CSC420: Assignment 4

Due on Nov. 7

Haoping Xu

November 7, 2018

Part 1

a

On a tennis court, the robot will mainly encounter tennis balls and other obstacles. There are several tasks it needs to accomplish in order to pick up all the balls. First, it needs to find balls and obstacles using its cameras. Which means it needs to search all the places on the court and detect objects in the captured images. To do that, it needs a searching algorithm to go through the whole court and a object detector to detect balls and obstacles. A CNN model can be trained to be a object detectors, with data of balls and obstacles. Also, it needs to locate detected objects using stereo images. Then it can store the locations and used in its search algorithm. Within that approach, it is necessary to assume that all the balls and obstacles are stilled, otherwise the stored information will not be correct. Finally it needs to able to keep track of its own location during the searching and organize its path to avoid detected obstacles and visit all the balls.

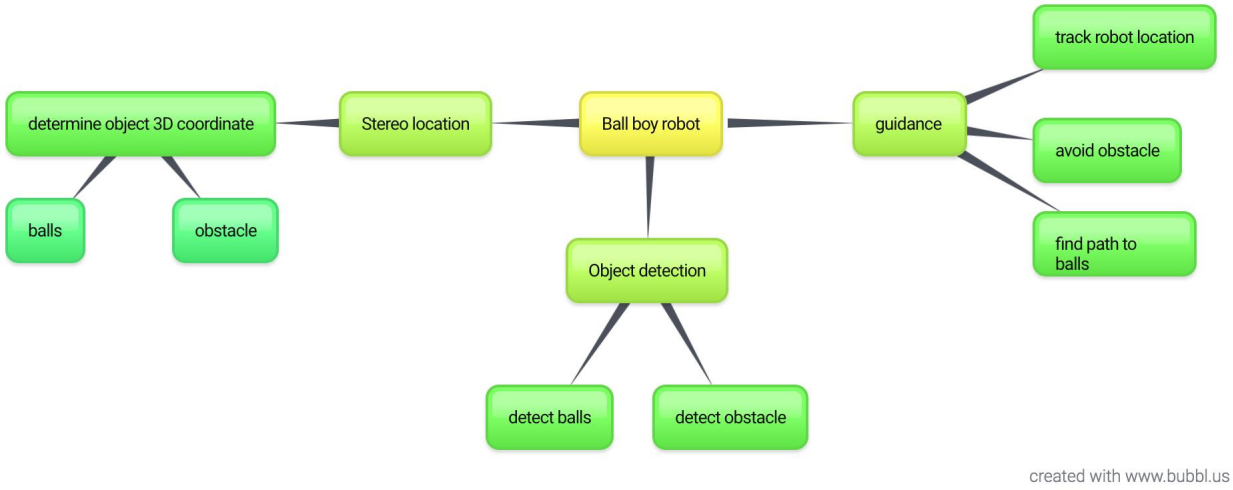

Figure 1: Brainstorms graph of the robot's world

b

1. object detect: using a CNN model like YOLO to detect objects boxes and classes in a given image
2. compute 3d location: use left and right images to compute a 3D center of mass of a object box in left image
3. move towards a ball: move from current location (X,Y) to next location($X \pm 1$, $Y \pm 1$) closest to the ball coordinate
4. avoid obstacles: if proposed next step has a obstacle, change the step by 1 unit towards to ball coordinate
5. change direction: Given current location (X,Y), give directions to (X+1, Y), (X, Y+1), (X-1, Y), (X, Y-1)
6. reset direction: Given current location (X,Y), give directions to (X+1, Y)

