Self Lane Assignment Using Smart Mobile Camera For Intelligent GPS Navigation and Traffic Interpretation

Tianshi Gao TIANSHIG@STANFORD.EDU
Stanford University

1. Introduction

Imagine that you are driving on the highway at 70 mph and trying to figure out which lane should follow to exit in front. The precision of the current GPS cannot tell on which lane you are on, so the instruction given by the GPS is just as simple as "keep right" resulting in the driver's panic caused by searching from multiple signs. However, if we have a smart camera mounted inside the vehicle which is capable of inferring the current lane the vehicle is on and feeding this information into GPS, then more intelligent instructions like "stay on the current lane" or "turn to your next right lane" can be achieved.

Another potential application using smart mobile camera is to estimate the speed and density of the vehicles surrounding a prob vehicle with camera mounted inside. As a result, real-time traffic status can be inferred from data collected by actively running vehicles instead of limited static loop detectors beneath the road. Moreover, if the prob vehicle knows which lane it is on, then lane by lane traffic flow can be obtained by reporting the self lane number of the prob vehicle.

In this report, I proposed a novel and very effective algorithm consisting of both computer vision and machine learning techniques to solve the problem which is defined as *given an image taken by a camera mounted inside a vehicle, infer on which lane the vehicle is.* I form this problem as a scene classification problem where different classes correspond to different scenes seen from different lanes as shown in Figure 1. In the proposed algorithm, horizon is detected, a set of features at different positions on the image is obtained from a filter bank consisting of oriented steerable filters at two scales and finally three different learning algorithms are applied to train the classifiers and the results are compared. For each class, both the precision and recall rates are around or above 90%.

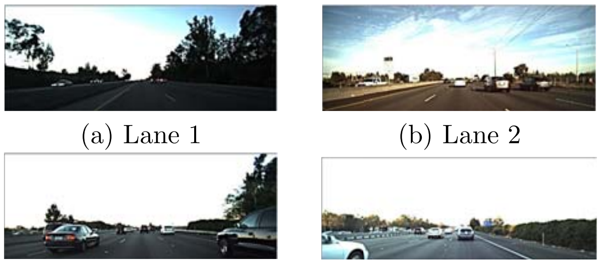

(a) Lane 1 (b) Lane 2
(c) Lane 3 (d) Lane 4

Figure 1. Sample images from four different classes

2. Approach

2.1. Problem Formation

In this work I focused on the highway situation where lanes exhibit low curvature and the number of lanes is four which is given as prior information. In practice, this is a reasonable assumption because we can get the number of lanes from the GPS/digital map and four lanes is a very common case.

Since perfectly detecting/fitting all lanes is unreliable or even impossible due to occlusion caused by other vehicles, I tried to find a global representation of the image directly from low-level features like oriented edges and infer the self lane number by the overall appearance. Therefore, we can think of this problem as a scene classification problem in which given a test image we want to classify it into four categories corresponding to four scenes seen on lane 1, 2, 3 and 4 respectively (lane 1 corresponds to the leftmost lane, and others are indexed from the left to right).

The scene classification problem considered here is different from the traditional one which is about classifying images into very broad categories like office, highway, forest, mountain, etc. Therefore, the problem here is more challenging in terms of understanding the scenes in a finer granularity. It is also different from object recognition which I'd like to bypass, since each object detector for the lane marker, vehicle and road will have certain error rate and it's very likely that errors from each detector will accumulate at later stage.

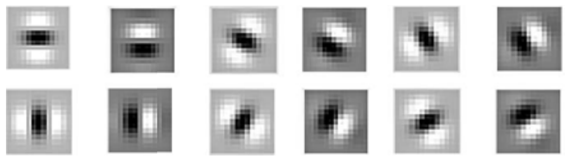
2.2. Feature

2.2.1. Feature Design

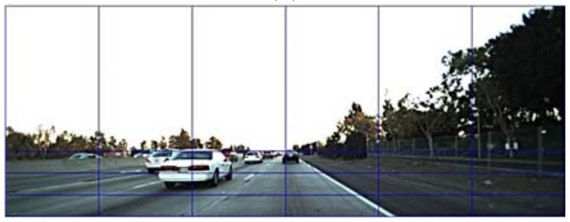
Based on the intuition of how people distinguish different lane situations, there are three desired properties for a good feature design to capture:

- *Lane markers.* Lanes are separated and defined by lane markers, so they are the most discriminative information.
- *Vehicles.* If there are multiple vehicles on your right, then it's less likely that you are on the rightmost lane.
- *Spatial distribution.* Besides the presence of lane markers and vehicles, the more important information is how they are spatially distributed on the image plane.

