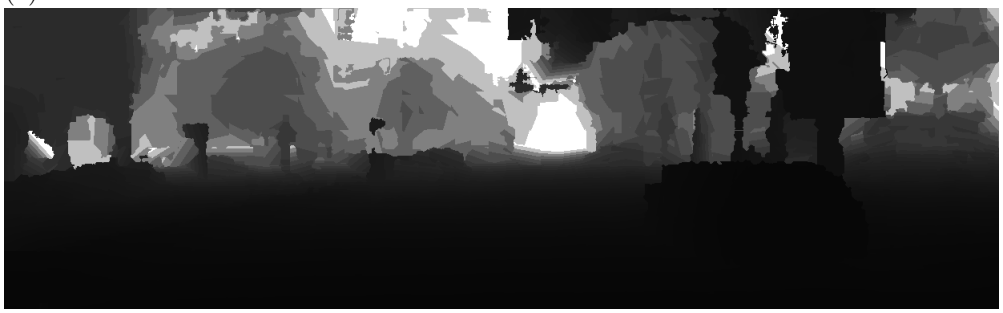


Figure 1.1 depth 004945

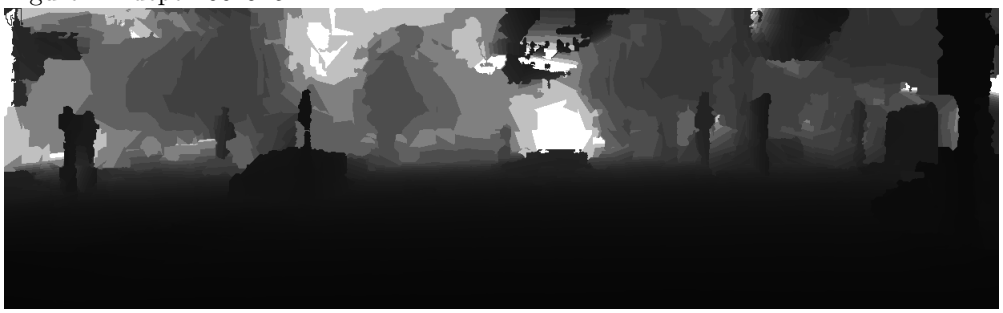


Figure 1.2 depth 004964

Figure 1.3 depth 005002

(b)

Using the pre-trained model, I run the test as it was done in the object_detection_tutorial.py and stored the raw data into a pickle file. In my script, I filtered the raw data by score larger than 0.5 and by classes. The processed data is stored in another pickle file which is uploaded along my submission.
The threshold I used is 0.5 which gave me the most reasonable visualization.

(c)

