Classification of Road Images for Lane Detection

Mingyu Kim minkyu89@stanford.edu

Insun Jang insunj@stanford.edu

Eunmo Yang eyang89@stanford.edu

1. Introduction

In the research on autonomous car, it is important to extract as much information about the road as possible from various sensors on the car. Especially, it is extremely helpful to know the relative position of traffic lanes and the car, so that the car can be driven properly on a legitimate lane and not on the opposite lanes or bike lanes.

Stanford Autonomous Car group has collected laser map data and preprocessed them to produce road images in the bird's eye view. They have manually labeled lanes on the raw image files and we developed a system to recognize the position of lane lines using support vector machine to make the labeling faster. Because laser beams are not affected by changes in sunlight and shade, we can circumvent such problems of noise from light sources.

The overall task can be divided into two phases – pixel-wise classification and post-processing. First, we learned support vector machine that classifies each pixel into whether or not it is on a lane line. After that, post-processing steps are done to filter outliers and to come up with a cleaner output format.

The following sections will describe what feature representation we have used for classification and experimental results of the classification, and will explain what post-processing steps we used.

2. Datasets

A sample of the preprocessed road images is shown in figure1. As briefly mentioned in the introduction, the images are preprocessed from laser map data which was collected from the panoramic laser on top of the research car of the Stanford Autonomous Car group.

Images are processed to be in the size of 500-by-500 pixel and each pixel represents the intensity of laser beam reflected off of the corresponding surface. Because lane lines tend to give stronger reflections to laser beams than the road surface does, we could identify lane lines from the laser image by the intensity value.

Since only the intensity value of a pixel is given in the raw image file and no other high-level information is provided, we have manually labeled the pixels into lanes and non-lanes by drawing three-pixel-thick lines on top of the lanes

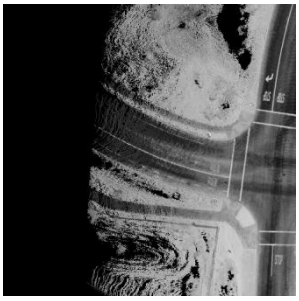

Figure 1 Sample road image

using image editing tools. We have been conservative in labeling lanes; if it was ambiguous to the human eye whether the given portion of the image is a lane or not, such parts were not labeled as lanes. Bike lanes were included as well as lanes. An example of labeling is shown in figure 2. Intuitively, the black pixels represent the lanes and the white pixels represent the non-lanes.

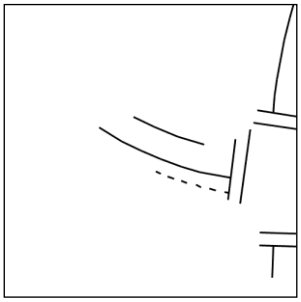

Figure 2 Sample road lane labeling

Next section describes how we extracted examples for classification from the laser images and the labeling.

3. Feature representation

We followed the following steps to create examples from the laser map images.

Acknowledgement

This work has been possible with advice from Quoc Le, Mike Sokolsky, Jesse Levinson and Jiquan Ngiam, and data from Stanford Autonomous Car Group.

1. Canny edge detection
2. Max-neighbor processing
3. Rotation alignment and averaging intensity value

3.1 Canny edge detection

Because an edge is a great indicator of a lane line, we performed canny edge detection on the original laser map images. The sample result is shown on figure 4. Shankar Sastry's matlab implementation of canny edge detection [1] was used.

