Ukrainian Catholic University

Faculty of Applied Sciences

Business Analytics & Computer Science Programmes

One camera driver assistant

Linear Algebra final project report

Authors:
Markian Matsyuk TECH GENIUS
Oleksandr Potapov RELIABLE TEAMMATE
Bryliak Pavlo ANCHOR

15 May 2019

Abstract

Fo the last 5 years, after the release of the 2014 Tesla autopilot in the model S, warning system and autopilots in cars are actively gaining popularity. In addition to the recognition of road signs, the detection of the lane and the distance to neighboring cars are perhaps the most important factors in such systems. Such systems are usually very complex and are being integrated at the manufacturer's plant, they are also definitely expensive as they use several types of cameras (for example, Tesla uses 8 [4]). Getting clear traffic labels for processing recognition is a daunting task. Therefore, we offer a method for recognizing the lane and the distance to surrounding cars using only one camera (for example, a video recorder that is barely in every car). Experimental test results on various videos from YouTube video hosting showed that our method works fairly accurately and computationally inexpensively. The code is available online at https://github.com/MarkiianAtUCU/LaneDetection

1. Introduction

Recognition of road lane and the nearest cars - an important task for cars using autopilot [5,6,7]. The latest algorithms for detecting the lane are based on a computer vision. Images taken from different types of cameras, such as visible light camera encoders, are processed to extract all significant data such as edges, stripe orientation and bandwidth, and they are combined with distance information measured by radar sensors. The computer vision system requires camera calibration before work, good environmental conditions and road conditions, as well as high speed processing to detect real-time lanes to match vehicle speeds. Therefore, most methods based on existing functions, offer three main stages of processing [5,8,9,10,11]: (1) preliminary processing: increasing the illumination of the original image taken from the camera; (2) main processing: extracting features of the road marking, such as edges, texture and color; and (3) post-processing: removing emissions or clustering the detected lines of the lines. To recognize the closest cars, we used Haar cascades [12] and wrote our own way of determining the distance to cars, which greatly facilitates this task unlike using AI [13].
We propose three stages of processing using only one camera to recognize road markings and distance to surrounding cars, from these two companies a report will be compiled.

2. Problem setting

For solving such complex task, we should split the project into the three partsthe three parts for each task:
Lane detection, Car Detection, Distance to car estimation. Also we decided not to use any kind of neural networks due to critical latency requirements (feedback of our system must be as quick as it possible)
First part is Lane detection – a lot of problems are connected with this part, so we must deal with not ideal real life conditions, bad visibility or absence of road markup. We developed our own way of lane detection, based on best approaches. First of all we warp perspective of our region of interest (zone at front of car), to get “bird eye view”
We use manually selected values of points that describes our region of interest (ROI)

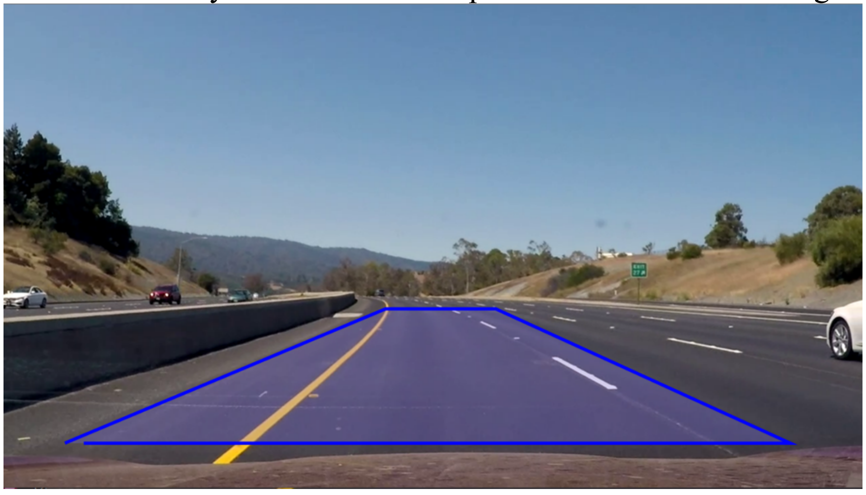


Figure 1. Region of interest

We map this values to a square image 500 pixels by 500pixels to obtain the 3×3 transformation matrix of the projective transformation.
To do so we solve system of linear equations, where $(x_1, y_1), \dots, (x_4, y_4)$ are coordinates of our ROI

$$\begin{bmatrix} x_1 & x_2 & x_3 \\ y_1 & y_2 & y_3 \\ 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} \lambda \\ \mu \\ \tau \end{bmatrix} = \begin{bmatrix} x_4 \\ y_4 \\ 1 \end{bmatrix}$$