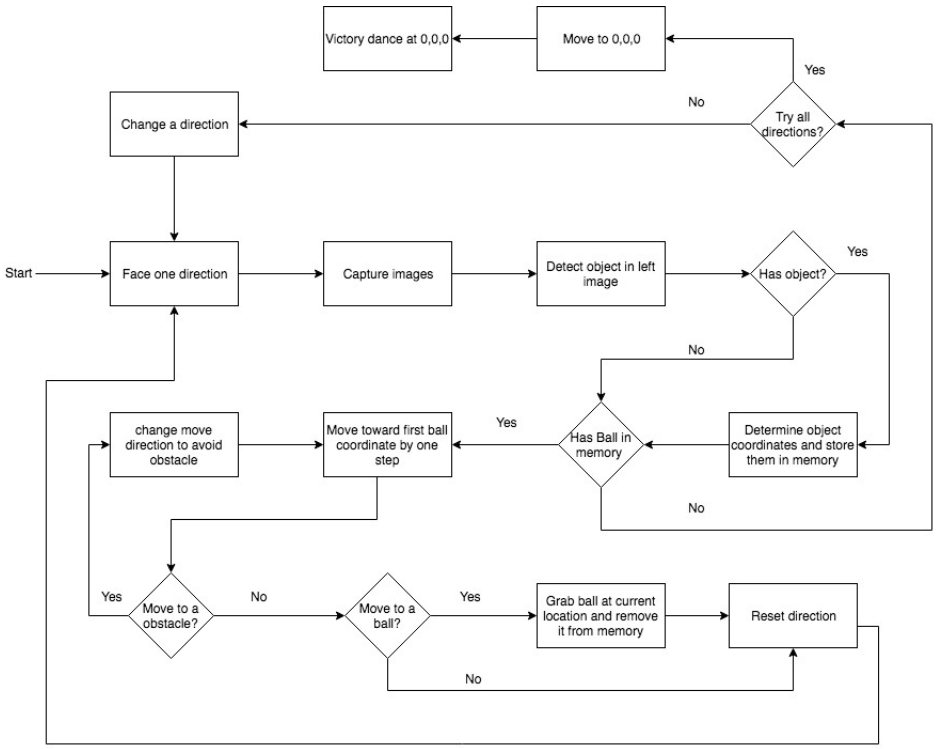

Figure 2: Flow chart of the robot main pipeline

c

```
def findLocation(image1, image2, startLocation,object, K, [R | t]):
    disparity = disparity(image1, image2)
    depth = K[0][0] * baseline / disparity
    X, Y = solve(K*[X,Y, depth, 1].T = a[object.x, object.y, 1].T)
    location = startLocation + inverse([R | T]) * [X, Y, depth, 1].T
    return location


startLocation = [0, 0, 0]
count = 0
directions = {0: [1, 0, 0], 1: [0, 1, 0], 2: [-1, 0, 0], 3: [0, -1, 0]}
balls = []
obstacles = []
while count != 4:
    faceDirection(startLocation + directions[count])
    image1, image2 = captureImagePair()
    objects = objectDetaction(image1)
    for object in objects:
        location = findLocation(image1, image2, startLocation, object, K, [R | t])
        if object.type == 'ball' and location not in balls:
            balls.append(location)
        elif object.type == 'obstacle' and location not in obstacles:
            obstacles.append(location)

    if len(balls) != 0:
        nextLocation = startLocation.copy()
        if startLocation[0] > balls[0][0]:
            nextLocation -= 1
        elif startLocation[0] < balls[0][0]:
            nextLocation += 1

        if startLocation[1] > balls[0][1]:
            nextLocation -= 1
        elif startLocation[1] < balls[0][1]:
            nextLocation += 1

        while nextLocation in obstacles:
            if startLocation[0] > balls[0][0]:
                nextLocation -= 1
            elif startLocation[0] < balls[0][0]:
                nextLocation += 1

        moveToLocation(nextLocation[0], nextLocation[1], nextLocation[2])
        startLocation = nextLocation
        if startLocation in balls:
            grabObject(nextLocation[0][0], nextLocation[0][1], nextLocation[0][2])
            balls.remove(nextLocation)
        count = 0
    else:
        count += 1
```

```
while startLocation != [0, 0, 0]:
    nextLocation = startLocation.copy()
    if startLocation[0] > 0:
        nextLocation -= 1
    elif startLocation[0] < 0:
        nextLocation += 1

    if startLocation[1] > 0:
        nextLocation -= 1
    elif startLocation[1] < 0:
        nextLocation += 1

    while nextLocation in obstacles:
        if startLocation[0] > 0:
            nextLocation -= 1
        elif startLocation[0] < 0:
            nextLocation += 1

    moveToLocation(nextLocation[0], nextLocation[1], nextLocation[2])
    startLocation = nextLocation
victoryDanceAtLocation(0, 0, 0)
```

