CSC420 Fall 2016
Autonomous Driving Project

Mengdi Xu, Yichun Chen

Deadline: 3/12/16

1 Introduction & Related Work

According to Association for Safe International Road Travel, there are nearly 1.3 million people who die in road crashes each year, on average 3,287 per day, with an additional 20 to 50 million injured or disabled. Compared to human drivers, self-driving cars do not blink, do not get tired or sleepy, and do not text while driving; therefore, it easy to see machines being better drivers. As a result, it is not surprising that a lot of effort is devoted to improve the safety of autonomous vehicles by both academics and industry.

In this project, my partner and I are going to implement algorithms to detect road, cars, cyclists, and pedestrians in stereo images using computer vision and machine learning techniques, so that we can recognize and send out warnings of nearby objects. The work she has done is indicated in the following report, otherwise is done by me.

2 Methods

2.1 Data

In this project we are using road and car images from KITTI, which provides left, right images, calibration matrices, and the ground truths for road and objects separately.

2.2 Road Detection

We used several machine learning algorithms to perform road detection, including supported vector machine (SVM) , convolutional neural networks (CNN).

2.2.1 Detection

- **Inputs**
  We selected 400 pixels from each image, then concatenated them together as the input

of training set. Each observation is a 1×16 vector, which includes both 2D and 3D features (i.e. $[R, G, B, x_r, y_r, z_r, y_i, HOG]$):

In 2D: For each pixel, we included

- Color information in (R,G,B) space - since road are usually gray;
- Pixel location in the image (i.e. x_i, y_i) - only y_i is used;
- 9 HOG descriptors on each pixel. We modified the code here to generate the HOG descriptor. The modified algorithm takes an input image and returns values of 9 histogram bins for each pixel in the image. (Details can be found in *HOGdescriptor.m.*)

In 3D: Since there could be objects such as walls or poles, which may have similar color to the road, we would like to include the 3D real world location (i.e. x_r, y_r, z_r) of each pixel in our input for training. In doing so, me and my partner both came up with our own algorithms and we compared the outputs. Since our approaches are similar to each other, I will explain and display my results. The following steps are used to compute the real world location of each pixel, an example is shown below:

1. Original left and right images, and the camera calibration are given by the data-set

Left: left image; Right: right image

2. Find disparity of each image using left and right stereo images and the calibration of the cameras which is included in the data-set. Me and my partner computed the disparity using different approaches:

   My approach: I used the disparity map function for stereo images, which can be found here. However, the disparity map shown below does not look right, since some of the dark fragments on the ground are clearly miscomputed. I have tried to experiment on various of the parameters, but it did not seem to work well.

My partner's approach: She modified and used the Efficient Joint Segmentation algorithm, which can be found here. The output shown below makes more sense, and we will use the outputs from this algorithm in this project.

3. Find the depth of each pixel in the image using camera calibration and disparity: *(findDepth.m, findZ.m)* $z_r = \frac{f \times baseline}{disparity}$
4. Find the real world location of each pixel located at (x_i, y_i) in the image using the computed depths z_r: *(findDepth.m, findXY.m)*

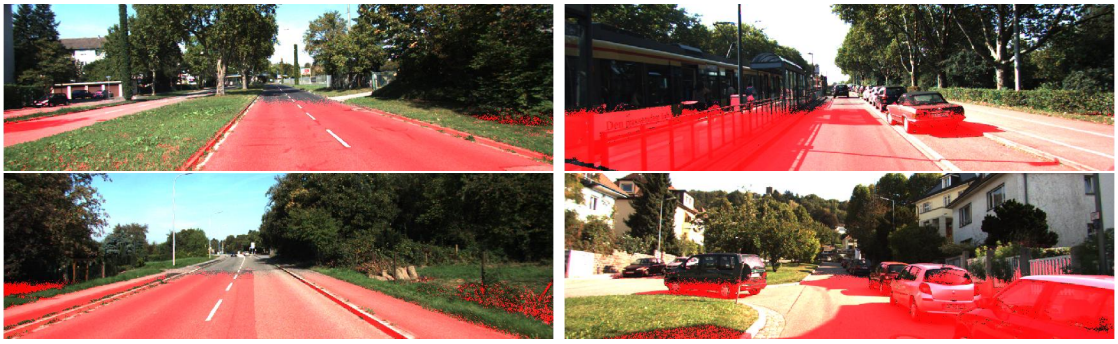
Other features we tried but did not use in the final model:

- We tried to include color information in (L*a*b) space - since there are shadows on the ground for some images. But since it contains the same information as RGB space does, it did not improve the accuracy. Thus, in the final model we did not include this.
- We tried to include SIFT and SURF descriptors on each pixel as training features, which obtains similar information as HOG does. Thus, we did not include them in the final model training process in order for the training process to be more efficient.