The column space form homogenious coordinates, by adding 1 as third coordinate, so multiplies of that vectors will give us the same points.
After that we scale our columns by the computed in previous step coeficients

$$A = \begin{bmatrix} \lambda x_1 & \mu x_2 & \tau x_3 \\ \lambda y_1 & \mu y_2 & \tau y_3 \\ \lambda & \mu & \tau \end{bmatrix}$$

Matrix A will map$(1,0,0), \dots, (1,1,1)$ to multiples of $(x_1, y_1, 1), \dots, (x_4, y_4, 1)$ So it maps basis vectors to specific position in the image. We are performing same steps for the target 4 points (500x500 square)
After that we obtain our transformation matrix $M = BA^{-1}$
A^{-1} maps source position into basis vectors, when B maps from basis vectors to perspective warp we are looking for. So M maps source points to transformed
To map (x, y) pixel to its corresponding location (x'', y'') we multiply it and dehomogenize it:

$$\begin{bmatrix} x' \\ y' \\ z \end{bmatrix} = M \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

$$x'' = \frac{x'}{z'}$$

$$y'' = \frac{y'}{z'}$$

To transform perspective back, we should multiple inverse of M on original points and perform dehomogenization.
The rest of pipeline is pretty straightforward:
Threshold images, match colors, find the most bright points on thresholded images (one set for left part of image and one for right), fit 2nd power polynom to this points, fit circle to this points, compare squared error, use more precise approach, after detection complited – warp perspective back, mark lanes on original image.
Also we use one trick in lane detection: if parameters of our circles/polynoms are changing very fast we try to extrapolate them and use extrapolated values instead of real, it helps us not to lose tracking, when road markup is berely visible.
Second part – car detection. Due to limitation we decided to use tricks to detect car on road. We know, that that usually other cars on the road didnt moves with precisely same speed, so to make life easier to our CPU lets pay attention to objects that moves with different speed than road. Every such object we bound with rectangle, If its area is big enough to be a car, we add it to list. But we check if it is not already present in list of trackable cars, we do so by comparing features of images present in list and candidate or adding in list. Features are detected with SIFT (scale-invariant feature transform) algorithm[1].
Also every entry of our tracking list have its own lifetime, decreasing every frame, if it wasn't detcted on image for a long time – it will be burnout from list. Also we use HAAR cascades for acquiring objects in list. Haar cascades are not very precise in detecting, so we asks classifier if our entries of tracking list contains at least one car, if so – this entry will get infinite lifetime and will be tracked as long as it possible.
Tracking is performed by MOSSE (Minimum Output Sum of Squared Error) filter [2].
Combinations of these tricks can give us more or less accurate, smooth tracking, with no use of heavy neural networks

Finally – estimating distance to car. It's the easiest part of our task. In every point of time, we draw line, from bottom center of our frame, to center of every car on image

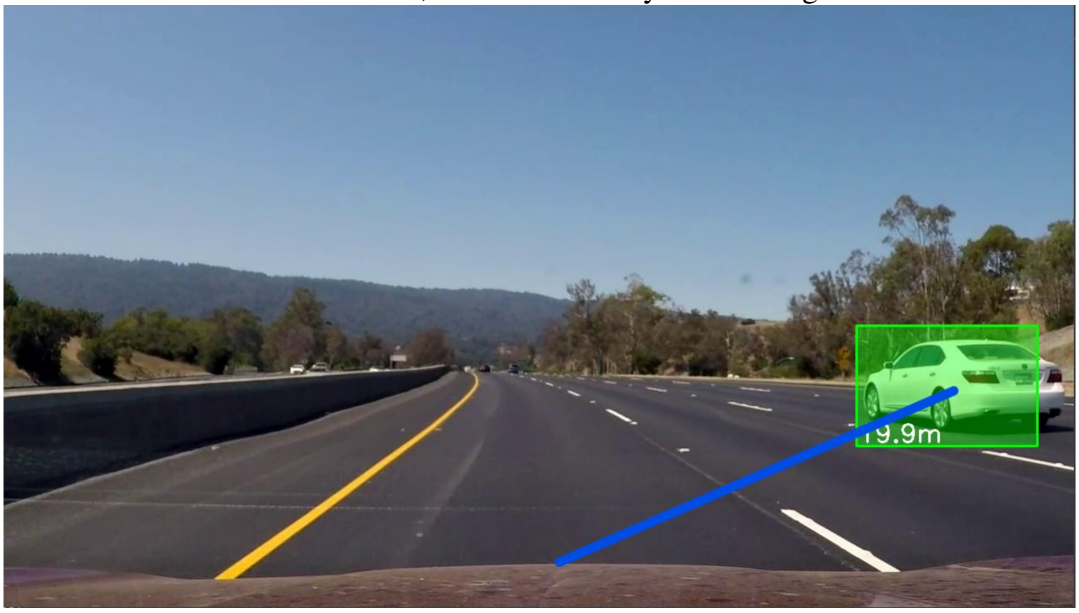


Figure 2. Estimated distance to the nearest car

After that with our transformation matrix from first task, we convert it into "bird-eye space", where we calculate its length. This length will be scaled distance to car. Knowing real distance to car we can calibrate our estimation.