d

if one of the cameras is damaged, the function to compute 3D location will not work as it depends on stereo images. In order to overcome it, we can take a image at location (X,Y) facing direction (X+a, Y+b), then move the robot to location (X+c, Y) and facing a parallel direction (X+a+c, Y+b). And use these two images as a stereo image pair with baseline as c. Then compute 3d location can use these images to complete the pipeline.

Part 2

a

I use this formula to compute depth based on disparity. $Z = \frac{f \times T}{x_l - x_r}$

```
def computeDepth():
    # get the image list and loop all the images
    imageList = open('data/test/test.txt')
    images = imageList.readlines()
    i = 0
    for image in images:
        image = image.rstrip()
        # get calibration file
        calFile = open('data/test/calib/{}_allcalib.txt'.format(image))
        cal = calFile.readlines()
        cal = [float(p.split()[1]) for p in cal]
        disparity = cv.imread('data/test/results/{}_left_disparity.png'.format(image), 0)
        # depth = f * baseline / disparity
        depth = cal[0] * cal[3] / disparity
        if i < 3:
            # show disparity and depth
```

```
            f, (ax1, ax2) = plt.subplots(2, sharex=True, sharey=True)
            ax1.imshow(disparity)
            ax2.matshow(depth)
            plt.show()
        i += 1
        # save depth info in result folder
        np.savetxt('data/test/results/{}_depth.txt'.format(image), depth)
```