- **Targets**
  The targets used in training is an $N \times 1$ array, where N is the number of observations and each row is a binary value representing whether the observation is road or not. (Note: The ground truth provided in the data-set contains labels of road and lane separately. We used road as the targets in training for this project.)

- **Training Outcomes & Predictions**
  After training the SVM predictor, we saved the model and used it on the test set. Our test set contains $23,753,250$ observations from 51 full size images. The classification rate for SVM on test set is 88.97%. *(trainRoad.m & getRoadX.m)*

An example of the SVM predicted outcome from test set is:



Left: good prediction; Right: bad prediction

From the above examples, the road prediction works well on road detection on highways or roads that do not have many traffic. However, it does not work so well on segmenting cars or buses from the ground plane, as the bad examples showed. Fitting a road plane using the detected road pixels can help with this problem, since the height of the road is fixed at ground level for most of the times. Details about fitting road plane will be introduced later in this report.

2.2.2 Visualization

- In order to represent and fit the road in 3D, we computed the depth of each pixel in the image and computed the real world 3D location of each pixel, which gives the point cloud of the entire image (see equation (2) and (3) above). Since we only need the road point cloud in order to fit a ground plane, we cropped out the parts that are recognized as road from the previous detection process. Now, we have a 3D point cloud of the road-like parts.

- Since the road is always horizontal from the view point of the camera, we want to fit a horizontal ground plane to the road 3D point cloud. Three methods are used in order to do so in *(fitPlane.m)*:

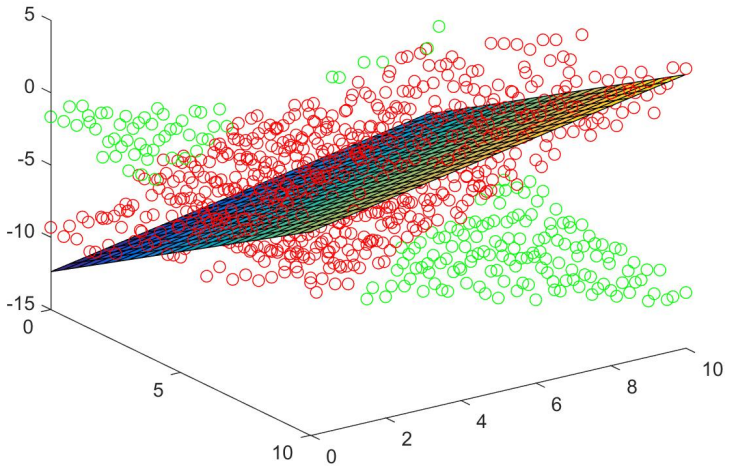
  Method 1: We implemented an algorithm *(dist2plane.m)* to calculate the sum of distances from each data point to a plane. Iteratively minimize the sum of distance using fminsearch. Then find a best fitted ground plane using RANSAC based on the loss function.

  Method 2: Since the loss function *(dist2plane.m)* is quadratic, we could compute the minimum points by using the closed form - affine fit. Then find a best fitted ground plane using RANSAC methods based on the loss function.

  Method 3: We used pcfitplane, which fits a plane to 3D point cloud. In order for the plane to be horizontal, we fixed the normal vector of the estimated plane to be [0 1 0] In addition to the fitted plane, this function returns a set of inliers and outliers. We

fit the plane to inliers for a second time to ensure that the plane is accurate. We do not know how the details about this process. Therefore, we came up with our own methods, but they did not perform as well as this.

Below is an example of plane fitting algorithm (see details in *testPlane.m*).



Data points are randomly generated following two different normal distributions. The inliers are marked in red and the outliers are green.

The detected road using the described method:

Above is the bounding box for road cropped from the ground plane.

2.3 Object Detection

2.3.1 Detection

- We used deformable parts model *(dpm.m)* to detected objects in the images. The locations of the 2D bounding boxes are saved to file for later use. (My partner ran the dpm and the results are saved to file.)

- Since deformable parts model are trained using 2D data, the predictions might include some objects appearing in some unusual locations in the image. Therefore, only the objects with y-value of 3D center of mass between ground and the camera are said to be valid (i.e. cars, cyclists, or pedestrians).

- Outputs

Above: good detections; Bottom: bad detections

The signs are sometimes recognized as people, because the deformable parts model thinks the round part of the signs are the heads of people and the height are usually similar to people.

