Autonomous Driving Report

Hongman Cho, 1000216250
Murali Komaravolu, 1000564490
Vibhavi Peiris, 1000597687

Introduction

For this project, we were tasked with taking a set of images and correctly identifying the road and the cars on the road. The advancement of machine learning and vision techniques has enabled growth of the autonomous driving industry and research. This was why we decided to tackle this particular project. Not only did it allow us to apply a lot of the algorithms and techniques we have learning through the semester but also apply it in a real world practical setting. If the is done correctly, it can allow cars to detect other obstacles while navigating on the road. It is very important that few mistakes are done else it can lead to accidents.

Before starting, we first researched con cepts and looked back at what we learning in the course that could be helpful to our project. Some of the work done from previous assignments were also useful. We thought A3 could be important in finding the depth and 3D location of each pixel. For disparity, we found that there was a built in disparity method that could give us a disparity map. This disparity was required for calculating the depth and 3D location of each pixel. During the last couple of weeks, we learned about HOG descriptors and superpixels, which we thought could work well as features for training an svm classifier. Once the detections are complete, we just need to visualize them using best fit plane and 3d bounding boxes.

We all contributed equally to all parts of this challenging project, and there isn't an easy to say who did what part. However below are the rough divisions as requested by the TA.

Murali: Found the disparity, depth, fit the plane for the road & detected cars. Part 1b,c,f & 2b
Hongman: trained viewpoint classifiers, predicted viewpoint & the 3d bounding box. Part 2 c,d,e
Vibhavi: trained road classifiers, predicted road & plotted 3d point cloud. Part 1d,e,g

Part 1

1.1. Methods

(main.m)

- First we downloaded the required training and testing images from KITTI. We downloaded DPM from assignment 4 and modified getData.m to retrieve the road images like we did with the car images (getDataRoad.m). This made it easier for us to set and get the image data such as disparity and calibration values.

- **Disparity**: (disparity1b.m) To find the disparity map, we took the left and right images and used the build-in disparity method to return the disparity values and save them. This disparity method takes in two stereo images, one for left and right, and returns a disparity map. It also takes in a Disparity Range of [0 16*15] and a patch size of 15 in our case. It returned something that looked like this:

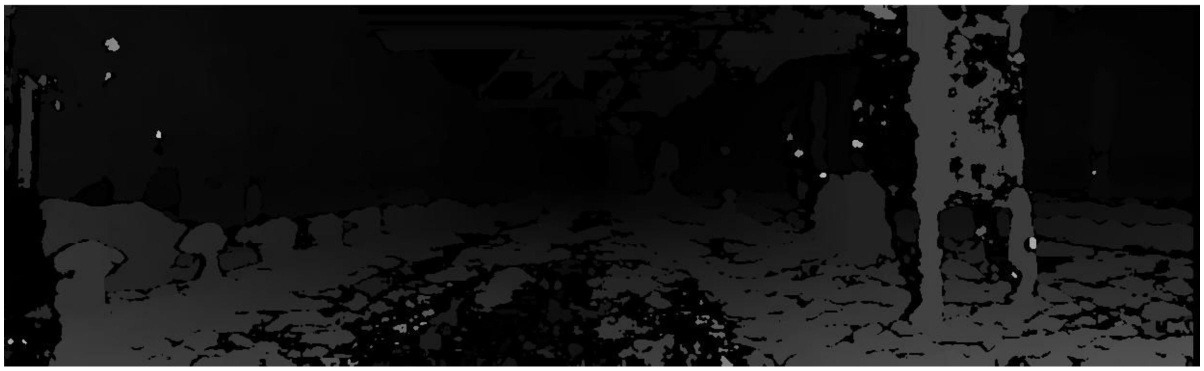


Figure 1.1: disparity for 'train/left/000000.png' Image

- **Depth**: (depth1c.m) Now that we had the disparity map and the calibration values, we could find the depth of the image using:

  depth = calib.f * calib.baseline / disparity;

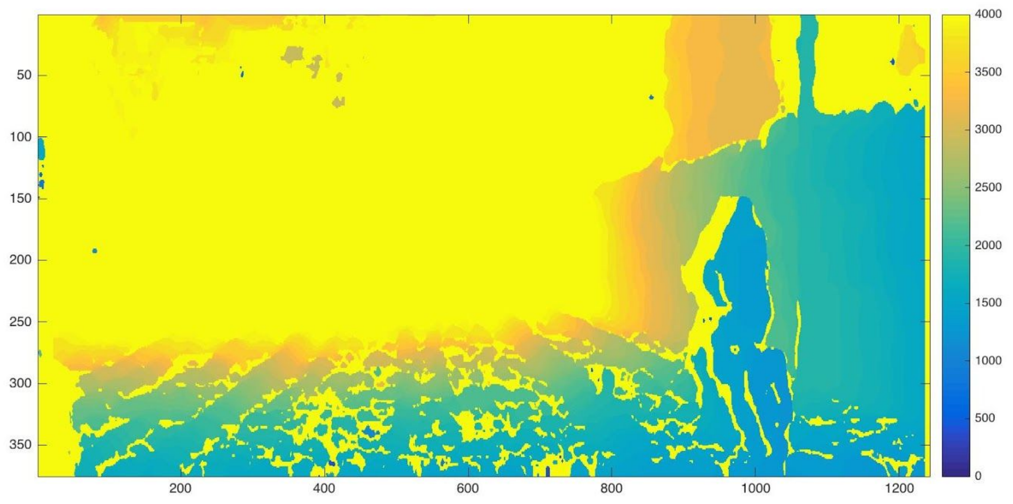


Figure 1.2: depth for ‘train/left/000000.png’ Image

- **Training**: (train1d.m) Murali and Vibhavi first thought of extracting features for each pixel to create a feature vector. Hongman suggested that we should use the superpixels of each image instead to reduce the training time. We looped through the training images superpixels and labeled each superpixel as ‘road’ or ‘not road’ based on its corresponding gt_road image. We created a [1 x 9] feature vector for each superpixel in each image, as follows:

  [R G B hue(1) hue(2) hue(3) 3Dy-coord Gradx Grady]

  We initially experimented with using HOG descriptors but then decided against as it was useless for road detection. It is better with object detection, so we used them in part B.