To capture the information above, as shown in Figure 2(a) a filter bank consisting of oriented steerable filters is used to implicitly capture the textures for both lane markers and vehicles. Due to perspective projection, the scales of edges at different positions on the image plane varies, so I used two sets of 12 filters which are at two different scales. To keep the spatial information, I partitioned the image below horizon into multiple cells which is shown in Figure 2(b).



(a)



(b)

Figure 2. (a) Filter bank consisting of 12 oriented steerable filters per 30 degree with even and odd phases (b) Spatial partition below horizon

2.2.2. Feature Representation

Denote the filters as $F_{k,m}$ where $k = 1, 2$ corresponding to two sets of filters with different scales and $m = 1, 2, \cdots, 12$ corresponding to each filter turned to different angles with even or odd phases. In the experiment, I've partitioned the images into $M \times N$ cells, where $M = 3$ and $N = 6$. Let the grayscale image be I and then the response of the image to each of the filter is incorporated into:

$$x_{i,j,k,m} = \sum_{(x,y)\in C_{i,j}} |(I * F_{k,m})(x, y)| \qquad (1)$$

where $C_{i,j}$ corresponds to the cell at ith row and jth column and $i = 1, \cdots, M$ and $j = 1, \cdots, N$. In order to make the response less sensitive to the illumination and contrast of the image, I normalize the 12-dimensional vector $\mathbf{x}_{i,j,k,\cdot} = [x_{i,j,k,1},\ x_{i,j,k,2},\ \cdots,\ x_{i,j,k,12}]^T$ to have energy 1, i.e., $\|\mathbf{x}_{i,j,k,\cdot}\|_2 = 1$. But if the energy of $\mathbf{x}_{i,j,k,\cdot}$ is too small, I'll not normalize it in order to capture those uniform road regions. Furthermore, three statistics used in [1] of the response for a set of 12 filters within one cell are also incorporated into the feature vector. These three statistics include mean, argmax and max−median of the components of $\mathbf{x}_{i,j,k,\cdot}$. So each cell has a $(12+3)\times 2$ dimensional descriptor, and all the 3×6 descriptors are stacked into a single $15 \times 2 \times 3 \times 6 = 540$ dimensional feature vector for each image.

2.3. Horizon Detection

As mentioned in the above subsection, the image is partitioned into multiple cells below horizon. I detect the horizon by detecting the vanishing point in two steps as follows:

1. *Lone line detection.* I used the same method mentioned in [2] to detect lone lines in the image and filtered out those nearly vertical or horizontal lines to reduce outliers. Denote the number of lone lines after filtering as L and these lines are parameterized by θ_i and r_i as follows:

$$x \sin\theta_i + y \cos\theta_i = r_i \qquad i = 1, 2, \cdots, L \qquad (2)$$

2. *Robust fitting in Hough domain.* Since the vanishing point is located at the intersection of the L lines, we can get the estimation of the vanishing point by minimize the norm of the residual:

$$\textit{minimize} \quad \|A\mathbf{x} - \mathbf{r}\|_1 \qquad (3)$$

where,

$$A = \begin{pmatrix} \sin\theta_1 & \cos\theta_1 \\ \sin\theta_2 & \cos\theta_2 \\ \vdots & \vdots \\ \sin\theta_L & \cos\theta_L \end{pmatrix} \qquad \mathbf{r} = \begin{pmatrix} r_1 \\ r_2 \\ \vdots \\ r_L \end{pmatrix}$$

The reason I used 1-norm instead of ordinary least square is to make the estimation less sensitive to outliers. This convex optimization problem (3) is solved using `CVX` [3]. Figure 3 shows some detection results.