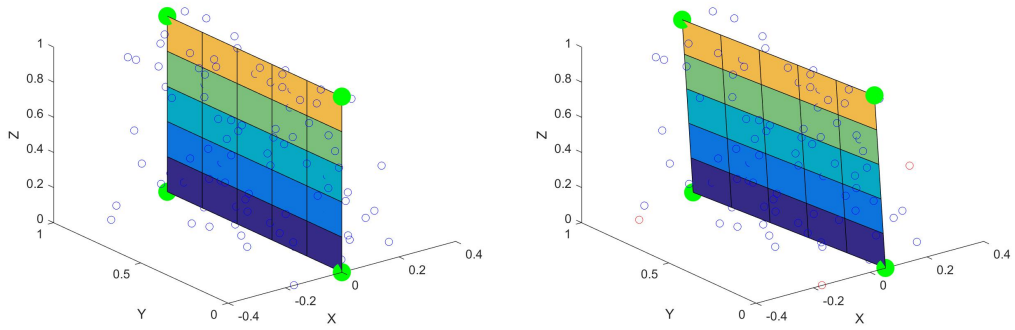
2.3.2 Visualization

- We computed the point cloud of each image and cropped out the local point clouds in each 2D bounding boxes obtained from the detection step *(findDepth.m)*. (This step is implemented by my partner.)

- After finding the 2D bounding boxes, we would like to calculate the dimension of each object in 3D. We had several approaches:

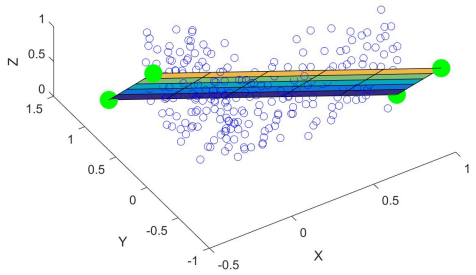
  Approach 1 Plane fitting:
  Since there is usually a surface of the object that is the closest to the camera, we tried to fit planes into the point clouds. We used *(ransac3d.m)*, which calls *(fitPlane.m)* to find the best fitted plane that is orthogonal to ground plane. This will give us a set of inliers and a set of outliers, and a plane that is represented by the normal vector. The outliers can be the background within the bounding box that does not belong to the object.

  – Fit an arbitrary plane to the point cloud:



The image on the left is the ground truth plane, the one on the right is the fitted plane. The inliers are marked in blue and the outliers are marked in red the fitted plane image.

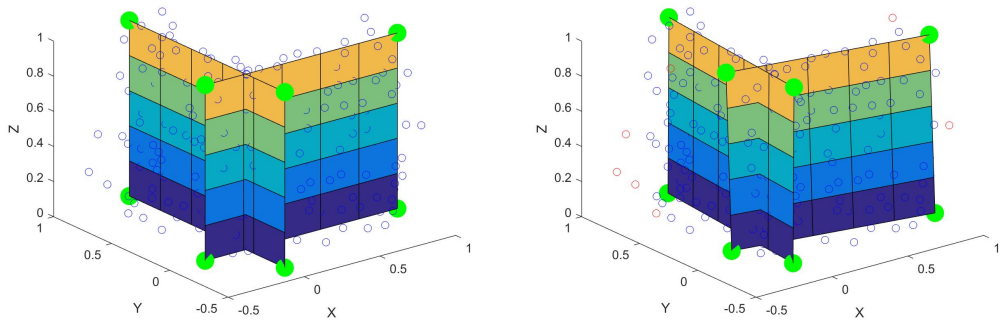
However, this method cannot be used in car detection. Because when there are two point clouds that are perpendicular to each other, the fitted plane might not be a part of the car. The following image demonstrates this case.



The image on the left is the ground truth plane, the one on the right is the fitted plane. The inliers are marked in blue and the outliers are marked in red the fitted plane image.

Therefore, we want to fit the point clouds by two planes that are orthogonal to each other, and both are vertical. This is more reasonable because for most of the cases, we can see two faces of the car.

– Fit 36 pairs of null basis of the normal vector of the ground plane by rotating the initial pair of basis by 10 degrees each time. Select the one with the most number of inliers.



The image on the left is the ground truth plane, the one on the right is the fitted plane. The inliers are marked in blue and the outliers are marked in red the fitted plane image.

This method works well theoretically in the demo which are shown above. However, it is not applicable on the data set. Because the point cloud we calculated does not provide as enough inliers as required by the pcfitplane algorithm. Therefore, a third method is approached.

Approach 2 Fix dimensions around center of mass
Since the cars, cyclists, and pedestrians are generally on the ground, and we restricted the y location (i.e. height of object) from the ground to be below the camera. This will make sure that the detected objects are on the ground. In addition to that, we segmented out the ground pixels and the pixels that are closest to the ground from the bounding boxes.

- After that we restricted the height and width of the fitted plane with inliers in order to find the boundary of closest surface of the object, *(drawPlane.m)* gives us two corners (top-left and bottom-right) of bounding rectangle in 3D. *(bbox3d.m)*.