  After getting the feature vectors for each superpixel, we took a random sample of 250 superpixels per image and trained a model using fitcsvm.

- **Predicting**: (predict1e.m) Once we had a trained svm model, we computed the feature vectors for our test image superpixels. We then used the built in predict method, which takes in our svm model and computed features, to return a prediction and score for each image. We used the prediction to find all the superpixels that belong to the road and saved the output(Figure 1.3).

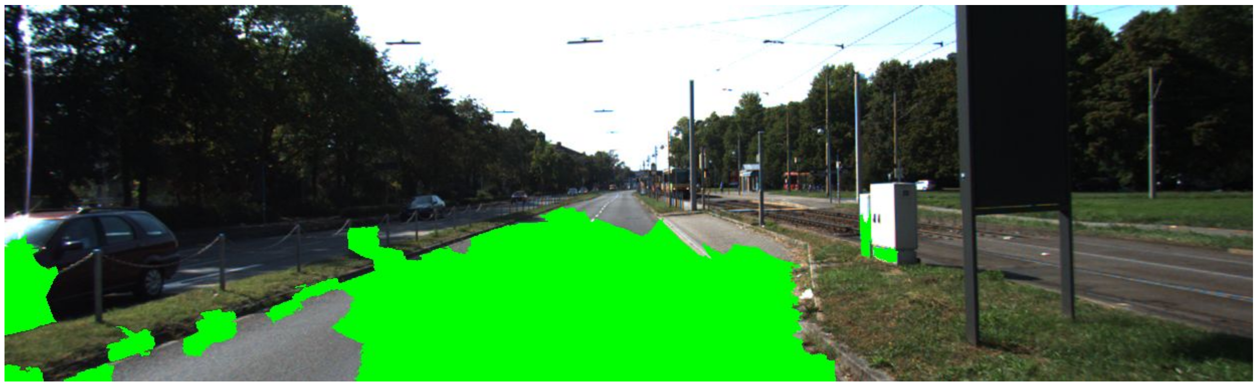


Figure 1.3: predicted road for ‘test/left/000000.png’ Image

- **Fit plane**: (fitplane1f.m) We used select() to return a cropped point cloud (Figure 1.4) given the predicted image and the original point cloud. Then, we used pcfitplane to return a plane model that goes through the point cloud and is robust to outliers. It returns a set of inliers (Figure 1.5), outliers, and the plane road.

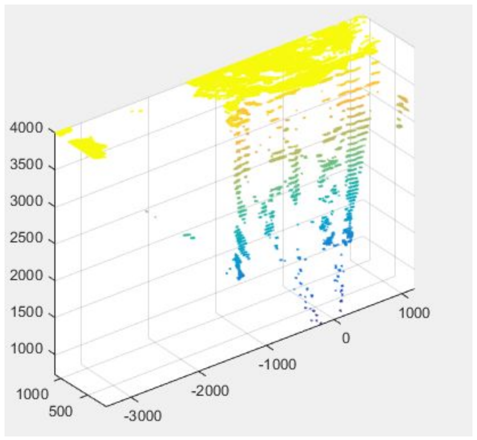


Figure 1.4: The cropped point cloud of the road pixels for ‘test/left/000000.png’ Image

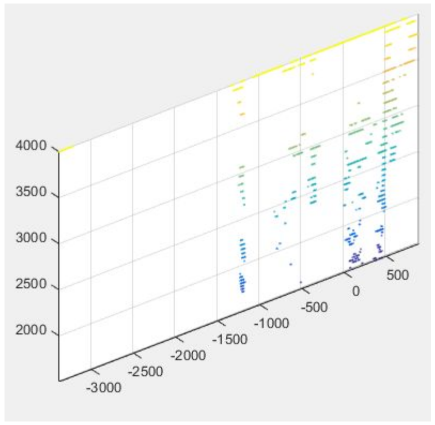


Figure 1.5: Inliers for ‘test/left/000000.png’ Image

- **Point Cloud:** (drawPlane.m) After getting a set of inliers, we fit a mesh grid through the points (Figure 1.6).

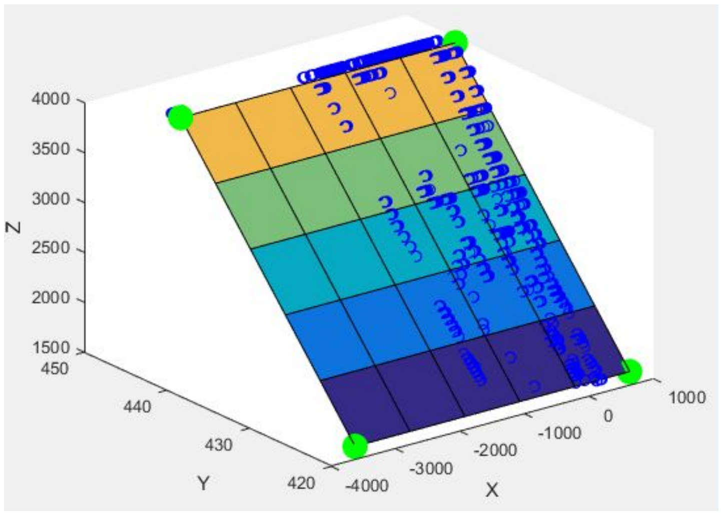


Figure 1.6: plane for 'test/left/000000.png' Image

1.2. Main Challenges

The biggest challenge we faced was figuring out what kind of features we would use for the classifier. Our initial approach of using a feature vector given from the HOG descriptor turned out not to fare well. The speed of the SVM training took a look time with our first approach because we were calculating for each pixel. The results from this implementation where random and did not get the road well at all figure(3.1). Some features resulted in the model always getting the sky.