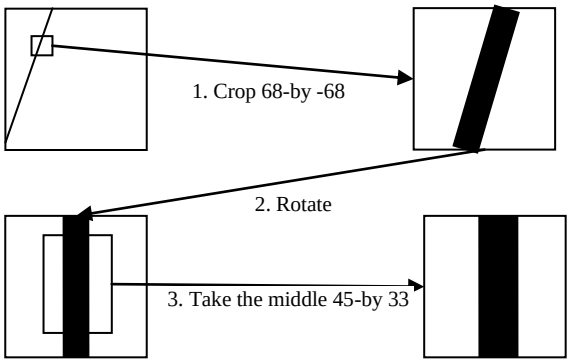

Figure 3 Overview of feature formation

Figure 4 Output of Canny edge detection

3.2 Max-neighbor processing

After canny edge detection, there were lots of edges on non-road portion of the image, which could be interpreted as lanes. To solve noise issues within the image, the image was processed such that each pixel value was reassigned to be maximum intensity value of its neighbors. In other words, we treated a pixel as an edge point if itself or any of its 8 immediate neighbors was determined to be an edge point by Canny edge detection. Non-road areas were then shown to be a clump of edge points and lanes were shown as a clearer line, which made it easier to separate out edges in non-road areas from true edges.

3.3 Rotation alignment and averaging intensity value

We decided to use 45 by 33 pixels surrounding each pixel to form an example. We started with a larger area (i.e. 68 by 68 pixel surrounding each pixel) so that rotation doesn't affect the middle 45 by 33 pixels of the box. The following explains the steps to form an example for one pixel.

1. Crop out the surrounding 68-by-68-pixel box from both the original intensity map and the max-neighbor-processed Canny edge image. Call them I and E respectively.
2. Compute the rotation angle of the box by averaging the gradient values from Canny edge detection of the middle 7-by-7 points.
3. Rotate I and E by the rotation angle computed at the step 2 so that the edge is aligned vertically. Call the rotated images I_{rot} and E_{rot} respectively.
4. Crop out the middle 45-by-33 box from I_{rot} and E_{rot}. By using sliding window of size 3-by-3 over I_{rot} without overlaps and averaging each window, get the average intensity matrix I_{rot_avg} of size 15-by-11.
5. Concatenate I_{rot_avg} and E_{rot} to generate a feature representation of 1650-dimension.

Cropping the middle 45-by-33-pixel is done because we are only interested in the middle part where the edge lies. Also, averaging the intensity values is done because intensity values do not have as much of information as the edge points and averaging them will make it more robust to noise.

The following section will describe the classification results using the feature representation described above.

4. Classification experiments and results

4.1 Balancing positive and negative examples

Because our dataset is highly biased with only 5% of the examples being positive, we had to adjust the weight of SVM on each of the labels. Following the general approach, we set the weight of negative examples to 1 and the weight of positive examples to the ratio of the number of negative examples to that of positive examples. However, this approach over-penalized misclassifying positive example to produce a great number of false positives. Because our ultimate goal is to find where the lane lines are, not to perfectly classify all the positive examples, high precision is preferred over high recall. Thus, we decided to penalize less on false positives by multiplying a scalar to the previous weight. Decreasing the

weight on positive examples definitely helped increasing precision and the result is shown on figure 5.

By plotting the results, we have concluded that the weight factor 0.3 has a good balance of precision and recall so that it removes a good number of false positives without missing too many positive examples. The plots of the results are shown on figure 6.