3. Overview of existing approaches

3.1 Hough Transform

This method takes as a base an assumption that most lanes are designed to be relatively straightforward not only as to encourage orderliness but also to make it easier for human drivers to steer vehicles with consistent speed. Therefore, its approach may be to first detect prominent straight lines in the camera feed through edge detection and feature extraction techniques.

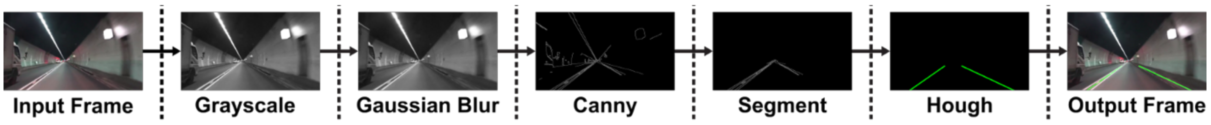


Figure 3. Example of Video registrator capture procced with Hough Transform

Its working principle is pretty simple. As you can see on figure 3, we tranform our input frame, on step 3 we apply Canny Detector, which is a multi-stage algorithm optimized for fast real-time edge detection. The fundamental goal of the algorithm is to detect sharp changes in luminosity (large gradients), such as a shift from white to black, and defines them as edges, given a set of thresholds. The Canny algorithm has four main stages: Noise reduction, Intensity gradient, Non-maximum suppression, Segmenting lane area. We will not cover in the details all of them. After applying Canny Detector we perform lane area segmentation and then Hough tranformation to get starting imageframe with fitted lines[3].

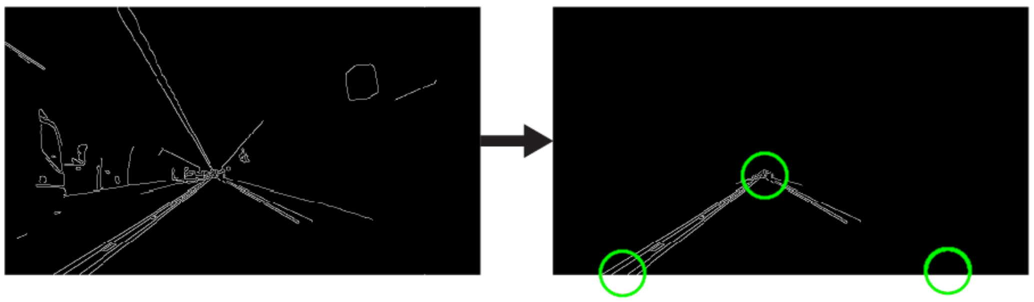


Figure 4. Example of Video registrator capture procced with Hough Transform

This algorythm`s main advantage is speed, it work good with real-video proccessing, but road are not straight all the time, so it behavior on corners and crossroads is difficult to predict.

3.2 Spatial CNN(Convolutional Neural Network)

While CNNs have proven to be efective architecture for both recognizing simple features at lower layers of images (e.g. edges, color gradients) as well as complex features and entities in deeper levels (e.g. object recognition), they often struggle to represent the "pose" of these features and entities — that is, CNNs are great for extracting semantics from raw pixels but perform poorly on capturing the spatial relationships (e.g. rotational and translational relationships) of pixels in a frame. These spatial relationships, however, are important for the task of lane detection, where there are strong shape priors but weak appearance coherences. For example, it is hard to determine traffic poles solely by extracting semantic features as they lack distinct and coherent appearance cues and are often occluded.

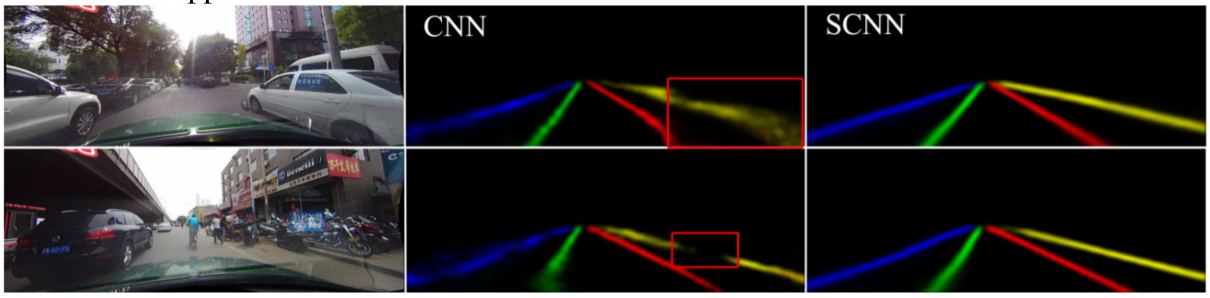


Figure 5. The car to the right of the top left image and the motorbike to the right of the bottom left image occlude the right lane markings and negatively affect CNN results

For sure CNN is a great approch, you can train the neural network to perform really good, but it is obvious that any NN needs certain amount of computational power to perform in real time.

