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 concepts 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

Methods

Part 1: (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;

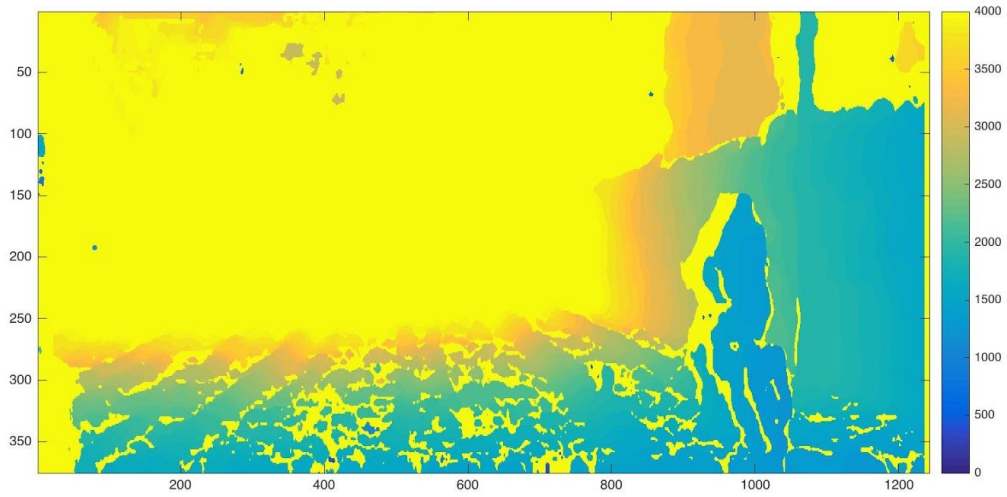

Figure 1.2: depth for 'train/left/000000.png' Image

- **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).

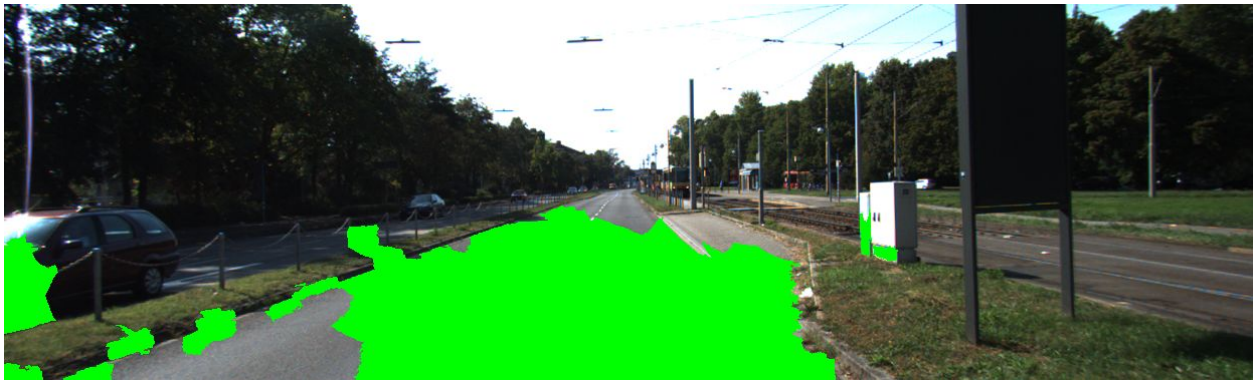

Figure 1.3: predicted road for 'test/left/000000.png' Image

- **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.