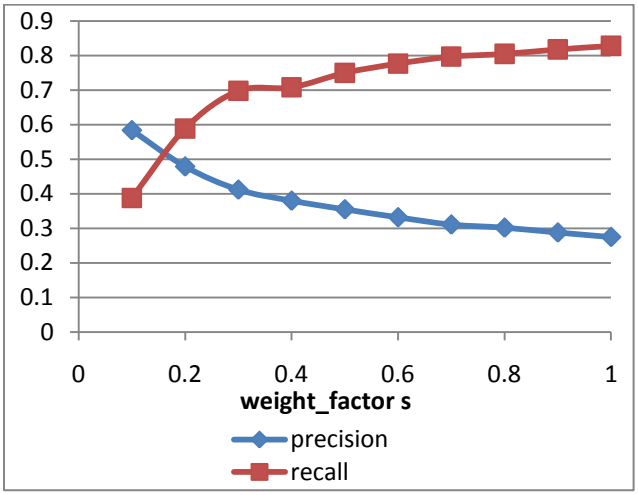

Figure 5 Precision and recall with various values of weight factor

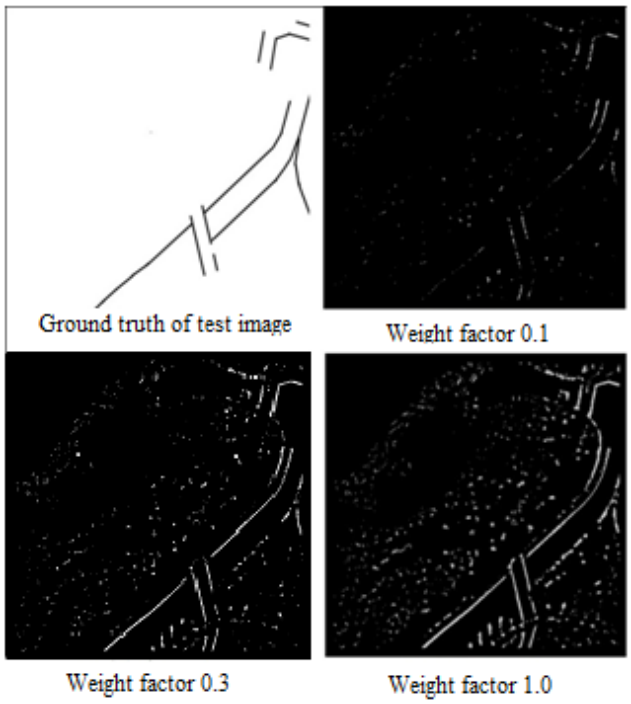

Figure 6 Plots of classification results and ground truth of test image

4.2 Choosing classification method

The classification library we worked with, LIBLINEAR [3], offered various options to classifying, so we have tried both SVM (with no kernels) and linear regression. The plots of the results are shown on figure 7. Generally, SVM had better precision and linear regression showed better recall. As stated above, we put our preference in high precision over high recall, so SVM was chosen over linear regression. In sum, we ran the classification with linear SVM with L2 regularization, and used weight factor 0.3 for all the ensuing results.

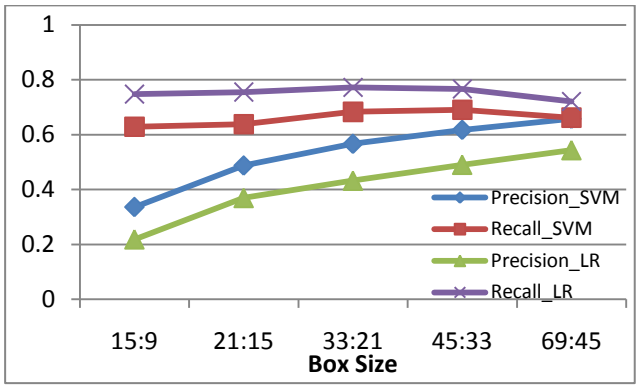

Figure 7 Precision and recall by SVM and logistic regression

4.3 Choosing box size

Originally we experimented with a box size of 21 by 15. The choice was arbitrary and quite small for fast prototyping. We strived for an optimal box size, and we tried to maintain the elongated rectangle box structure. We have tried five sizes, each one approximately double the previous one: 15 by 9, 21 by 15, 33 by 21, 45 by 33, and 69 by 45. The results are shown on figure 8. Box sizes smaller than 15 by 9 would have done poorly and box sizes larger than 69 by 45 was computationally impractical since each feature matrix would have been larger than 1GB. Although box size 69 by 45 had better precision, it started showing decrease in recall, and we thought over-fitting had occurred. So for the following results, we have used a box size of 45 by 33.