4. Implementaion

In this part we are going to go step by step the proccesses we described in Problem setting.

4. 1 Lane recognition

As a starting point we have a video recorded on video-registrator camera.

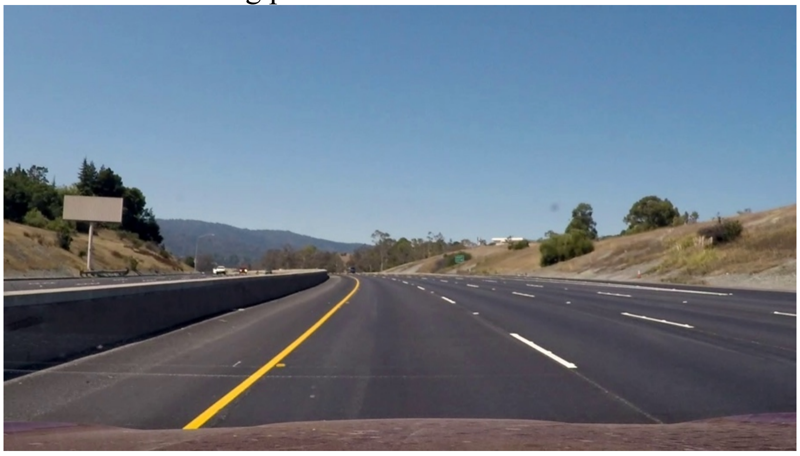


Figure 6. Example of Video registrator capture

Next step is to perform the image transformation to get a “bird-view” perspective.

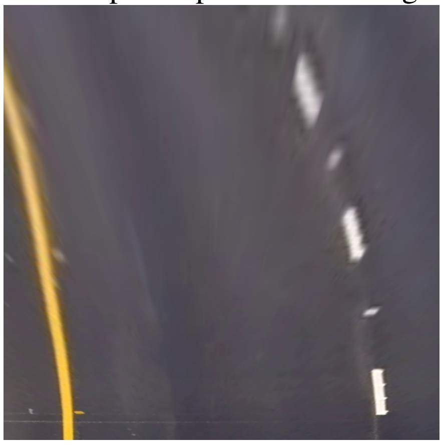


Figure 7. Image tranformed to “bird-eye” perspective

Then follows futher image tranformations of our image to:
-Get saturation chanel
-Get yellow and white pixels of an image
-Combine two previous images
-Threshold image, to get rid of unnecessary details

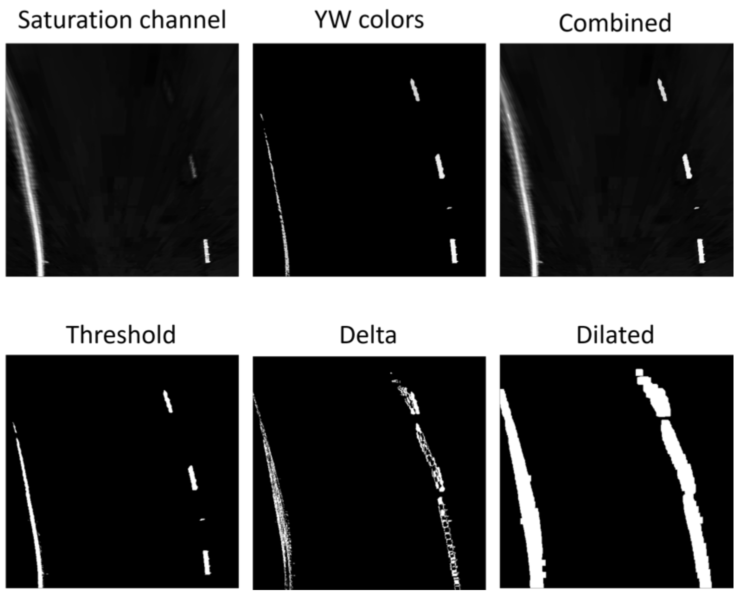


Figure 8. Image tranformation for detecting lanes

As we have specified lanes and cleared all the noise, we are looking for delta of 10 frames to get an information about movement. The next step is to detect points on that lane and fit curves to them. To do so, we should threshhold an image and perform histograms(see figure 8). If there is missing point somewhere because of the shadow or other noise, we estimate it(as you can see a yellow dot on figure 8).