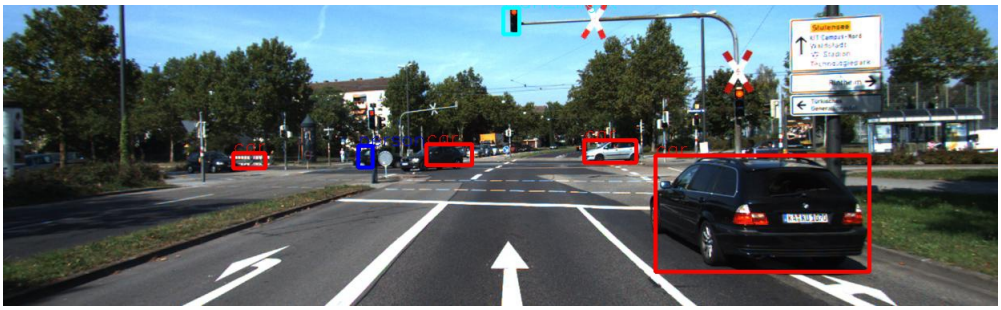


Figure 2.1 visualization 004945

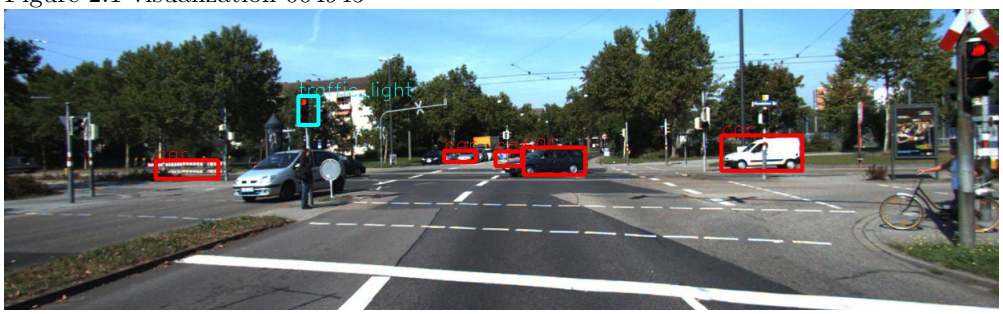


Figure 2.2 visualization 004964

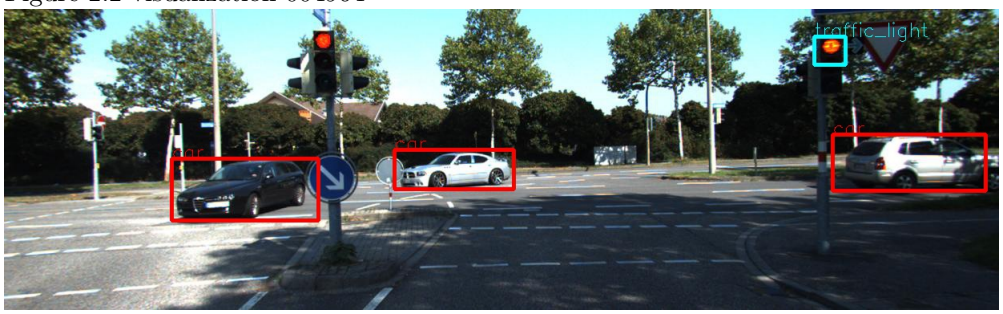


Figure 2.3 visualization 005002

(d)

For each patch of detected object in the image, we get the center pixel of the bounding box whose 3D (world) location is calculated.

Derived from the projection equation slides 10, the following equations are used to convert to world coordinates. Besides, Z is acquired from the depth we have calculated in part (a)

$$X = \frac{Z * (x - px)}{f}$$

$$Y = \frac{Z * (y - py)}{f}$$

(e)