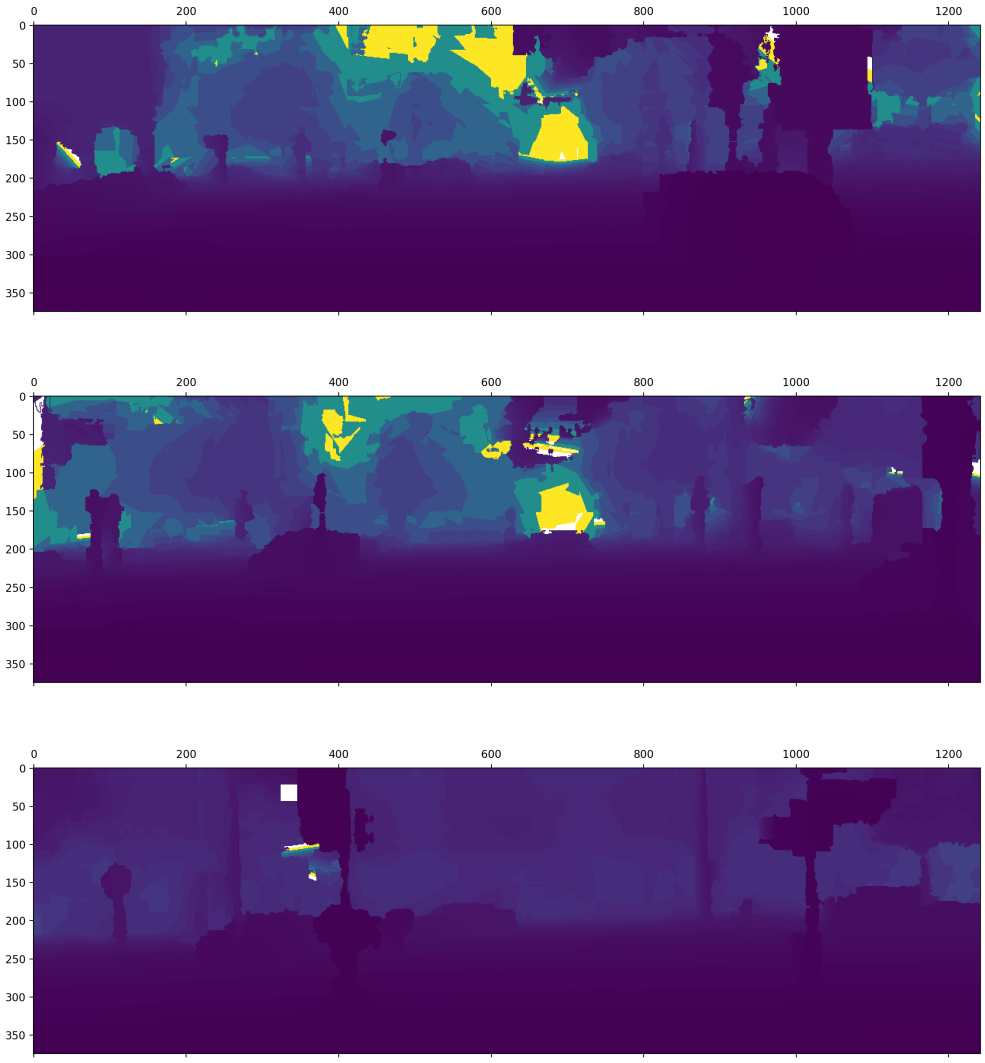

Figure 3: Depth visualization for first 3 images

b

I use the tutorial code in *model/research/object*$_d$*etection*. Then use the following code to filter and save the results. The threshold I use is 0.35.

```
class_mask = np.logical_and(
```

```
    np.logical_or(output_dict['detection_classes'] < 4,
                  output_dict['detection_classes'] == 10),
    output_dict['detection_scores'] >= 0.35)
output_dict['detection_boxes'] = output_dict['detection_boxes'][np.nonzero(class_mask)]
output_dict['detection_classes'] = np.extract(class_mask, output_dict['detection_classes'])
output_dict['detection_scores'] = np.extract(class_mask, output_dict['detection_scores'])
```

c

```
def visualBox():
    # get the image list and loop all the images
    imageList = open('data/test/test.txt')
    images = imageList.readlines()
    colors = {1: (0, 0, 255), 2: (0, 255, 0), 3: (255, 0, 0), 10: (0, 255, 255)}
    className = {1: 'person', 2: 'bicycle', 3: 'car', 10: 'traffic light'}
    for i in range(3):
        image = images[i].rstrip()
        im = cv.imread('data/test/left/{}.jpg'.format(image))
        im = cv.cvtColor(im, cv.COLOR_BGR2RGB)
        # get box and classes result of object detector
        boxes = np.loadtxt('data/test/results/{}_box.txt'.format(image)) *
        (im.shape[:2] + im.shape[:2])
        boxes = boxes.astype(int)
        classes = np.loadtxt('data/test/results/{}_class.txt'.format(image))
        for i in range(classes.shape[0]):
            # a box for each object detected
            cv.rectangle(im, (boxes[i][1], boxes[i][0]), (boxes[i][3], boxes[i][2]),
            colors[classes[i]], 7)
            # put the class name of the object
            name = className[classes[i]]
            cv.putText(im, name, (boxes[i][3] - 16 * len(name), boxes[i][0] - 10),
            cv.FONT_HERSHEY_SIMPLEX, 1.2,
                       colors[classes[i]], 5)
        plt.imshow(im)
        plt.show()
```