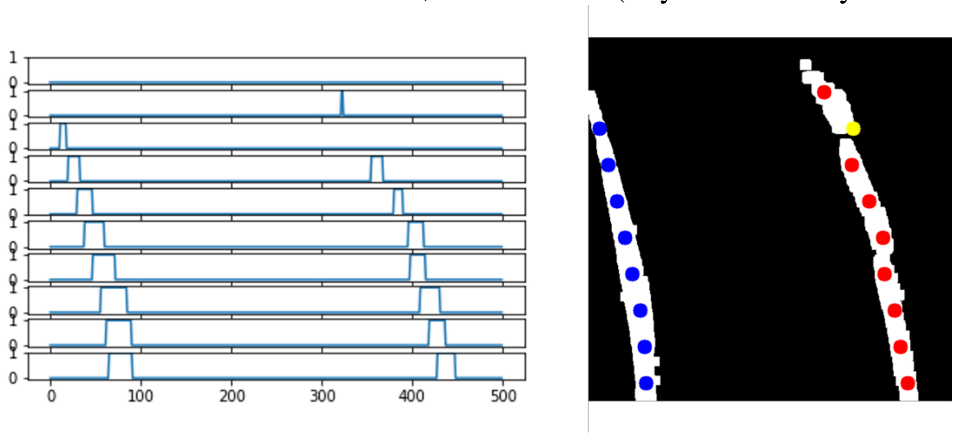


Figure 9. Marking lines with dots

To fit a curve, we use the second power polynome.

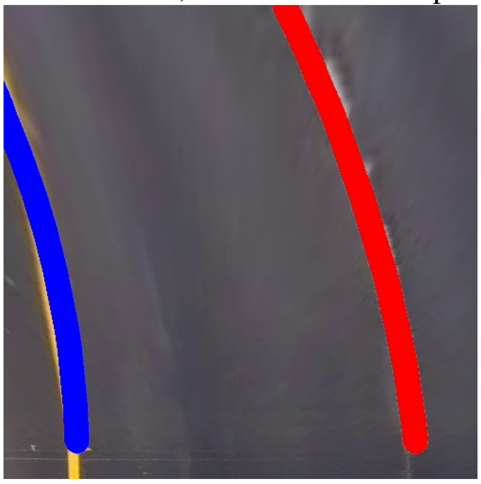


Figure 10. Fitting curves

As the last step, we warp(tranform) image back from a “bird-eye” perspective to a normal one.

