Lane Detection using Particle Filters

Rafael de Carvalho, Francisco Salgado, Mehmet Ozgur Turkoglu

***Abstract*—Accurate and robust lane detection is a difficult task in some cases such as when there are dashed lane markings and curved lanes. In this work, we propose a method to detect the road lane in those challenging sceneries. For the near vision, our method is based on color and structure information and particle filtering. For the distant vision, our method combines RANSAC and particle filtering. The effectiveness of our approach has been shown on the data-set which consists lanes in different conditions.**

I. Introduction

Lane detection is the first step in driver assistance. In its most simple way, it consists of identifying the lane boundaries, which are usually lines marked on the ground. Identifying this marks, will allow to find out the position of the vehicle and proceed to planning out its movement. This step however, as described, is more complex than this and the need to plan out movements such as changing lanes, following traffic rules and simply reaching the destination.

Currently, with lane detection, several applications can be applied. This include three categories of systems: Informative systems, Assisted driving and autonomous driving [1]. For Informative systems, we have applications such as Lane Departure Warning [2]. For assisted driving, Adaptive Cruise Control, Lane Change Assist and Turn assist can already be found [3], [4].

This area has been subjected to great advances, as in past research [2], [5]–[9]. However, providing a general algorithm is difficult. Road topology significantly varies in the following aspects: The lane markings can vary in shape, color and position (such as double lines, yellow marks, inconsistencies); The road and the boundaries can take many forms; The systems are gravelly affected by noise such as puddles, other markings on the ground and shadows.

The presented algorithm will focus on tackling the scenario of daylight driving with consistent marks on both sides of the lane. It will consist of a pre-processing where the captured frames will be transformed into birds' eye view and the lane marks will be extracted, a lane detecting phase where, the lane which was divided into a near and distant part, detecting the position of the lane boundaries on the near part, and the curvature of the lane on the distant part. A tracking phase where previous frames help determine the results of future frames, providing a more robust, accurate system.

With this algorithm, it is aimed to explore the problem of lane detection, in order to make several applications possible.For example, the position of the lane can be used for lane departure warning and the lane curvature can be useful in turn assist.

This report is structured as follows. In Section 2, we describe the proposed method in detail. Then, we mention dataset and implementation details and provide some results from the experiments in Section 3. Finally we discuss the results and draw a conclusion in Section 4.

II. Method

The architecture of our system starts with a small preprocessing step, following up by splitting the lane in two parts before the actual detection: the near and the distant part.

A. Camera calibration

In order for our algorithm to be applied, some parameters have to be manually chosen. This choice is made using only one frame per video, since the only it is to determine the position of the camera relatively to the lane and the center of the car.

B. Bird's eye view

The first step is to virtually rotating the camera, changing the perspective from the mentioned frontal view, to a vertical top view, designated *Bird's eye view*. This is accomplished in the same way as [7], [10] where 4 points are manually chosen during the camera calibration, where this points are assumed to define an horizontal rectangular plane on the real world. The chosen points were always placed in the lane markings. This method is proven to work very well, as it can be seen in figures 1a and 1b.
An additional advantage, is that one can specify at which distance there will be a better resolution, by adjusting the "length" in depth of the rectangle. By adjusting, some areas are more or less distorted, due to the perspective homography operation. This "length" is dependent on the resolution of the image as well as the region of interest (ROI) for only the ROI is used in the rest of the algorithm. The cropped and transformed image is then divided into the near part (30% of the image,starting from the bottom) and the distant part (the 70% of the image).

C. Near part

We assume that near part of the lane is straight but distant part can be curved so we have different approaches to detect near and distant parts of the lane. We roughly determine 30% of the transformed image is straight and the rest can be curved. In this section, we introduce our method for near part.