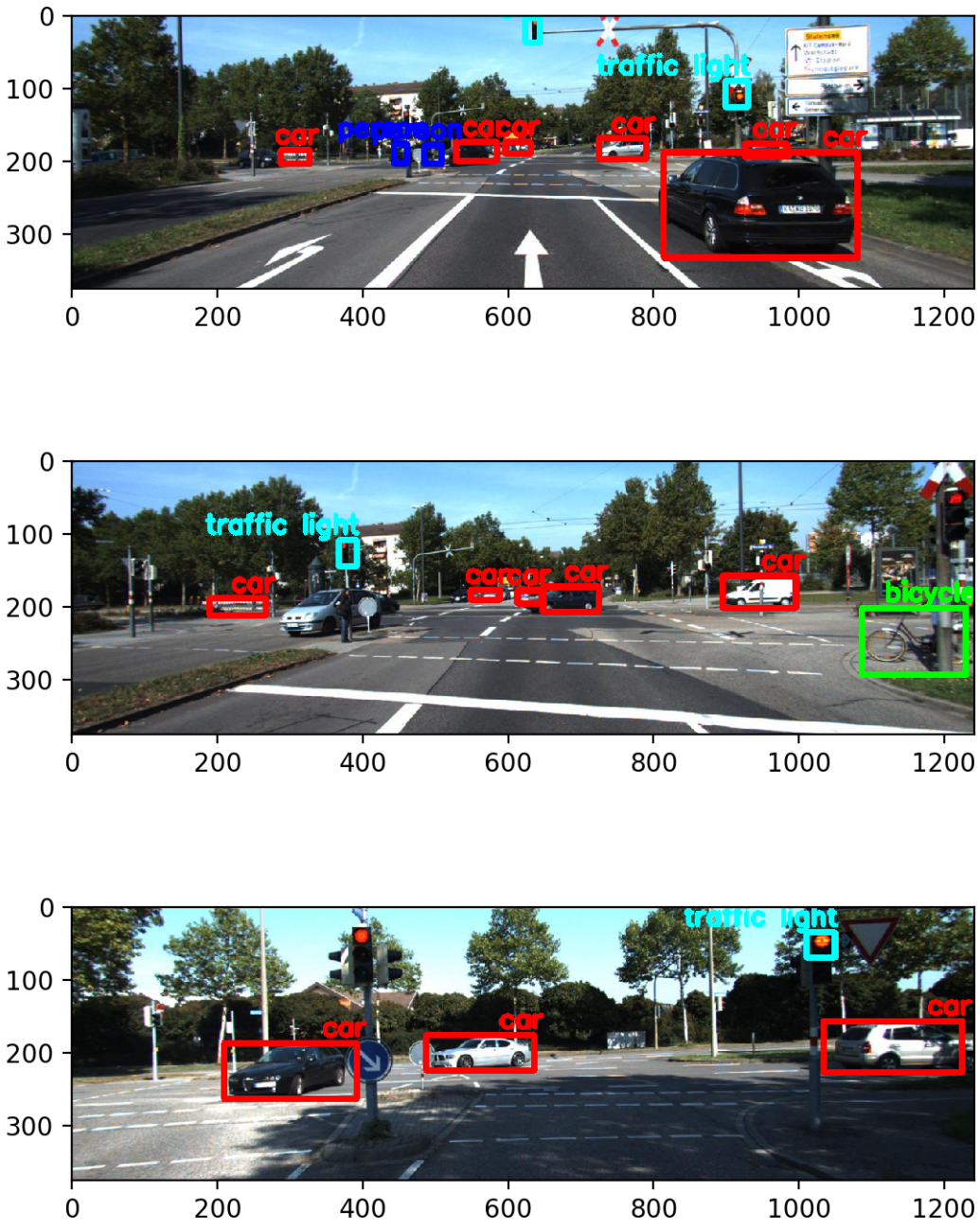

Figure 4: Box and class visualization for first 3 images

d

For each detected box, I find the most common depth within it and use it and the center of box to compute the center of mass for that box. Since $x = \frac{f \times X}{Z} + p_x$ and $y = \frac{f \times Y}{Z} + p_y$, $[X, Y, Z]$ can be computed with Z, x, y, f, p_x and p_y.

```
def center3D():
    # get the image list and loop all the images
    imageList = open('data/test/test.txt')
    images = imageList.readlines()
    for image in images:
        image = image.rstrip()
        im = cv.imread('data/test/left/{}.jpg'.format(image))
        # get box object detector
        boxes = np.loadtxt('data/test/results/{}_box.txt'.format(image)) *
        (im.shape[:2] + im.shape[:2])
        boxes = boxes.astype(int)
        # get depth info
        depth = np.loadtxt('data/test/results/{}_depth.txt'.format(image))
        # get calibration info
        calFile = open('data/test/calib/{}_allcalib.txt'.format(image))
        cal = calFile.readlines()
        cal = [float(p.split()[1]) for p in cal]
        center_depth = []
        for i in range(boxes.shape[0]):
            # for each detected object, get the depth info in its box
            box = boxes[i]
            box_depth = depth[box[0]: box[2] + 1, box[1]:box[3] + 1]
            box_depth[np.isinf(box_depth)] = 0
            # get histogram of all pixels depth within that box
            hist, bin = np.histogram(box_depth.flatten(), 'auto')
            # use the most common depth as center of mass's depth
            max_bin = np.argmax(hist)
            box_depth = (bin[max_bin] + bin[max_bin + 1]) / 2.0
            # use depth and box center to get 3D center of mass
            center = reprojectTo3D(cal, (box[1] + box[3]) / 2.0,
                (box[0] + box[2]) / 2.0, box_depth)
            center_depth.append(center)
        # save the boxes' center of mass to result folder
        center_depth = np.array(center_depth)