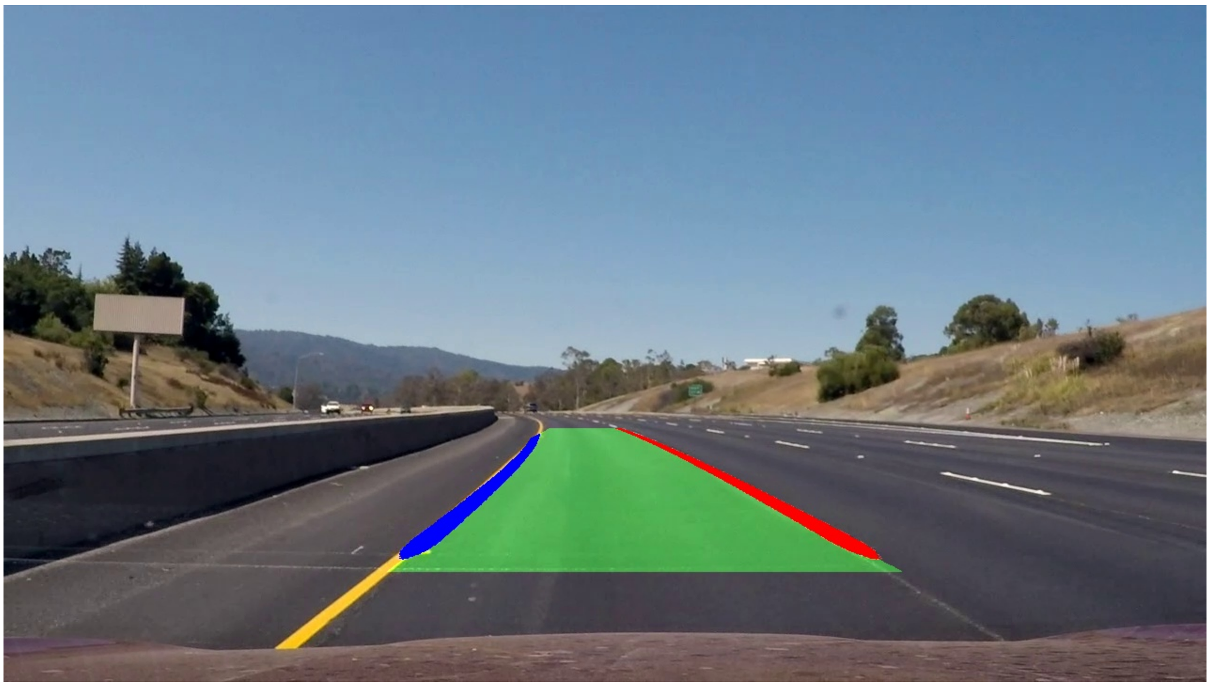


Figure 11. Photo from Demo video of working algorythm

4. 2 Cars recognition

The next step is the nearest cars recognition, to do so we use Haar Cascade Classifier, it is a method utilized for detecting object, also called as Viola Jones method due to its introduction by Paul Viola and Michael Jones for face detection. This method has 4 points for detecting an object, such as Haar-like feature, integral image, AdaBoost learning and Cascade Classifier[12].

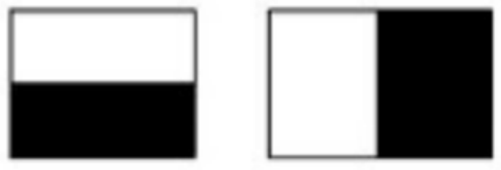


Figure 12. Edge Feature [14]

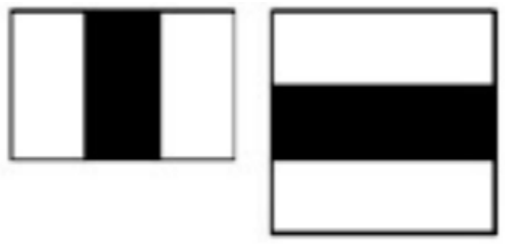


Figure 13. Line Feature [14]

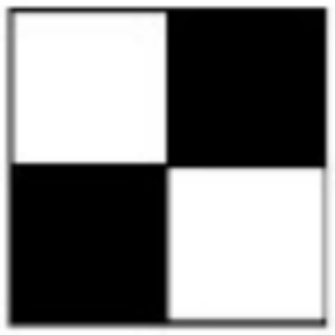


Figure 14. Four-rectangle Feature [14]

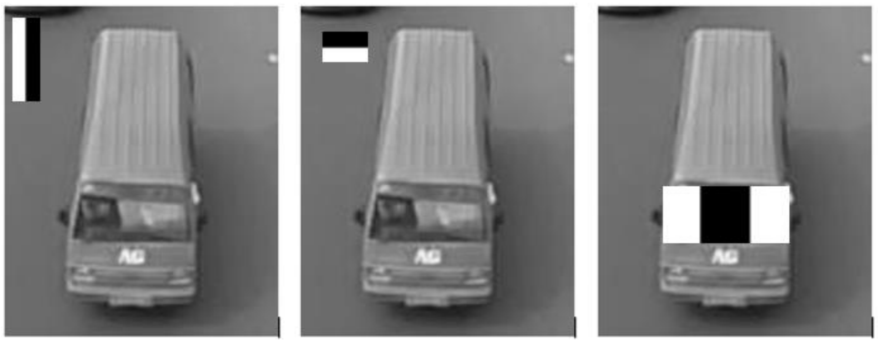
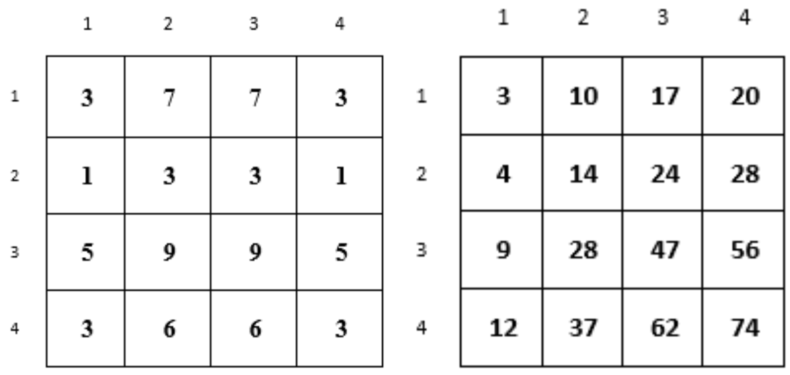


Figure 15. Feature Detection Scheme

Haar-like feature is a rectangular feature providing specific indication to an image**.** Figure 12, Figure 13 and Figure 14 are the examples of common variety of Haar-like feature. Haar-like feature offers high speed computation depending the number of pixels inside the rectangle feature and not depending on each pixel value of the image [13]. In obtaining object detection value, Haar- like feature value was calculated using integral image. Integral image could calculate values accurately and relatively quick by creating new presentation of image by using value of region previously scanned by specific Haar-like feature as shown in Figure 15. The value of integral image was obtained by sum value of previous index, started by left top until right bottom; moreover, Integral image could be calculated using Equation 1, for example the input and new presentation depicted in Figure 16(a) and 16(b).