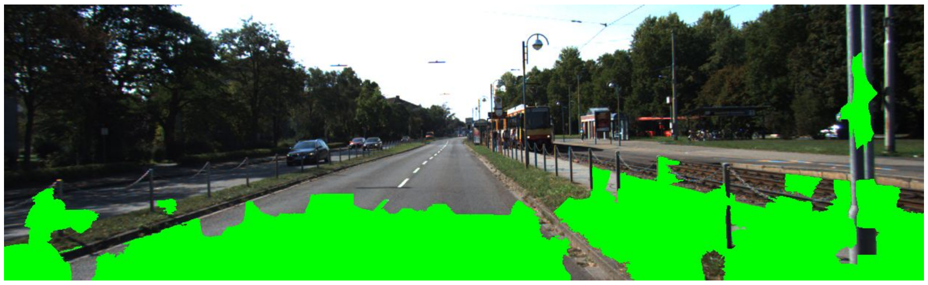


Figure 3.1: Bad predictions, due to bad features used.

The use of superpixels fixed this problem because it reduced the number of pixels from thousands to around 300. Since HOG descriptors are useless for detecting road, we tried several 3D features, such as gradients, hue, 3D location, depth, magnitude gradient, etc. After isolating each feature and training, we came up with a feature vector that we ended up using.

1.3. Results

We trained 94 images which took a total of 48 minutes. It took 15 to 30 seconds to compute the features for each image. Predicting an image took around 20 to 30 seconds.

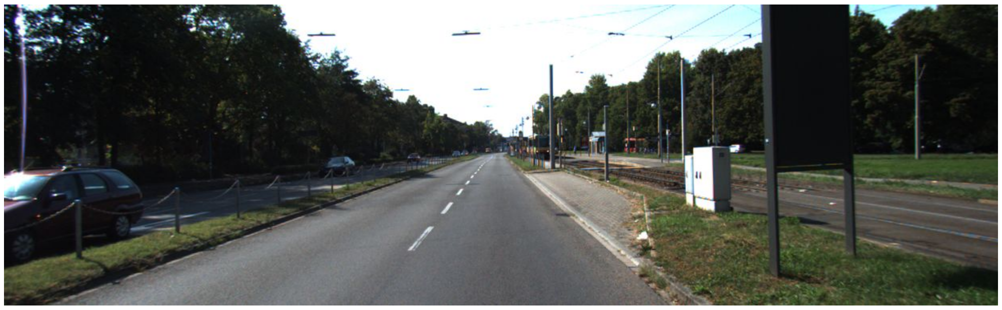


Figure 4.1: Test image ‘um_000000’

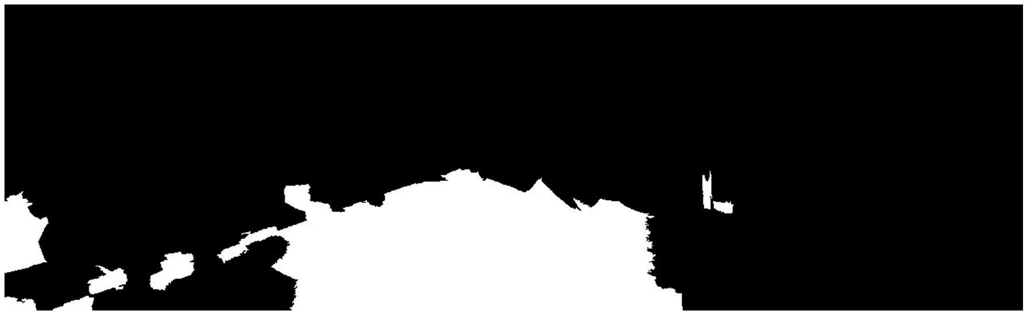


Figure 4.2: Predicted gt image for test image ‘um_000000’

Most of the results we get from the training identify superpixels that are on the road correctly. However there are times as seen the in figure 4.3 it gets other artifacts as well. Another issue is that the predictions don’t always get all the road pixels as seen in figure 4.4.

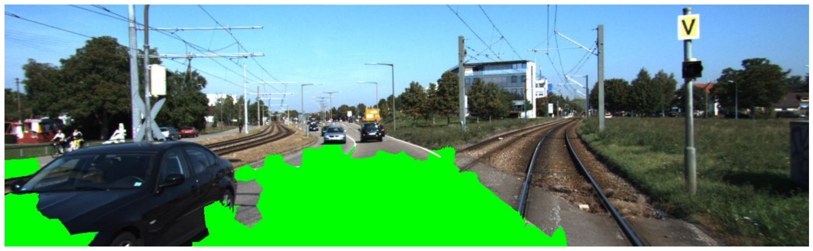


Figure 4.3: Gets the road but sometimes it also gets parts of the car.