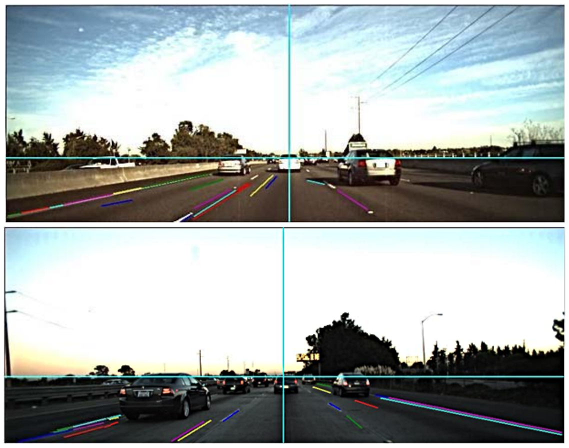


Figure 3. Sample horizon detection results. Color lines are the detected long lines and the cross is the detected vanishing point.

2.4. Learning Algorithm

Each dimension in the feature vector doesn't provide equal information. For example, those features from the cell that is in front of the vehicle contain little information. To avoid overfitting, i.e., reducing the model complexity, I have chosen three different learning algorithms:

1. *Adaboost with decision trees.* I used logistic regression version of Adaboost [1, 4] with weak learners based on decision trees. Decision trees make good weak learners, since they provide explicit feature selection and limited modeling of the joint statistics of features.

2. *Bayesian logistic regression.* By assuming a prior on the coefficients in logistic regression, Bayesian logistic regression is capable of shrinking the coefficients to avoid overfitting. The parameters can be learned by MAP and I used zero mean Gaussian prior and implementation from [5].

3. *SVM.* Although SVM doesn't provide explicit feature selection or shrinkage, it's still possible to have relatively low error rate. I used the [6] implementation with linear kernel.

Table 1. Horizon Detection Error Rate

mean of relative error	std of relative error
1.33%	3.85%

3. Experiment and Result

3.1. Data

I sampled frames from 12 short sequences and the numbers of images collected for lane 1 to lane 4 are 112, 500, 750 and 750 respectively. Since the dataset is unbalanced, I set the initial weights for Adaboost according to the proportion of the number of images for each class, and also adjust the penalty parameters of relaxation for different classes in SVM according to the ratio between the numbers of different classes.

3.2. Horizon Detection

I estimated the horizon position for all the 2112 images, and Table 1 and Figure 4 show the detection result. Around 98% of the detected horizon lies in the 5% relative error band. The mean of the relative error is only 1.33% which is 3 pixels in this case.

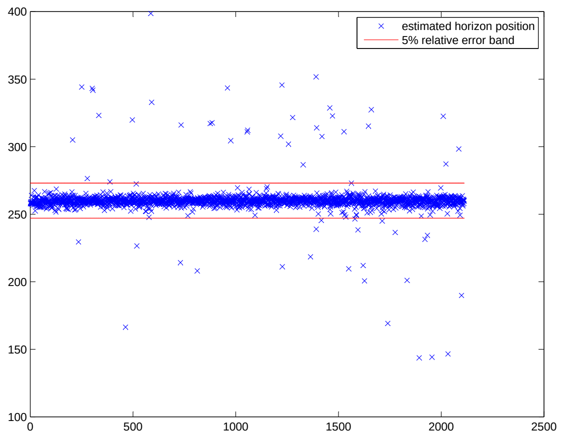


Figure 4. Estimated horizon positions.

3.3. Classification

The classifier for each class is trained in one vs. all fashion. At the test stage, the classification result is chosen as the one with the highest probability. I used 3-fold cross validation to evaluate the performances of three learning algorithms. The fold number is 3 instead of most common 5 or 10 is because I want to reduce the correlation between the training and test data.