$$s(x, y) = i(x, y) + s(x, y - 1) + s(x - 1, y1) + s(x - 1, y - 1) \quad (1)$$



	1	2	3	4
1	3	7	7	3
2	1	3	3	1
3	5	9	9	5
4	3	6	6	3

	1	2	3	4
1	3	10	17	20
2	4	14	24	28
3	9	28	47	56
4	12	37	62	74

Figure 16. (a) Input Image and (b) Integral Image of Input Image

Value which had been calculated by using integral image would then be compared with the threshold value of specific features provided by AdaBoost. This should be completed to find potential features because not all features were relevant to use for specific object detection. AdaBoost combines potential features called weak classifier to become strong classifier. Weak classifier means less accurate or also irrelevant prediction [14]. Relevan and irrelevan features shown by Figures 17. [14]

1) All features example 2) Relevant feature 3)Irrelevvant feature

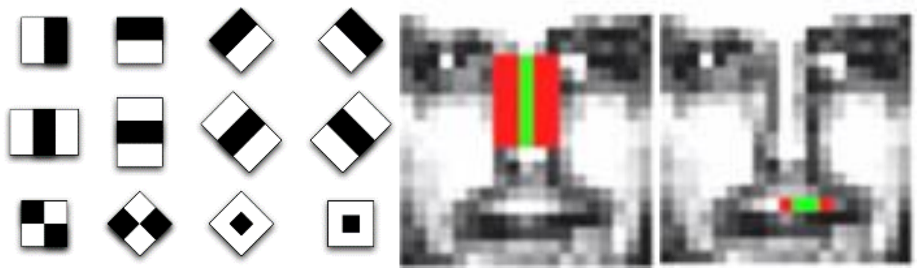
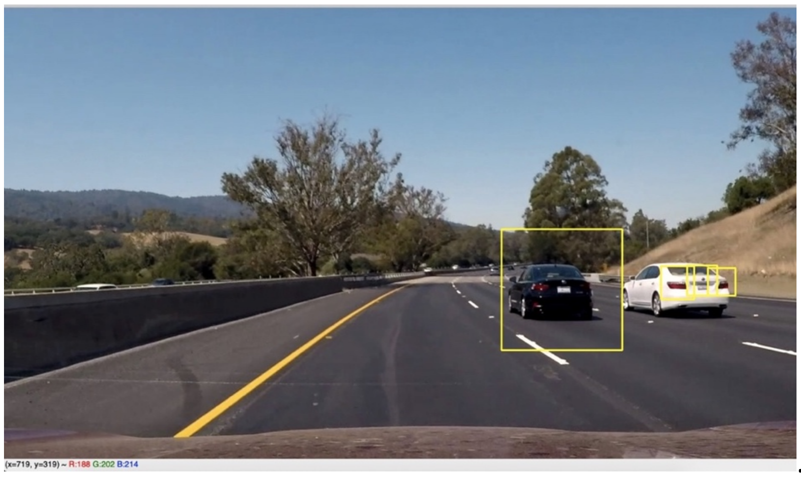


Figure 17. Examples of Relevant and Irrelevant Features

We use same approach for detecting cars, as we metioned before, we use existing Haar Cascade Classifier[12], it used back window of a car as a relevant feature, images were taken from the camera on the bridge. It is a good cascade classifier to use, because we mostly see backwards of the cars during the movement of our car



.

Figure 18. Cars recognition using Haar Cascades Classifier

4. 3 Distance to the cars estimation

After we recognized a car, we then convert an image to a “bird-eye view” using matrix transformation, to estimate a distance to a car(we already did it with lane detection). Afterwards we should combine those to methods to get a final result – lane detection and distance to the nearest cars in one image.