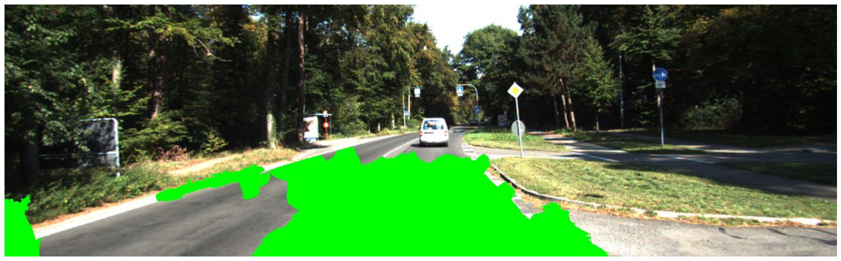


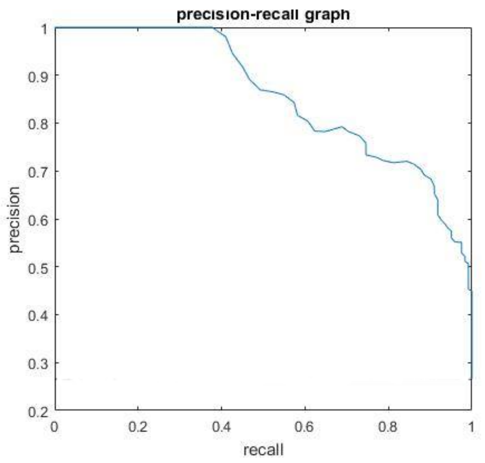
Figure 4.4: Not all the road pixels are predicted to be road.

There are 2 possible solutions to these issues:

- One is to further tune the parameters like what features we use and how many random superpixels we choose. For example, we noticed that using histogram of hue values (as suggested by our professor), as one of our features greatly improved the results.
- The other option is to use ransac to only take predicted superpixels that lie on the ground plane.

Precision-Recall Curve and ROC:

- This is precision-recall curve



1.4. Conclusion

Overall, we believe that with more time we could increase the level of accuracy and speed of our solution. We used concepts and techniques that we learned through the year and applied them accordingly, like training models, using superpixels, computing depth and disparity. If we had to move forward with our project, we would also train for pedestrians and cyclists so that the car could be even better at navigating safely.