The confusion table for each algorithm is shown in

Table 2. Confusion Table for Adaboost with Decision Trees

	Lane 1	Lane 2	Lane 3	Lane 4	Recall
Lane 1	109	3			97.32%
Lane 2	2	435	53	10	87.00%
Lane 3		36	688	26	91.73%
Lane 4		8	43	699	93.20%
Precision	98.20%	90.25%	87.76%	95.10%	

Table 3. Confusion Table for Bayesian Logistic Regression

	Lane 1	Lane 2	Lane 3	Lane 4	Recall
Lane 1	110	2			98.21%
Lane 2	3	464	30	3	92.80%
Lane 3		23	689	38	91.87%
Lane 4		3	40	707	94.27%
Precision	97.35%	94.31%	90.78%	94.52%	

Table 2, 3, 4. All three methods give comparable results, and Bayesian logistic regression has slightly better accuracy for lane 2 and lane 3. In general, the precision and recall rate for each class is around or above 90%. In addition, the results are consistent with the intuitions in terms of: i) lane 1 and lane 4 have better accuracy than lane 2 and lane 3; ii) lane 2 and lane 3 are more likely to be confused by each other; iii) and lane 1 is more likely to be confused by lane 2 than lane 3,4, and lane 4 is more likely to be confused by lane 3 than lane 1,2.

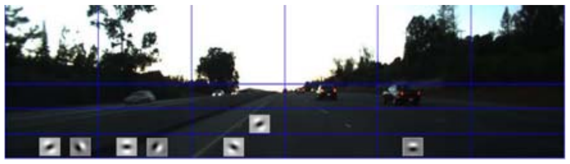
Moreover, top features selected by Adaboost with decision trees are shown in Figure 5. The oriented filters at different cells correspond to the selected features. The positions and orientations of these filters are more or less consistent with the positions and orientations with the lane markers or the vehicles like the vertical edge response at the 2nd row and 3rd column for classifier lane 3.

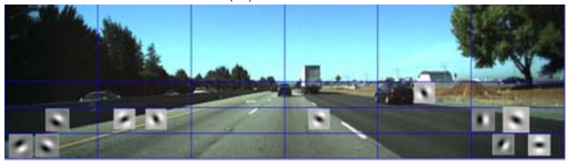
4. Conclusion and Discussion

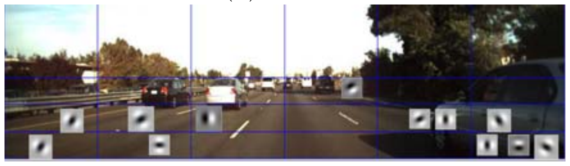
In this report, I have proposed a novel and effective algorithm to infer the lane number from a single image. The results show that bypassing explicit object recognition and achieving the inference goal directly from low-level representation with the spatial distribution works well in this scene classification problem requiring finer granularity. Some further improvements could include incorporating temporal dimension using HMM, making spatial partition more robust maybe by drawing rays from vanishing point.

Table 4. Confusion Table for SVM

	Lane 1	Lane 2	Lane 3	Lane 4	Recall
Lane 1	110	2			98.21%
Lane 2	2	453	37	8	90.60%
Lane 3		46	670	33	89.33%
Lane 4		6	44	700	93.33%
Precision	98.21%	89.35%	89.21%	94.47%	


(a) Lane 1


(b) Lane 2


(c) Lane 3

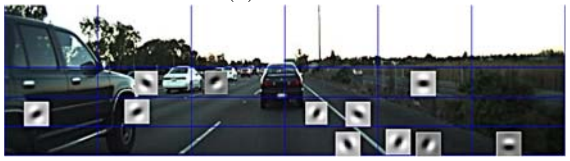

(d) Lane 4

Figure 5. Selected top features by Adaboost with decision trees for different classes.

Acknowledgments

I'd like to thank Honglak Lee and Zhi Li for some helpful discussions.

References

[1] D. Hoiem, A. Efros, and M. Herbert, "Geometric context from a single image," *International Conference on Computer Vision (ICCV)*, 2005.

[2] J. Kosecka and W. Zhang, "Video compass," *Proc. ECCV*, Springer-Verlag, 2002.

[3] M. Grant and S. Boyd. CVX: Matlab software for disciplined convex programming (web page and software). http://stanford.edu/ boyd/cvx, December 2008.

[4] M. Collins, R. Schapire, and Y. Singer, "Logistic regression, adaboost and bregman distances," *Machine Learning*, vol. 48, no. 1-3, 2002

[5] A. Genkin, D. D.Lewis and D. Madigan, "BBR: Bayesian Logistic Regression Software," *http://www.stat.rutgers.edu/ madigan/BBR/*

[6] C. Chang and C. Lin, "LIBSVM – A Library for Support Vector Machines,"

http://www.csie.ntu.edu.tw/ cjlin/libsvm