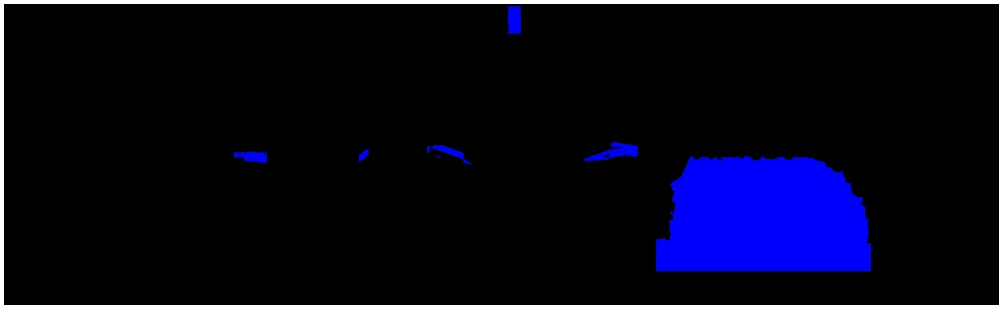


Figure 3.1 segmentation 004945

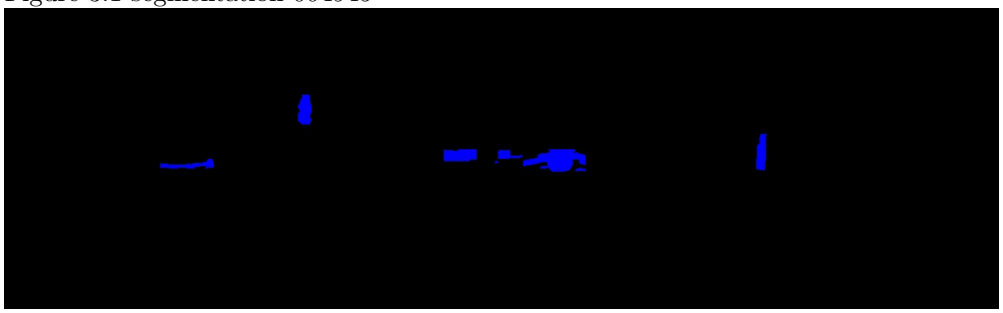


Figure 3.2 segmentation 004964

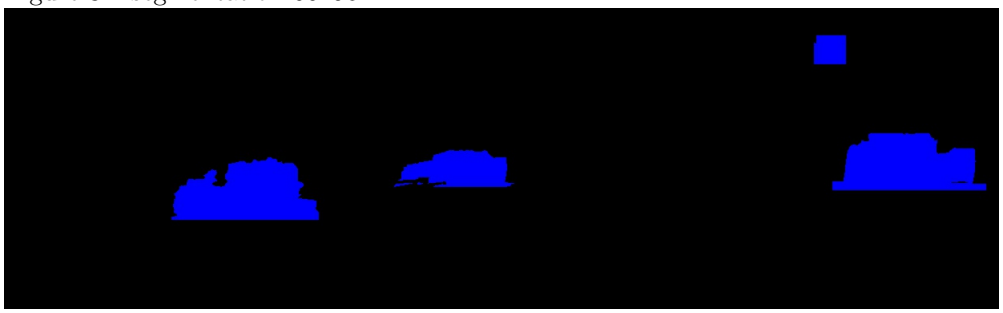


Figure 3.3 segmentation 005002

(f)

As a driver, the most important to know is the objects that are approaching and nearest to the driver, including pedestrians, cyclists and other cars. Drivers also need to know the traffic light ahead to decide whether pass or stop. Thus, it is informative for drivers to know the nearest object and the relative location of the object.

output from 004964:

```
There is a traffic_light -5.328492758320775 meters to your left

It is 17.574278228339363 meters away from you

There is a car 12.782829875673011 meters to your right

It is 30.285958452902378 meters away from you

There is a car 3.7261117000201955 meters to your right

It is 35.14968786487141 meters away from you

There is a car 1.7704414597703593 meters to your right

It is 64.09704394291467 meters away from you

There is a car -33.87686510909631 meters to your left

It is 72.50636636784198 meters away from you

There is a car -4.321284586755568 meters to your left

It is 77.00329468647266 meters away from you
```

output from 004945:

```
There is a car 3.214732894946824 meters to your right

It is 7.623999429930712 meters away from you

There is a traffic_light 0.6852553914853765 meters to your right

It is 20.69331390324515 meters away from you

There is a car -3.140591338197538 meters to your left

It is 42.830358058289924 meters away from you

There is a car 9.851221955627768 meters to your right

It is 49.04977594832761 meters away from you

There is a person -10.49171482542223 meters to your left
```

```
It is 49.18967952606931 meters away from you

There is a car -32.235389655875565 meters to your left

It is 83.38839420921117 meters away from you
```

output from 005002:

```
There is a traffic_light 4.874785482770046 meters to your right

It is 9.775467687108389 meters away from you

There is a car -6.585583032663113 meters to your left

It is 16.762088590855008 meters away from you

There is a car 13.189457769458196 meters to your right

It is 22.565210592808032 meters away from you

There is a car -1.7067841599851894 meters to your left

It is 25.69899025347936 meters away from you
```