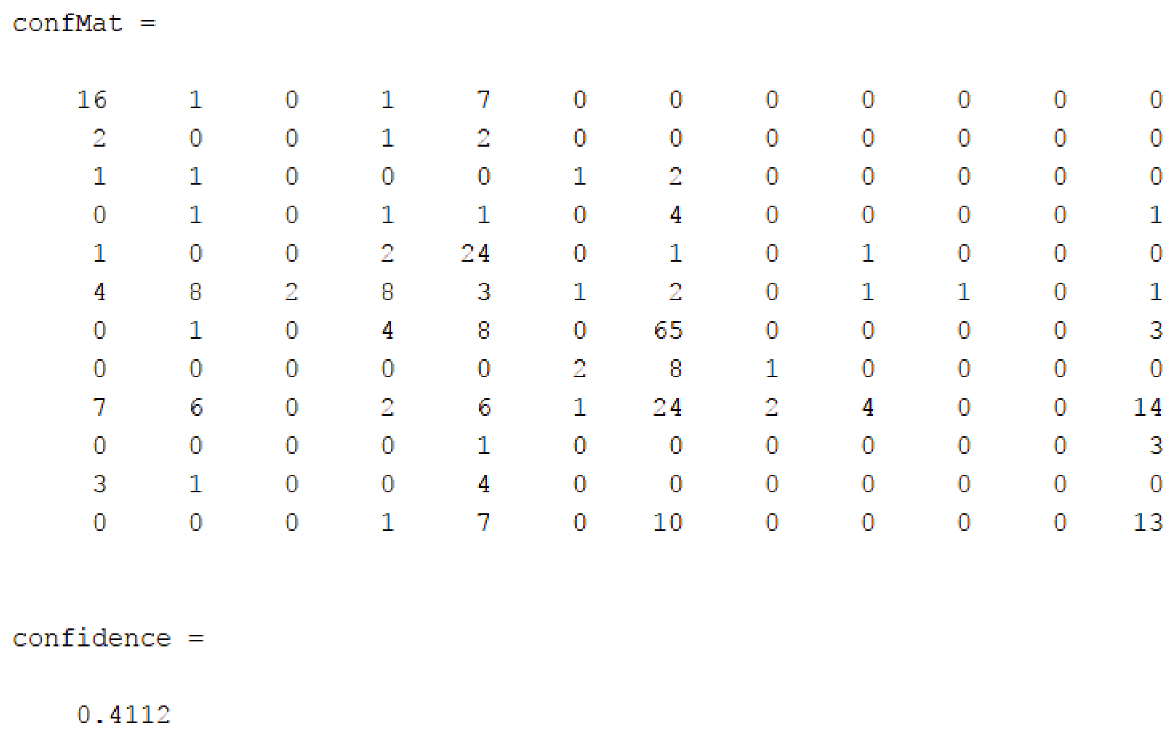
Part 2

Results

Best classification accuracy: **46.38%**

2.1. Confusion Matrices

Figure 1. Test 1 condition: All samples, and 3x3 HOG feature size

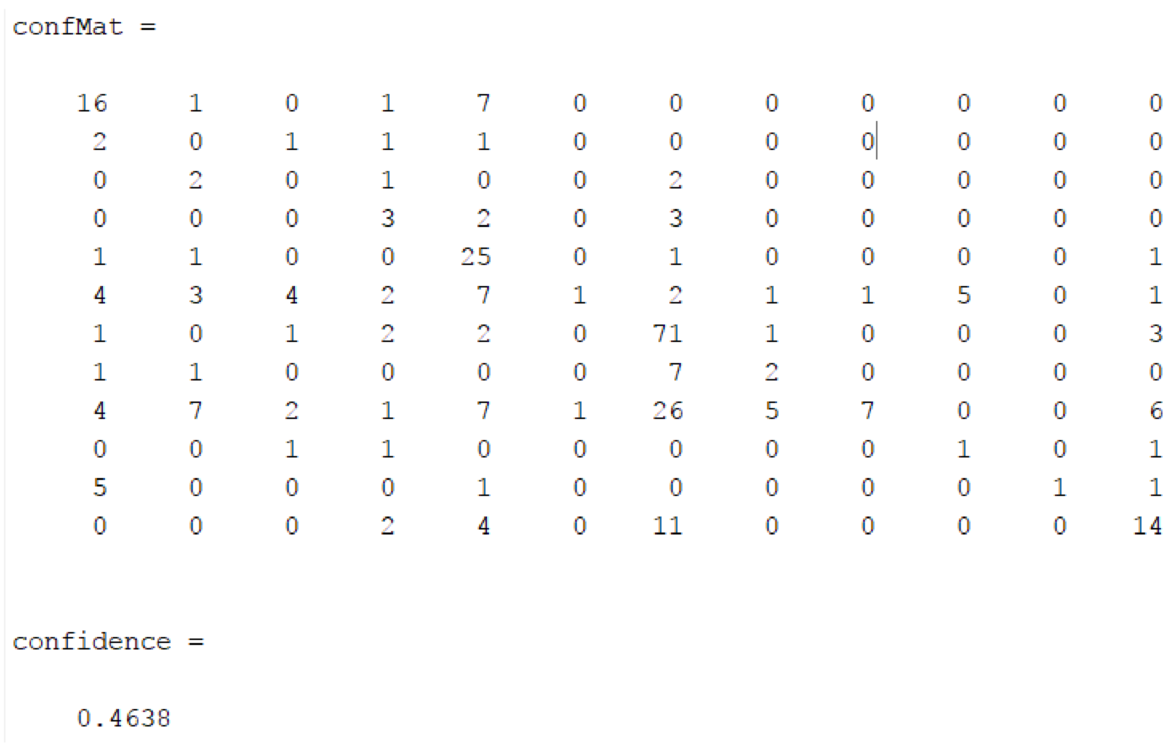


```
confMat =

    16     1     0     1     7     0     0     0     0     0     0     0
     2     0     0     1     2     0     0     0     0     0     0     0
     1     1     0     0     0     1     2     0     0     0     0     0
     0     1     0     1     1     0     4     0     0     0     0     1
     1     0     0     2    24     0     1     0     1     0     0     0
     4     8     2     8     3     1     2     0     1     1     0     1
     0     1     0     4     8     0    65     0     0     0     0     3
     0     0     0     0     0     2     8     1     0     0     0     0
     7     6     0     2     6     1    24     2     4     0     0    14
     0     0     0     0     1     0     0     0     0     0     0     3
     3     1     0     0     4     0     0     0     0     0     0     0
     0     0     0     1     7     0    10     0     0     0     0    13


confidence =

    0.4112
```

Note

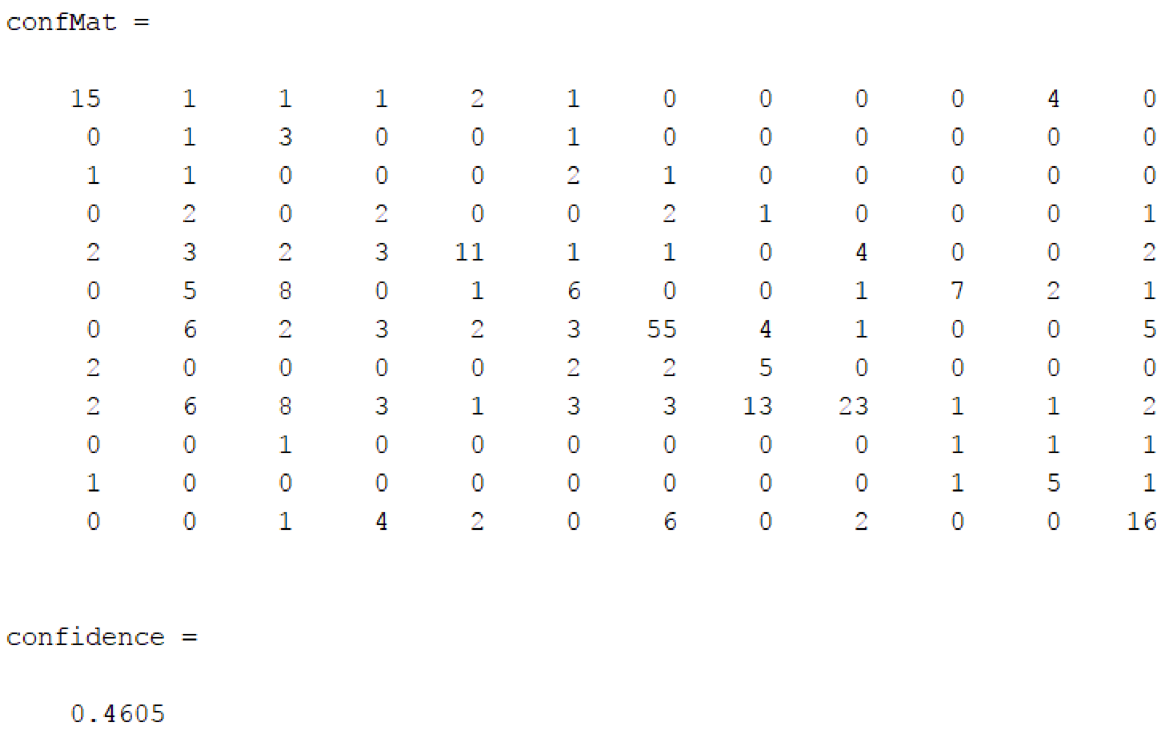
- Refer to **Table 1** for **column and row labels**.
- Refer to **Table 2** for **sample size** used in a test.
- Each viewpoint category consisted of car samples of alpha angle within the range of **plus/minus 10 degrees.**