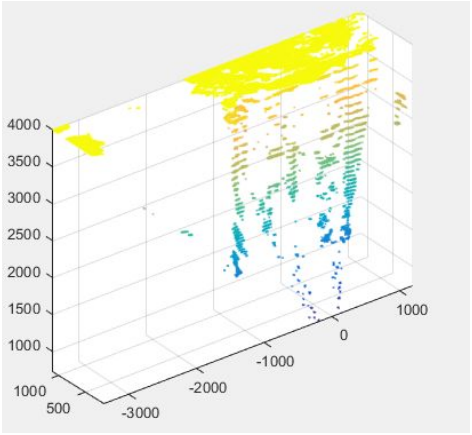

Figure 1.4: The cropped point cloud of the road pixels for 'test/left/000000.png' Image

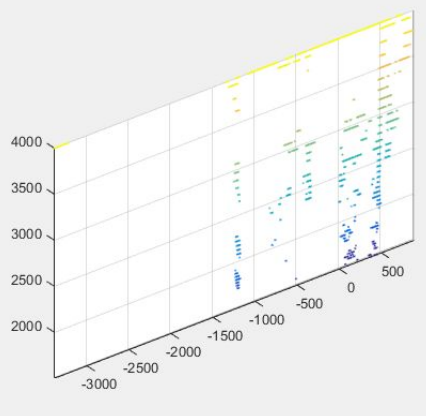

Figure 1.5: Inliers for ‘test/left/000000.png’ Image

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

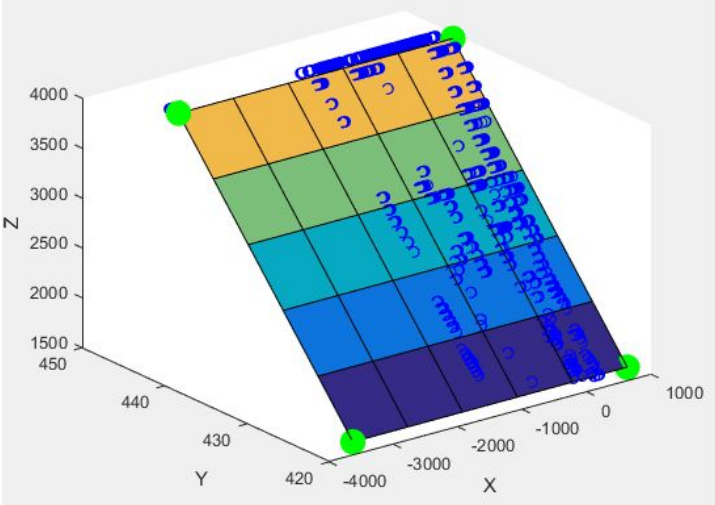

Figure 1.6: plane for ‘test/left/000000.png’ Image

Part 2

Overall logic and process

About 7000 training images (each depicting various objects like car, truck, pedestrian, building, etc) are used to crop-out car images for training car-models. Each image file is associated with a text file containing pre-detected object info (Table 1). The info included items in the following table.

Table 1. Pre-detected object info table (example)

type	truncated	occluded	**alpha**	**Bbox (x1, y1, x2, y2)**	Dimensions (h, w, l)	location	rotation_y
Pedestrian	0	0	**-0.2**	**712, 143, 810, 308**	1.89,0.48,1.2	1.84, 47, 8.41.1	-1.56

Procedure for gathering training samples

Cropped-out car images (total of 17,235 samples) were then saved into 12 viewpoint categories.
Good samples (high score samples judged by eyes) were then selected for training classifiers.
Figure 1. Examples of bad samples
Occluded, shadowed, partial, and inaccurate (labeled as -60 when closer to -30)
Finally, samples sizes were adjusted to meet the balance between classes. Following table summarize the logistics.

Table 2. Summary of training sample filtering process

viewpoints	-150	-120	-90	-60	-30	0	30	60	90	120	150	180	Total
# training	617	2002	39	228	1360	384	77	205	63	222	758	2284	17235
# selected	40	40	18	35	35	35	33	35	17	27	36	36	352

Note: Bad samples are due to the inaccurate (or low threshold) car-detector used by the data providers. Not filtering out low score images for training resulted in bad classifier which in turn resulted in low detection rate.