(a) Sample image from from the dataset (b) Respective cropped transformed image

Fig. 1: Demonstration of the transformation step

Lane detection

In order to extract the lane features, an edge map is made looking to identify the exact position of the lane. Introduced in [10], [11], an edge detection method that produces bar-like structures is used.
This is specially designed to find vertical thick lines and works as follows: Every pixel is of the *Bird's eye view* *b*(x,y) compares it's value with it's horizontal left *b*(x-m,y) and right *b*(x+m,y) neighbors at a distance *m*. We define the distances, as follows:

$$\left.\begin{matrix} R(x,y) = b(x,y) - b(x+m,y) \\ L(x,y) = b(x,y) - b(x+m,y) \end{matrix}\right\} \quad (1)$$

And, using a Threshold *T*, the edge map is defined:

$$map = \left.\begin{matrix} 1, & if\ \ R, L > 0, R + L \geq T \\ 0, & Otherwise \end{matrix}\right\} \quad (2)$$

This algorithm has some advantages over the traditional edge map. First, the *m* parameter can be adjusted to fit different lane widths; Second, instead of providing the limits of the lanes, each value depicts whether a pixel is/is not part of the lane, which is extremely useful for the next steps; Third, this method adds a layer of resistance to brightness changes, as shadowed areas will generally maintain the relationship between nearby pixels. The parameters that were found to be more effective were 20 for both the *m* and the *T*
This method generates some random blobs and noise, which are filtered out by the rest of the pre-processing. The first step is to remove the small groups of blobs, this is done by neighborhood deletion (250 pixels) using the function *bwareaopen* from MATLAB. The second step, given that the first one does not take care of the long yet thing noise that was found to be also seen to be present, as a last step Hysteresis reconstruction was applied after eroding the image, through MATLAB's *Imreconstruct*.

Color-selection

For better obtaining an edge map describing the road lanes, it is useful to remove all noise coming from the edge detector. For this, the areas of the image containing lanes could be filtered based on colour (in the dataset used, lanes are either white or yellow). However, due to different lighting and road conditions, selecting colours using the RGB color-space might be tricky, as the relationship between these 3 channels (R, G and B) is unintuitive. For instance, a it is difficult to understand how lighting might affect each channel in RGB space and set boundaries on the channels to select a specific colour independently of the environment conditions. As an alternative, the original color image is converted to the HSI color-space (Hue, Saturation, Intensity). This proves to be easier for selecting areas of a frame of the video containing a single colour by setting boundaries on the channels, because it represents colours similarly to the human eye. The hue channel represents the color itself, while the saturation channel tells how much the colour is affected by white. The intensity channel also limits the saturation values. White and yellow colours are selected if the value of their channel belongs to the following ranges.
White color:

$$S < 0.03, 0.65 < I$$

Yellow color:

$$0.05 < H < 0.17, 0.2 < S < 0.5, 0.5 < I$$

Image reconstruction for final edge map

"Imreconstruct" (MATLAB built-in function) is then used to obtain the final edge map. The colour mask is used as the seed that is applied to the output of the edge detection algorithm, resulting in a much less noisy edge map. This process can be seen in Figure 3.

Oriented Histogram

From the edge map, the goal is to obtain the location of the lane markings on the left and right of the car. An histogram is produced, where for each column of the edge map, the value on the histogram corresponds to the sum of the intensity of the pixels in that column. It can easily be seen that this accumulator will have two peaks on the columns where the lane markings are located. This method does not prove as effective for the top part of the edge map, due to the curvature of the lane resulting in the two peaks being more spread.

(a) Result from the color selection

(b) Result from the initial edge map

(c) Final edge map obtained for the frame

Fig. 2: Example process of obtaining the final map

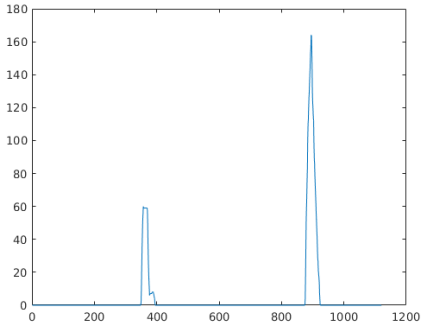


Fig. 3: Oriented histogram obtained from final edge map.