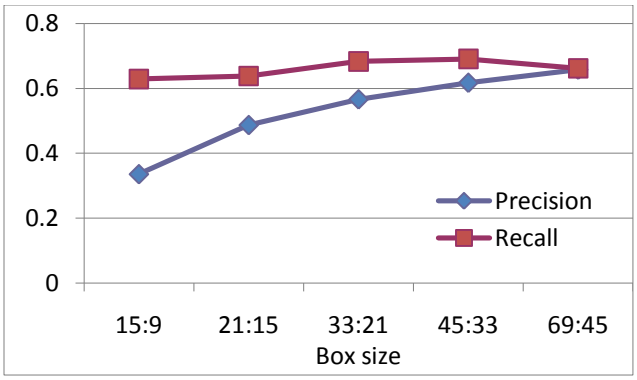

Figure 8 Precision and recall with different box sizes

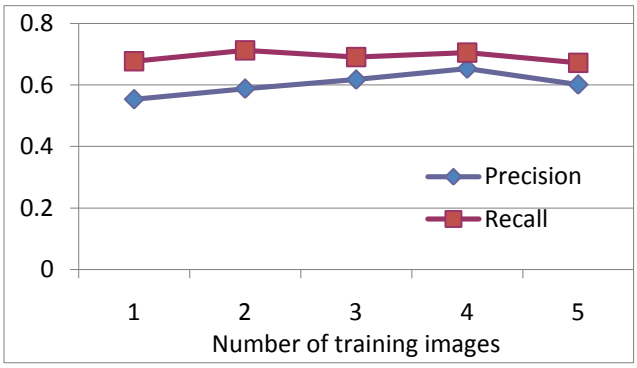

Figure 9 Precision and recall with different number of training images

4.4 Choosing training size

For the experiment whose data is plotted in figure 8, we have tested on 3 images and tested on 1. Because we only identified pixels as feature points if and only if it had an edge point in its immediate neighborhood, each image had nearly 100,000 feature points, and we thought 3 was a large enough number. After prototyping, we tried testing out different training set sizes. The results are displayed in figure 9. With five training images, we start to see decrease in both precision and recall, so we assumed that over-fitting started to take effect. Therefore we chose the training set of 4 images as our optimum.

To summarize, we used linear SVM with L2 normalization as the classification method, chose 45*33 as the box size and trained 4 images. The sample output is shown in figure 10. Pixel value is the probability to be on the lane line if positively classified, and 0 otherwise.

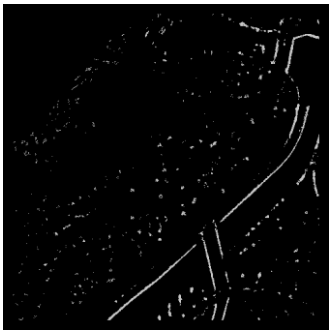

Figure 10 Sample output from pixel-wise classification

5. Post-processing

Pixel-wise classification result contains fair number of false positives, and we developed the following post-processing steps to filter out false positives and to output cleaner lane lines.

1. Direction estimation
2. Non-maximum suppression
3. Clustering
4. Cluster discretization

5.1 Direction estimation

We first estimated the direction of the lane lines for each of the positively classified pixels by linear regression. For each pixel, we used 21-by-21surrounding box to run linear regression without intercept terms. 21-by-21 box is small enough to assume lane lines are straight in the box. We replaced the pixel value with the sum of probabilities of all positively classified pixels within 2 pixels from the learned line. Pixels with the sum-of-probabilities value less than 30% of the maximum sum-of-probabilities value, because such pixels are likely to be false positives.

5.2 Non-maximum suppression

As can be seen from figure 6, detected lane lines are more than one pixels wide and non-maximum suppression is performed to attain cleaner outputs with each lane line at most 1-pixel wide. This is the same step as the one in the Canny edge detection algorithm with normal vector to the estimated slope from 5.1 as the direction of gradient and the sum-of-probabilities value as the norm of gradient. This step suppressed all the pixels except for the one-pixel wide lane lines.