- We set the default dimension of cyclist, car, and person to be [2 1 2]; [5 2 2]; [0.5 0.5 2] in meters, respectively. Based on the view point and the corners of bounding rectangle, we are able to construct the bounding boxes in 3D point cloud. Another approach which we used is to use a preset ratio of the three dimensions of bounding boxes in 3D, instead of the preset dimensions approach mentioned above. The example shown below is generated using the preset ratio approach

- Then, we projected the coordinates of the 8 corners of 3D bounding box back into our 2D image *(proj2photo.m)*, with the help of camera calibrations by left multiplying by the left camera projection matrix - P_{left}.

- Finally, we calculated the center of mass *(centersOfMass.m)* based on the information we obtained from above, which is done by my partner. Then generate the warning messages using the center of mass of each near-by object *(genText.m)* - I implemented

this code for assignment 4, which does something similar. My partner is working on generating the message warnings for this project and she is going to turn it in late. Please refer to her report for the outputs for this step.

An example of a detected cyclist:

First, detect the cyclist in the image and give a 2D bounding box in image coordinates. Then, find the real world coordinates of the 2D bounding box (imagine corners 1, 2, 3, 4 in the above image in 3D point cloud). After that, construct the other four points (corners 5, 6, 7, 8) in 3D point cloud to get the positions of the 3D bounding box in real world coordinates. Finally, project the 8 corners back onto the image and the resulting image is displayed above.

3 Main Challenges

While doing the segmentation of road and cars, there could be some other objects in the bounding box. For example, road is usually reconized in the bottom of car bounding box. In this case, we took out the road pixels by restricting the height y_r of the point in point cloud. Since the car cannot be lying on the ground. However, this might not be a big issue in practice since during the plane fitting process RANSAC will take care of the outliers. But being more accurate is always a good thing.

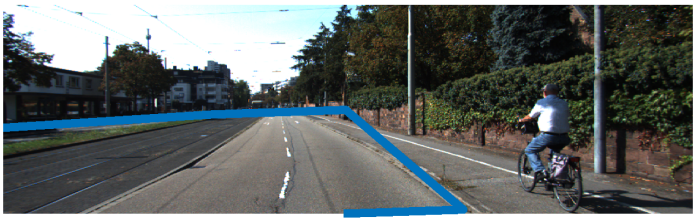
The greatest challenge we had in this project is we do not have enough time to experiment and implement all the ideas that we have. Researching and understanding the existing work from other academic paper is sometimes hard. Coming up with our own ideas is fun. We wish we had more time working on this project to improve the performance.

4 Results

In this project we achieved an 89% accuracy for road detection using SVM, and fitted ground plane based on the detected results. Then we drew the ground plane representing road in

images. In addition, we detected cars, cyclists, and pedestrians using deformable parts model. We found the 3D location and drew 3D bounding boxes around the point cloud of each object. Then we projected the 3D bounding boxes back into the 2D images for displaying. Finally, we computed the center of mass of each near-by object and displayed warning messages.

The detected road using the described method:



Above is the bounding box for road cropped from the ground plane.

The detected objects in 3D location:

5 Future Work & Discussion

There are many algorithms that we would like to experiment in the future, for instance:

- Since current observation should be similar to the observation made 0.5 seconds ago when we are driving, there should exist auto-correlation between images in a short time interval. The current predictions of near-by objects should not only base on the current image, but also the images within the previous seconds. Instead of treating each frame independently like we did in this project, if the video is sampled at higher frequency, we could set the threshold to a relatively smaller number to include more object-like regions. Then rule out the false positive examples using the predictions of the previous images.

- If we can get the frame from 0.5 second before, then we can involve frame difference methods to speed up the algorithm and improve the accuracy. If we know the velocity of our car then we can compute the relative direction and speed of our car to a near-by object. Then we can warn the driver to slow down relative to other vehicles instead of just giving their locations.

6 Code, Pictures & Videos

The algorithms, annotated photos, and videos can be downloaded from this link.

7 Reference

- Ankit Laddha, Road Detection and Semantic Segmentation without Strong Human Supervision. July 2016, Pittsburgh, PA 15213.
- Bihao Wang, Vincent Fremont, Sergio Alberto Rodriguez Florez. Color-Based Road Detection and Its Evaluation on the KITTI Road Benchmark. IEEE Intelligent Vehicles Symposium (IV 2014), Jun 2014, Dearborn, United States. pp.31-36, 2014.
- Koichiro Yamaguchi, David McAllester, Raquel Urtasun, "Efficient Joint Segmentation, Occlusion Labeling, Stereo and Flow Estimation" European Conference on Computer Vision (ECCV), 2014.