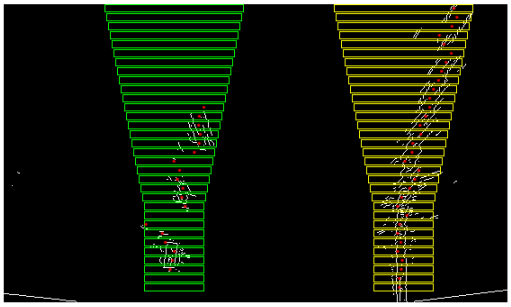


Fig. 4: Diagram of the small areas used for extracting line points

Tracking near part

We track the horizontal position of the lane edges in the image by using particle filter. We use 1000 particles and our state vector includes 2 variables correspond to these vertical positions. We use oriented histogram introduced in previous section as a likelihood function. In order not to confine all the particles in a very small region, we add some little constant to oriented histogram before using it as a likelihood function. In the update state of the particle filter, the weight of each particle is updated as following.

$$w^{(k)} = h_{left}(x_1^{(k)}) + h_{right}(x_2^{(k)})$$

$$w^{(k)} = \frac{w^{(k)}}{\sum_{k=1}^{1000} w^{(k)}}$$

where h_{left} and h_{right} are the oriented histogram functions of the left and right parts of the image respectively and $\boldsymbol{x}^{(k)}$ is predicted state of particle k. Then, we calculate MMSE estimation.

$$\tilde{\boldsymbol{x}} = \sum_{k=1}^{1000} w^{(k)} \boldsymbol{x}^{(k)}$$

One-ahead prediction for particles are done as following.

$$\boldsymbol{x}^{(k)}(i+1) = \boldsymbol{x}^{(k)}(i) + f_{switch}(\tilde{\boldsymbol{x}})\tilde{w} + \boldsymbol{n}$$

where $\boldsymbol{w}$ is Gaussian noise, $\tilde{w}$ is estimated lane width ($\tilde{w} = \tilde{x}_2 - \tilde{x}_1$) and f_{switch} is a function which handles lane switching as defined following.

$$f_{switch} = \begin{cases} 1 & \tilde{x}_1 > T_1 \\ -1 & \tilde{x}_2 < T_2 \\ 0 & otherwise \end{cases}$$

where T_1 and T_2 are defined thresholds. The function provides particles move left or right as much as estimated lane width if the the vehicle is about to switch the lane.

D. Distant part

For the distant part, the applied algorithm is able to extract the change in curvature of the road.

From the 70% distant part of the transformed image, an edge map is created using *Sobel's* Isotropic 3x3 Gradient Operator. Then, knowledge from the near part is applied. A small rectangular area is chosen at the lower end of the image, centered on the estimated left and right lines. In this small patch, an oriented histogram is computed in the same way as in the near part and more rectangles are sequentially placed after the previous one. Additionally, in order to capture large curvatures, the rectangles width is increased on every placement. With this algorithm, the result is a set of points that are situated on top of the lines (See Figure 4.).

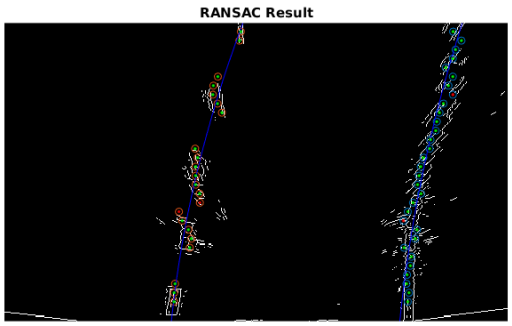


Fig. 5: Result of applying RANSAC to a set of lane points, where only the inliers count for the polynomial fit

RANSAC

The goal is then to find where the lane edges (lane markings) are located based on a set of points representing the edges of the lane markings. To find the second-order polynomial that fits over the input set of points, some filtering needs to be done in order to eliminate the noise. For this, RANSAC (Random Sample Consensus) proved to be an effective method. RANSAC is an iterative algorithm, that aims to fit a model to the given data, by separating the inliers from the outliers. Each iteration consists of two parts. Firstly, a model is fit to a random selection of data points. Then, all points of the input set are evaluated using a given distance function against this model to decide if they are an inlier or outlier. The model with the most number of inliers is picked as the model that best fits the input data after a given number of iterations. The fitting function used for this algorithm was a simple second-order polynomial fit and the distance function evaluates the distance from each point of the input data to the modeled polynomial curve and compares it to a threshold. (See Figure 5.)