        np.savetxt('data/test/results/{}_3D.txt'.format(image), center_depth)


def reprojectTo3D(cal, x, y, z):
    # as K * [X, Y, Z].T = w[x, y, 1]
    # x = f*X/Z + px
    # y = f*Y/Z + py
    X = (x - cal[1]) * z / cal[0]
    Y = (y - cal[2]) * z / cal[0]
    return [X, Y, z]
```

e

```
def reprojectTo3D(cal, x, y, z):
    # as K * [X, Y, Z].T = w[x, y, 1]
    # x = f*X/Z + px
    # y = f*Y/Z + py
    X = (x - cal[1]) * z / cal[0]
    Y = (y - cal[2]) * z / cal[0]
    return [X, Y, z]


def distance3D(point1, point2):
    return np.sqrt(np.sum(np.square(point1 - point2)))


def segment():
    # get the image list and loop all the images
    imageList = open('data/test/test.txt')
    images = imageList.readlines()
    for i in range(3):
        image = images[i].rstrip()
        im = cv.imread('data/test/left/{}.jpg'.format(image))
        # get boxes result
        boxes = np.loadtxt('data/test/results/{}_box.txt'.format(image)) *
        (im.shape[:2] + im.shape[:2])
        boxes = boxes.astype(int)
        # get depth info
        depth = np.loadtxt('data/test/results/{}_depth.txt'.format(image))
        # get center of mass 3D point info
        center = np.loadtxt('data/test/results/{}_3D.txt'.format(image))
        # get calibration info
        calFile = open('data/test/calib/{}_allcalib.txt'.format(image))
        cal = calFile.readlines()
        cal = [float(p.split()[1]) for p in cal]
        seg = np.zeros((im.shape[0], im.shape[1]))
        for i in range(boxes.shape[0]):
            box = boxes[i]
            for y in range(box[0], box[2]):
                for x in range(box[1], box[3]):
                    # for each pixel within a box, get it 3D point using
                    # picture coordinate and depth
                    point = reprojectTo3D(cal, x, y, depth[y][x])
                    # compute its distance from center of mass
                    distance = distance3D(point, center[i])
                    if distance <= 3:
                        seg[y][x] = i+1
        plt.matshow(seg, cmap='coolwarm')
        plt.show()
```