Figure 2. Test 2 condition: All samples, and 4x4 HOG feature size



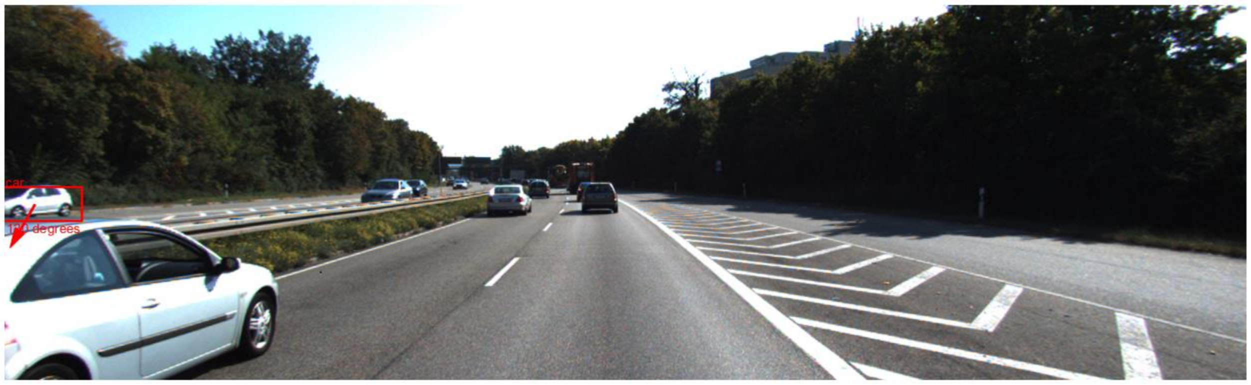
```
confMat =

    16     1     0     1     7     0     0     0     0     0     0     0
     2     0     1     1     1     0     0     0     0     0     0     0
     0     2     0     1     0     0     2     0     0     0     0     0
     0     0     0     3     2     0     3     0     0     0     0     0
     1     1     0     0    25     0     1     0     0     0     0     1
     4     3     4     2     7     1     2     1     1     5     0     1
     1     0     1     2     2     0    71     1     0     0     0     3
     1     1     0     0     0     0     7     2     0     0     0     0
     4     7     2     1     7     1    26     5     7     0     0     6
     0     0     1     1     0     0     0     0     0     1     0     1
     5     0     0     0     1     0     0     0     0     0     1     1
     0     0     0     2     4     0    11     0     0     0     0    14


confidence =

    0.4638
```

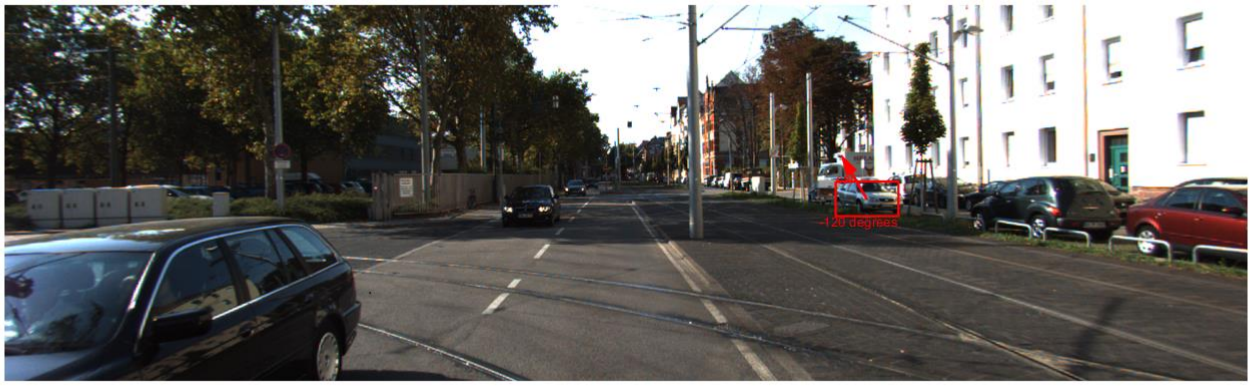
Figure 3. Test 3 condition: Equal sample size between classes, and 4x4 HOG feature size