Tracking distant part

In order to determine the slope and the curvature of the road lane (distant part of the road) more accurately, we track these 2 parameters by using particle filters (State vector consists 2 values which correspond slope and curvature.). We use RANSAC result as a measurement in th particle filter. We take either use left or right line parameters depends on the number of inliers they have. The weight of each particle is calculated as follow.

$$w^{(k)} = e^{-0.5(\boldsymbol{r}^{(k)})^T \boldsymbol{C_v}^{-1} \boldsymbol{r}^{(k)}}$$

$$w^{(k)} = \frac{w^{(k)}}{\sum_{k=1}^{1000} w^{(k)}}$$

where $\boldsymbol{r}^{(k)} = \boldsymbol{x}^{(k)} - \boldsymbol{z}$, $\boldsymbol{z}$ is the measurement which correspond to RANSAC output (slope and curvature). $\boldsymbol{C_v}$ is covariance matrix of the measurement noise which is empirically roughly determined. MMSE estimation is used for this particle filter as well and prediction step only consist of process noise.

$$\boldsymbol{x}^{(k)}(i+1) = \boldsymbol{x}^{(k)}(i) + \boldsymbol{n}$$

Inverse Transformation

Finally, we generate bunch of points on the lines and by using inverse of transformation matrix, we transform these points which correspond to lane edges in original input image.

III. Experiments & Results

A. Data-set

For developing this algorithm, a set of video recordings of driving on different roads in daylight was used. These videos were recorded by us with cameras placed in front of the vehicle, facing forward where the road right in front of the vehicle is shown. The lane markings should be clearly visible on both sides of the road.

B. Results

The proposed method is tested on different conditions, including solid and dashed lane markings, straight and curved lanes, vehicle occlusion (not in traffic jam) and road marks. The results of our experiment indicate that our method is robust to track the lanes especially near part (linear part of the lane). But also, most of time it successfully tracks the distant part.

Some experimental results are given in Figure 6. Also the real-time video-clip is created for around 900 frames which is available via the link: https://www.youtube.com/watch?v=arXEdqjhSaI&feature=youtu.be

C. Implementation Details

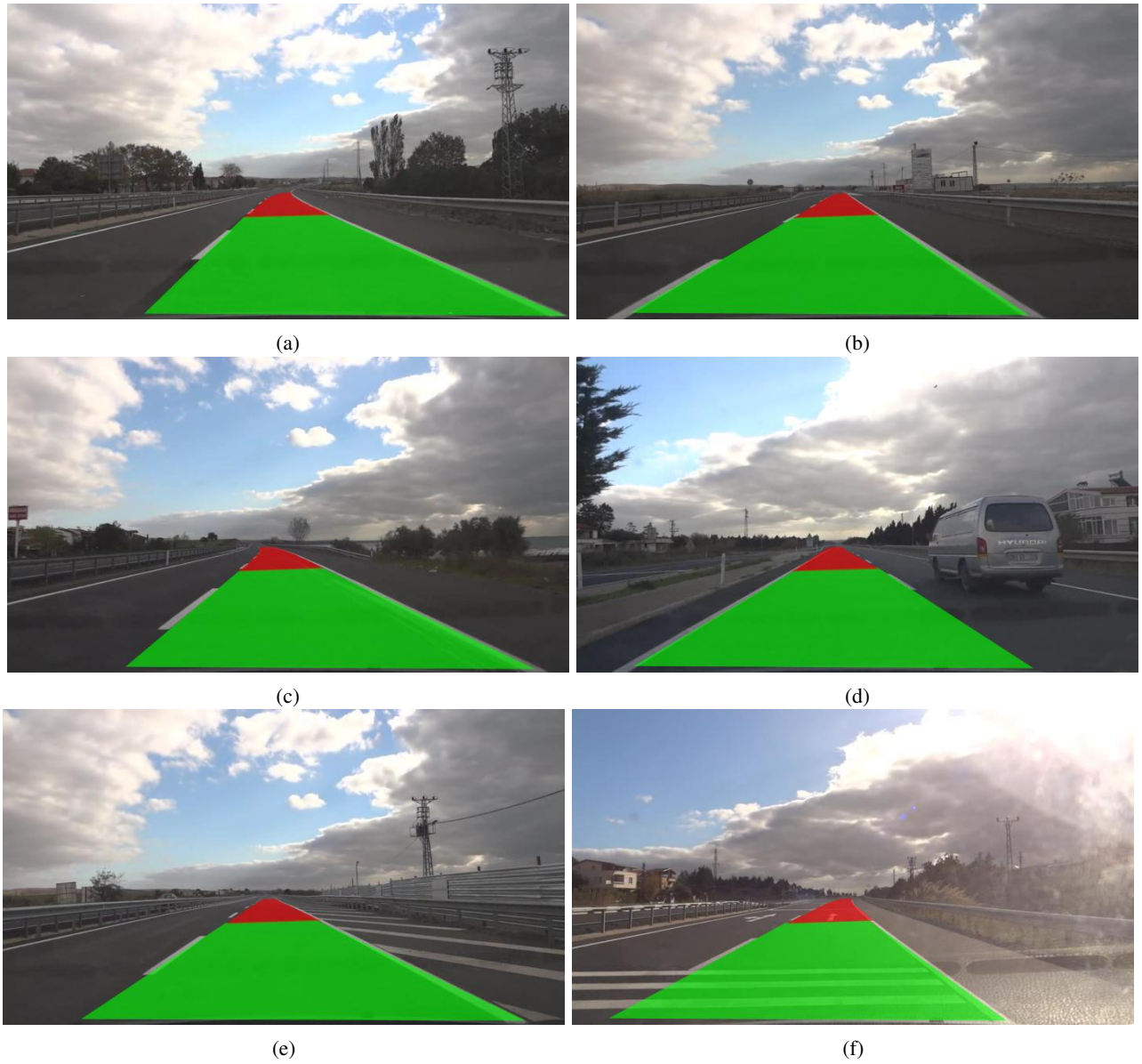
The code is implemented in MATLAB. We used "ParticleFilter" class from MATLAB "Robotics" package. For RANSAC algorithm, we used Peter Kovesi's implementation which is publicly available. The CPU time for each frame is approximately 0.6 second. The code is available via the following link. https://github.com/0zgur0/Lane-Detection-using-particle-filters