Procedure for gathering test samples
Using the given car-detector, car objects were detected in 531 test images. The detected corner values were used to crop-out the sample images and save them. Good samples were filtered by eyes. Test samples were then categorically named based on their expected-viewpoint classes by eyes.

Table 2. Summary of test sample filtering process

Expected viewpoints	-150	-120	-90	-60	-30	0	30	60	90	120	150	180	Total
# selected	25	5	20	66	81	31	8	4	31	5	8	27	311

Class training procedure
Variables: class sample size = x, total training samples N, total test samples T

Sample sizing rules chosen:

- N = sum([c1, ..., c12].*x)
- T = N

Logic

1. Image features were extracted from each sample using HOG. Feature size were set to be 3 by 3 (=3*3*9) so all samples would have the same number of features regardless of image size.
2. Total of 12 classifiers were generated where each classifier is trained with two unique labels: a positive label associated with the class samples and a negative label associated with all non-class samples.
3. Test sample features were classified against each trained viewpoint models to achieve a list of predicted labels.
4. Each generated predicted label was aligned with expected-viewpoint labels to evaluate the classifier's confidence.

Table 3: Classification Test #1 (front-view vs left-view)

viewpoints	-150	-120	-90	-60	-30	0	30	60	90	120	150	180	Total
# training	0	0	20	0	0	20	0	0	0	0	0	0	40
# testing	0	0	20	0	0	20	0	0	0	0	0	0	40
# positively classified	0	0	16	0	0	17	0	0	0	0	0	0	33
Confidence	0	0	0.8	0	0	0.85	0	0	0	0	0	0	0.825

Table 4: Classification Test #2 (4 viewpoints)

viewpoints	-150	-120	-90	-60	-30	0	30	60	90	120	150	180	Total
# training	0	0	18	0	0	20	0	0	20	0	0	20	78
# testing	0	0	20	0	0	20	0	0	20	0	0	20	80
# positively classified	0	0	6	0	0	11	0	0	9	0	0	6	33
Confidence	0	0	0.3	0	0	0.55	0	0	0.45	0	0	0	0.4

Table 5: Classification Test #3 (12 viewpoints; <= 10 samples each)

viewpoints	-150	-120	-90	-60	-30	0	30	60	90	120	150	180	Total

# training	40	40	18	35	35	35	33	35	17	27	36	36	352
# testing	10	5	10	10	10	10	8	4	10	5	8	10	100
# positively classified	2	0	0	2	0	0	2	0	0	0	2	0	8
Confidence	0.2	0	0	0.2	0	0	0.25	0	0	0	0.25	0	0.825

A pattern can be found by analysing the three tests above. Classification becomes more accurate with smaller number of classes. This is mainly because having more viewpoint classes led to smaller variations in features between classes. The accuracy of classification result depended on two factors: Firstly, the more proportionate the number of train data used between classes, the better. Secondly, although not certain, the number of test data and its expected-class ratio appeared to affect the outcome. It appeared that when test data number were closer to the total number of train data, the confidence value was higher. However, the most important factor seemed to be the expected-class ratio of test data. In order to get a higher confidence value, the ratio of test data between expected-classes needed to be closer to the ratio of train data between the classes.

Main Challenges

Part 1:

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.

Part 2:

The main challenge of classification problem was to be able to find the factors contributing to the classification results. Because the classification of test data depended on the controlled train data with respect to the expected classes and number of test data, it presented a problem when test objects needed to be classified separately. Having too many viewpoint divisions resulted in similar features between classes

Results

Part 1:

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.

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.

Part 2:

Figure 2-1. Computing segmented points of the detected object.

Figures 2-2. Computing 3D-bounding box around detected car.

Once computing segmented points of a detected object (from A4), depth of center point, averaged depths of the closest and furthest points in segmented pool were used to compute the ratio for adjusting boundary size. Then the boundaries were placed at the averaged closest and furthest points.

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.