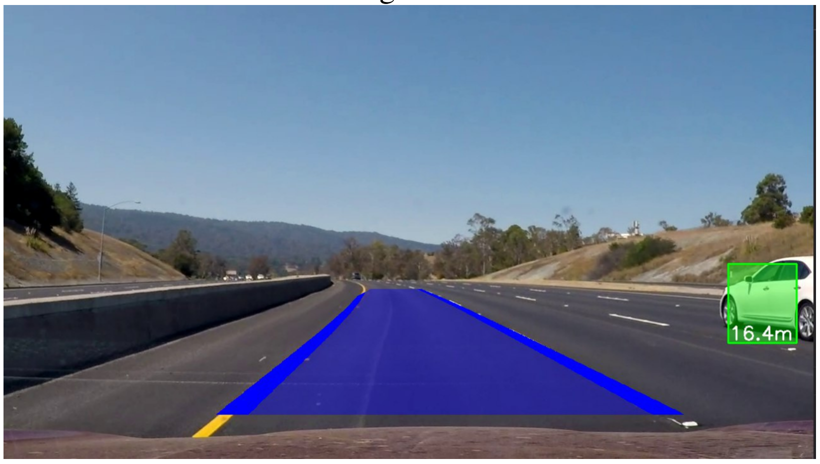


Figure 19. Combined distance estimation and lane detection

6. Complexity, parameters, limitations

To analyze video and make this task computationally not heavy, we proceed images every 10ms, anyway it takes some time to proceed computations, we could say that this system could be used in real-time proceeding but it should be tested in real life. Also we did not tested how algorythm performs with shaded roads(we do estimations of missing points see figure 4) but we should definitely pay more attention to it. One more question is predicting the road line, it is especially actual question while car is riding a road that turns to the side having a small radius, so our camera captures only 15-20 meters of the road and it could cause a really dangerous situation. Safety is also an important issue here, the proposed system limits our view to a certain angle in fron of the car, while we do not know what is going on the sides and backwards, so we are not able to react(Tesla`s autopilot sensors captures 360 degrees).

7. Conclusion

We proved that it is possible to perform lane detection and distance to the nearest cars estimation with just one camera, but using it for real life self-driving systems leads to many obvious risks – not having a 360 degree angle of vision takes an opportunity from the car to react on inapropriate behavior of ther cars near you. Using only one camera increases risks of ‘losing’ the lane owing to shadows, its damages and during a nighttime. This system should definitely be tested in real time proccessing, but works like this one helps to find better algebraic ways of solving the same task, which will hopefully lead to an updates in existing systems to make them more eficient and accurate. This project is now ready to be tested as a driver assistant in a car.

References

1. https://www.cs.ubc.ca/~lowe/papers/ijcv04.pdf
2. http://www.cs.colostate.edu/~vision/publications/bolme_cvpr10.pdf
3. Detailed discription of Hough Transform and Spatial CNN https://towardsdatascience.com/tutorial-build-a-lane-detector-679fd8953132
4. https://insideevs.com/news/337337/how-many-cameras-does-tesla-autopilot-20-currently-use/
5. Nan Z., Wei P., Xu L., Zheng N. Efficient Lane Boundary Detection with Spatial-Temporal Knowledge Filtering. Sensors. 2016;16:1276. doi: 10.3390/s16081276.
6. Lee B.-Y., Song J.-H., Im J.-H., Im S.-H., Heo M.-B., Jee G.-I. GPS/DR Error Estimation for Autonomous Vehicle Localization. Sensors. 2015;15:20779–20798. doi: 10.3390/s150820779.
7. Hernández D.C., Kurnianggoro L., Filonenko A., Jo K.H. Real-Time Lane Region Detection Using a Combination of Geometrical and Image Features. Sensors. 2016;16:1935. doi: 10.3390/s16111935.
8. Truong Q.-B., Lee B.-R. New Lane Detection Algorithm for Autonomous Vehicles Using Computer Vision; Proceedings of the International Conference on Control, Automation and Systems; Seoul, Korea. 14–17 October 2008; pp. 1208–1213.
9. Aly M. Real Time Detection of Lane Markers in Urban Streets; Proceedings of the IEEE Intelligent Vehicles Symposium; Eindhoven, Netherlands. 4–6 June 2008; pp. 7–12.
10. Hoang T.M., Hong H.G., Vokhidov H., Park K.R. Road Lane Detection by Discriminating Dashed and Solid Road Lanes Using a Visible Light Camera Sensor. Sensors. 2016;16:1313. doi: 10.3390/s16081313.
11. Advanced-Lane-Detection. [(accessed on 26 October 2017)]; Available online: https://github.com/kylesf/Advanced-Lane-Detection.
12. Haar Cascades for cars detection https://github.com/andrewssobral/vehicle_detection_haarcascades
13. Nvidia`s AI for cars detection in autopilot systems https://blogs.nvidia.com/blog/2019/01/07/nvidia-drive-autopilot/
14. P. Viola and M. M. J. Jones, "*Robust Real-time Face Detection*," Int. J. Comput. Vis., Vol. 55, No. 2, Pp. 137–154, 2004.
15. A. Mordvintsev, "*OpenCV-Python Tutorials Documentation Release Beta*," 2017.
16. Moch Ilham Ramadhani*[1], Agus Eko Minarno[2], Eko Budi Cahyono*[3]
"Vehicle Classification using Haar Cascade Classifier Method in Traffic Surveillance System", **KINETIK**, Vol. 3, No. 1, February 2018, Pp. 58-59