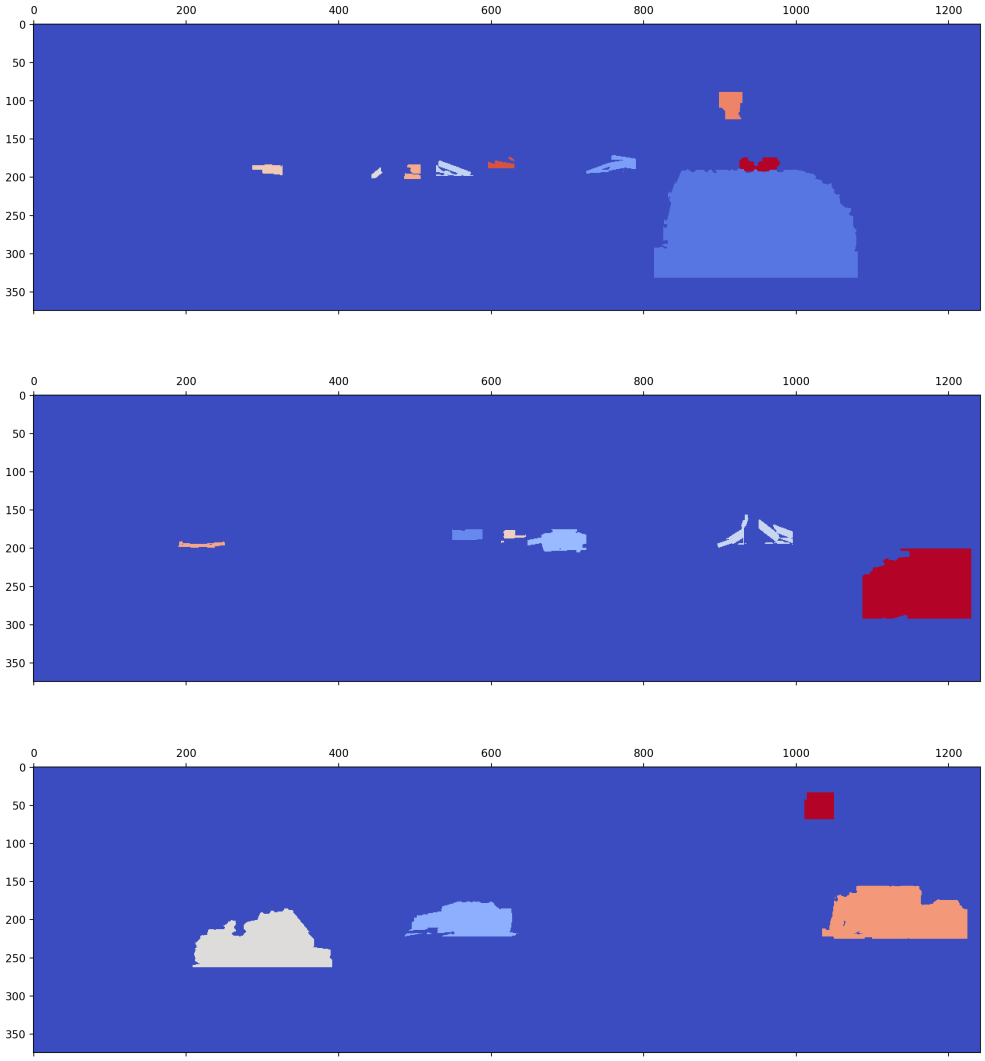

Figure 5: Segmentation visualization for first 3 images

f

```
def sentence():
    # get the image list and loop all the images
    imageList = open('data/test/test.txt')
    images = imageList.readlines()
    className = {1: 'person', 2: 'bicycle', 3: 'car', 10: 'traffic light'}
    for image in images:
        image = image.rstrip()
        # get center of mass 3D point info
        center = np.loadtxt('data/test/results/{}_3D.txt'.format(image))
        # get class results
        classes = np.loadtxt('data/test/results/{}_class.txt'.format(image))
        # count each class' appearance
        people = np.count_nonzero(classes == 1)
```

```
        bicy = np.count_nonzero(classes == 2)
        car = np.count_nonzero(classes == 3)
        light = np.count_nonzero(classes == 10)
        summary = "There are {} people, {} bicycles, {} cars and {} traffic light
        in the scene".format(people, bicy, car, light)
        # compute distance of each box to camera origin
        ori = np.zeros(3)
        dist = np.array([distance3D(ori, point) for point in center])
        # find closest box
        closest = np.argmin(dist)
        # determine its class and location
        name = className[classes[closest]]
        direction = 'left' if center[closest][0] < 0 else 'right'
        closest = "The closest {} is {} meters away to your {}".format(
        name, dist[closest], direction)
        # print both sentences
        print(summary)
        print(closest)
```

And here are the results:

```
There are 2 people, 0 bicycles, 6 cars and 2 traffic light in the scene
The closest car is 7.744101046625386 meters away to your right
There are 0 people, 1 bicycles, 5 cars and 1 traffic light in the scene
The closest bicycle is 13.909172269062523 meters away to your right
There are 0 people, 0 bicycles, 3 cars and 1 traffic light in the scene
The closest traffic light is 11.845450206465687 meters away to your right
There are 2 people, 0 bicycles, 2 cars and 0 traffic light in the scene
The closest car is 18.593482354410863 meters away to your left