5.3 Clustering

For better applicability, we clustered the lane line points from 5.2. We clustered two points into the same cluster if they are less than 20-pixel apart and the angle between the two corresponding slopes are smaller than 20 degrees. This step allowed us to differentiate separate lane lines

and to drop out more false positives by ignoring clusters with less than 30 pixels.

5.4 Cluster discretization

For better applicability, it is important to output lanes by clear lines rather than clumped lines of pixels. For each cluster of pixels, we selected starting and ending pixel and in-between pixels that are separated by certain distance. Then, we output each cluster, which corresponds to one lane line, as a sequence of several points so that the clear lane line can be reconstructed by just connecting the points in order.

6. Final results

After post-processing steps, each lane is represented as a sequence of points and it can be reconstructed by connecting the points in order. The sample lane line detection results are shown below.

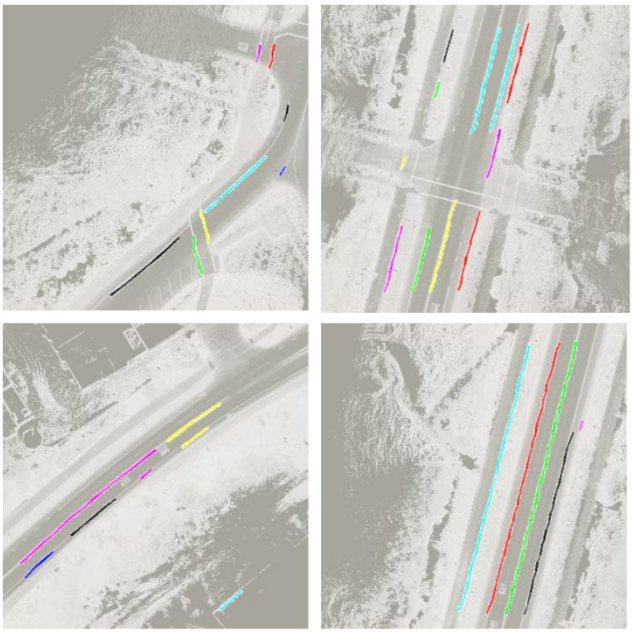

Figure 11 Sample results

7. Discussion

As can be seen from figure 11, we successfully found some of the lane lines but there is room for further improvement.

- Better post-processing steps: We see that output lanes are sometimes disconnected or there are undetected portion of lines. In this work, we haven't utilized the fact that lane lines are approximately parallel to each other and lane lines usually form a long straight line. We can possibly taking into account of such higher level knowledge to improve the performance of post-processing steps.
- Better roadmap image: There is room for improvement in the pre-processing steps of laser roadmap image. Higher-quality input images from better calibration and pre-processing steps will definitely help improving the performance.
- Lane classification: The lane we identified doesn't tell much about their functions. For example, we treat output lines all equally, such as boundaries between roads and non roads, centre lines, dotted lines, or cross-roads. It will be more useful if we differentiate preprocessing procedure of the raw image by lane functionalities and run our lane detection algorithm on each to detect each lanes. For example, for boundaries we will focus on preprocessing to discriminate from non-road(white) region to road(black) region in the image; after detecting boundaries of roads, we can detect centre lines by focusing on the road region only; we can detect crossroads or stop lines by focusing on intersections.

8. References

[1] "Canny Edge Detector Algorithm Matlab Codes." *UC Berkeley Robotics and Intelligent Machines Lab Home Page*. Web. 14 Nov. 2010. <http://robotics.eecs.berkeley.edu/~sastry/ee20/cacode.html>.

[2] Canny, J., *A Computational Approach To Edge Detection*, IEEE Trans. Pattern Analysis and Machine Intelligence, 8(6):679–698, 1986

[3] R.-E. Fan, K.-W. Chang, C.-J. Hsieh, X.-R. Wang, and C.-J. Lin. LIBLINEAR: A library for large linear classification Journal of Machine Learning Research 9(2008), 1871-1874.