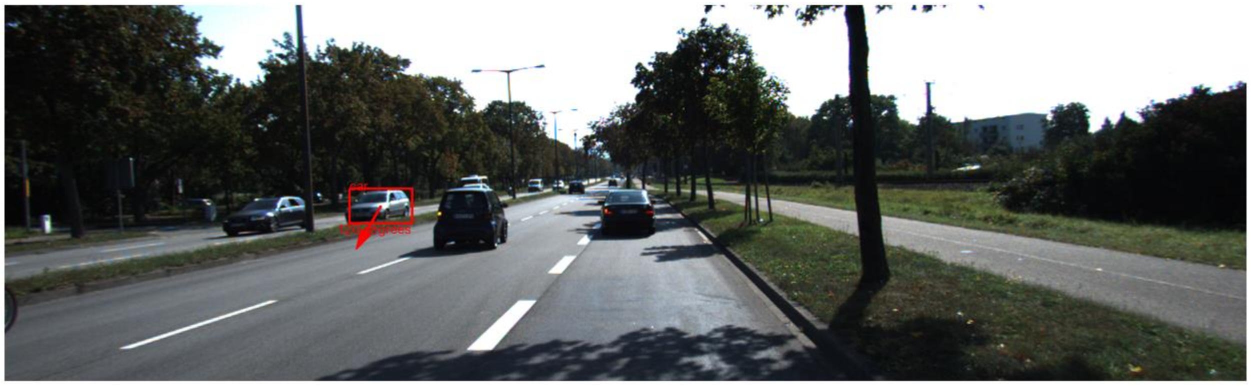


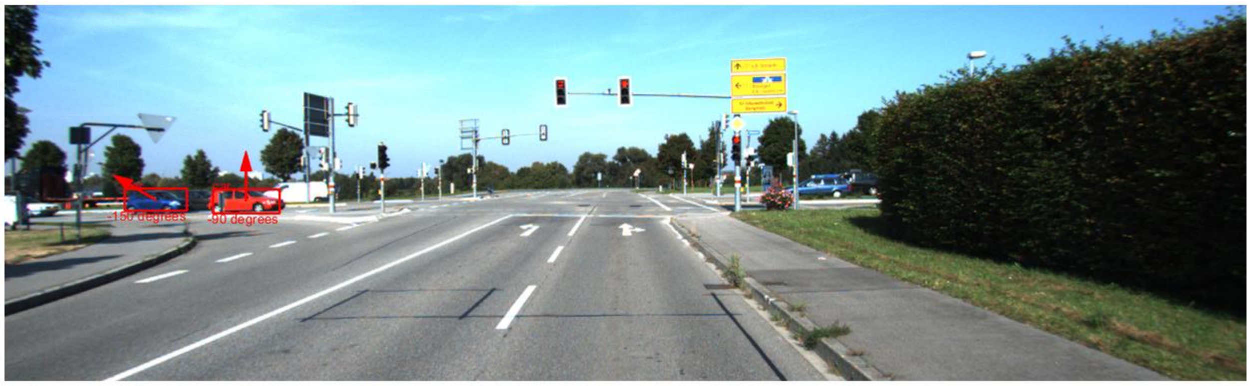
```
confMat =

    15     1     1     1     2     1     0     0     0     0     4     0
     0     1     3     0     0     1     0     0     0     0     0     0
     1     1     0     0     0     2     1     0     0     0     0     0
     0     2     0     2     0     0     2     1     0     0     0     1
     2     3     2     3    11     1     1     0     4     0     0     2
     0     5     8     0     1     6     0     0     1     7     2     1
     0     6     2     3     2     3    55     4     1     0     0     5
     2     0     0     0     0     2     2     5     0     0     0     0
     2     6     8     3     1     3     3    13    23     1     1     2
     0     0     1     0     0     0     0     0     0     1     1     1
     1     0     0     0     0     0     0     0     0     1     5     1
     0     0     1     4     2     0     6     0     2     0     0    16


confidence =

    0.4605
```

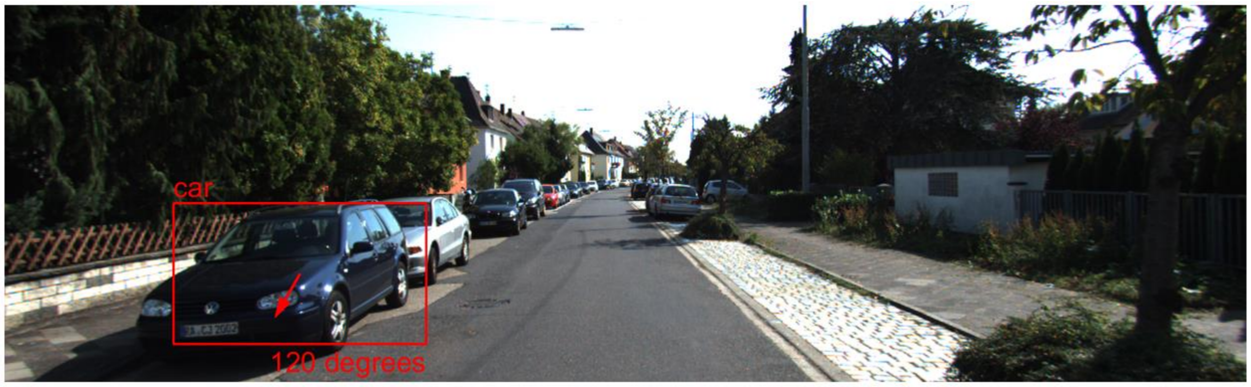
2.2. Directional arrows for Alpha Rotation

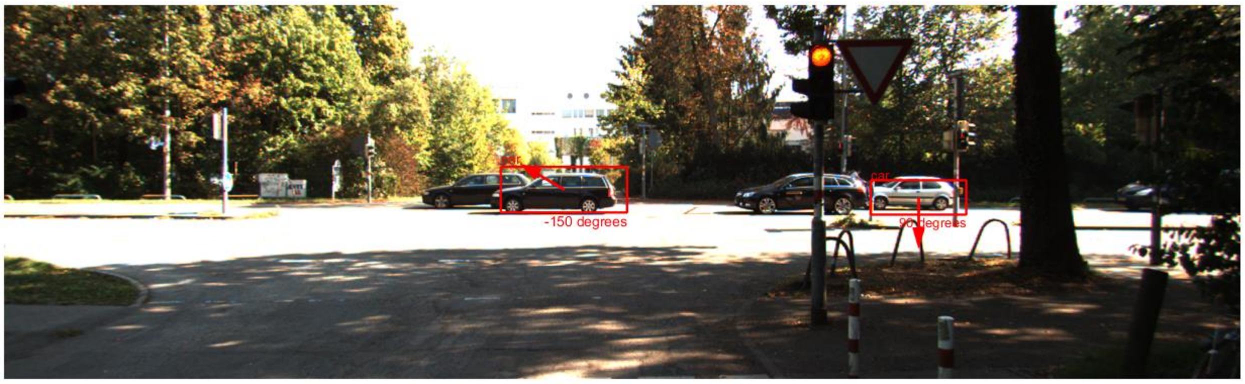







-150 degrees
-90 degrees


car
120 degrees


car
-150 degrees
car
90 degrees

2.3. 3D-bounding box (depth information only)