There are 0 people, 0 bicycles, 7 cars and 1 traffic light in the scene
The closest car is 14.94728936071231 meters away to your left
There are 0 people, 0 bicycles, 0 cars and 2 traffic light in the scene
The closest traffic light is 7.070970546178106 meters away to your left
There are 1 people, 0 bicycles, 5 cars and 0 traffic light in the scene
The closest car is 5.952771073713146 meters away to your left
There are 0 people, 0 bicycles, 4 cars and 0 traffic light in the scene
The closest car is 16.613716281676474 meters away to your left
There are 0 people, 0 bicycles, 3 cars and 2 traffic light in the scene
The closest car is 4.775939868748039 meters away to your right
There are 0 people, 0 bicycles, 6 cars and 0 traffic light in the scene
The closest car is 4.01660938938406 meters away to your right
There are 0 people, 0 bicycles, 11 cars and 0 traffic light in the scene
The closest car is 5.537027960597485 meters away to your right
There are 0 people, 0 bicycles, 4 cars and 0 traffic light in the scene
The closest car is 5.860247433358296 meters away to your left
There are 0 people, 0 bicycles, 2 cars and 0 traffic light in the scene
The closest car is 7.377663873651953 meters away to your right
There are 0 people, 0 bicycles, 3 cars and 0 traffic light in the scene
The closest car is 39.844200524036665 meters away to your left
There are 0 people, 0 bicycles, 4 cars and 0 traffic light in the scene
The closest car is 12.714121129432309 meters away to your left
There are 0 people, 0 bicycles, 3 cars and 2 traffic light in the scene
The closest car is 8.842611611180166 meters away to your left
There are 1 people, 0 bicycles, 2 cars and 1 traffic light in the scene
```

```
The closest traffic light is 7.448269131402707 meters away to your left
There are 0 people, 0 bicycles, 4 cars and 2 traffic light in the scene
The closest car is 4.9864581749136425 meters away to your right
There are 1 people, 0 bicycles, 2 cars and 0 traffic light in the scene
The closest car is 7.222476251787816 meters away to your left
There are 0 people, 0 bicycles, 7 cars and 0 traffic light in the scene
The closest car is 6.606471340692629 meters away to your right
```