Figure 4. Segmented points of the detected object

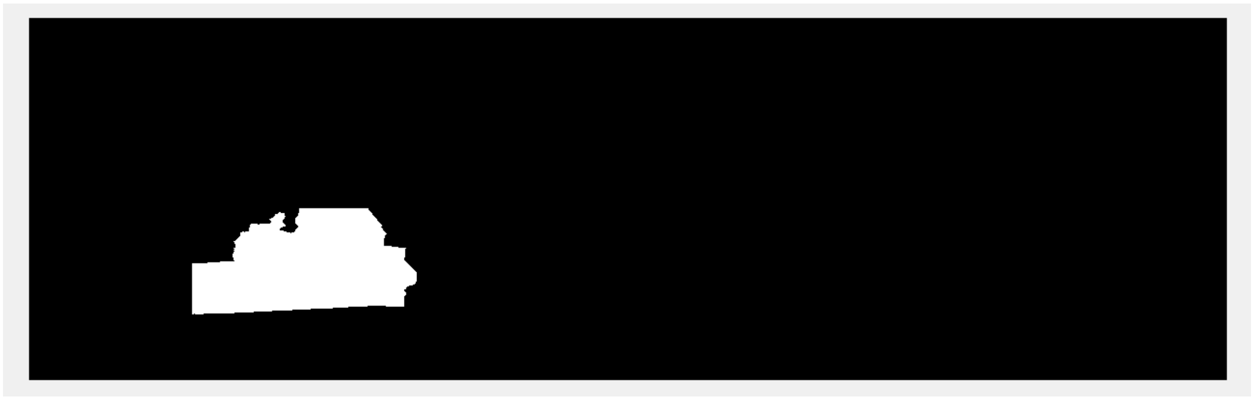
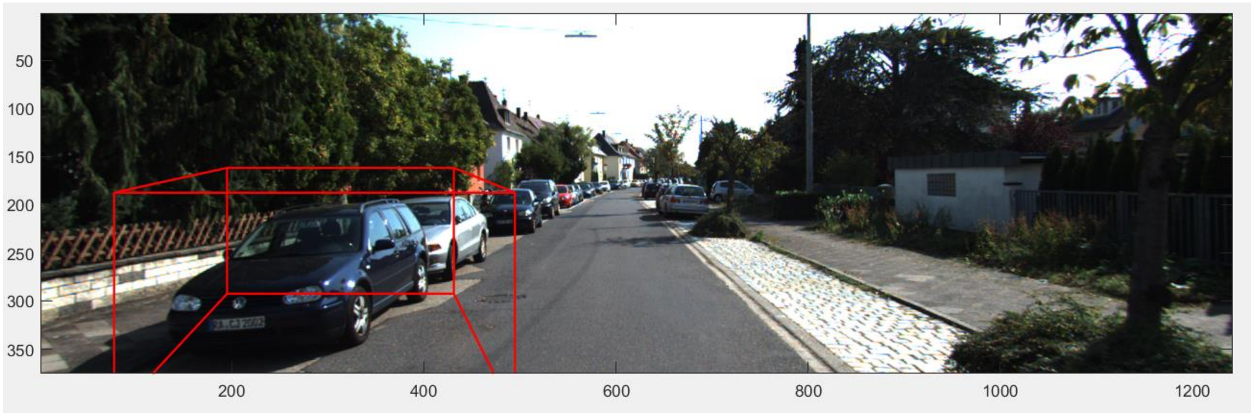


Figure 5. 3D-bounding box around detected car



Analysis

2.4. Viewpoint classification

As depicted in confusion matrices, viewpoint classes that are more prevalent in real life [high-lighted in Table 1; These are: -90 (front-view), 90 (rear-view), -120, and 120 (slighted tilted front-views)] had more visually accurate samples which in turn resulted in higher accuracy in classification. In addition, these classes made up 74.2% of total sample sizes in training which further contributed to better performance in classifying these classes than the less prevalent ones [0 (right side-view), 180 (left side-view)]. In conclusion, to improve the overall performance of viewpoint classification, training samples should include more accurate samples for lower performance classes.

2.5. 3D-bounding box

We were only able to estimate it using depth information because we could not compute the ground plane (solid parallelogram with four corners) in part 1. However, if ground plane were given, we could simply take the four corners of the plane to estimate the line that crosses the center point of detected car on image to find the furthest and closest points along that line. We could then place the two 2D bounding boxes (window size adjusted based on depth) on those two points and then connect them to get the 3D bounding box.

2.6. Appendix

Table 1. Categorical labels in confusion matrix

-120	-150	-30	-60	**-90**	0	**120**	150	180	30	60	**90**

Table 2. Sample size used for each test

	-150	-120	-90	-60	-30	0	30	60	90	120	150	180	Total
T1	800	1262	5715	1554	492	295	425	196	2433	2957	539	372	16668
T2	800	1262	5715	1554	492	295	425	196	2433	2957	539	372	16668
T3	196	196	196	196	196	196	196	196	196	196	196	196	2156