IV. Discussion & Conclusion

Lane recognition can be tricky to execute as different lighting conditions affect how the lanes are captured. Lanes can be continuous, dashed on non existing at all which also difficult recognizing where the road ends. The method proposed here works fine for footage where the lanes are visible and the scene is homogeneously lit.

By visual inspection, it can be seen that the presented algorithm manages to achieve a good result. The near part fits the lane perfectly at almost all times, only requiring some time to stabilize during a few frames in the beginning and after lane switching. The distant part, being largely dependent on the near part, exhibits the same characteristics plus some additional occasional flickering on more noisy areas.

Many improvements can be made in order to make the algorithm more reliable and faster. Firstly calibrating the camera used for acquiring the footage, would allow eliminating the camera distortion when obtaining the bird's eye view of the road and no manual input would be required when running the algorithm. The fitting function for fitting the second order



(a) (b) (c) (d) (e) (f)

Fig. 6: Experimental results. Some successful frames generated by our algorithm.

polynomial to the lanes could be improved by restricting the values of each polynomial, eliminating models corresponding to impossible curvatures of the road. For using this algorithm in a real time application, using compiled code instead of interpreted as is the case of MATLAB, would drastically improve the execution speed. For instance, compiled C++ could be used, recurring too OpenCV for the image processing tasks.

More functionalities could be paired with this lane detection algorithm to make it suitable for self-driving vehicles. Determining the lane curvature can be done from the fitting polynomial. Furthermore, other cars detection should be implemented too (useful for avoiding collisions) together with traffic signs recognition.

REFERENCES

[1] Emilio Frazzoli, Munther A Dahleh, and Eric Feron. Real-time motion planning for agile autonomous vehicles. In *American Control Conference, 2001. Proceedings of the 2001*, volume 1, pages 43–49. IEEE, 2001.

[2] Joel C McCall and Mohan M Trivedi. Video-based lane estimation and tracking for driver assistance: survey, system, and evaluation. *IEEE transactions on intelligent transportation systems*, 7(1):20–37, 2006.

[3] Ardalan Vahidi and Azim Eskandarian. Research advances in intelligent collision avoidance and adaptive cruise control. *IEEE transactions on intelligent transportation systems*, 4(3):143–153, 2003.

[4] Chan Yee Low, Hairi Zamzuri, and Saiful Amri Mazlan. Simple robust road lane detection algorithm. In *Intelligent and Advanced Systems (ICIAS), 2014 5th International Conference on*, pages 1–4. IEEE, 2014.

[5] Yue Wang, Eam Khwang Teoh, and Dinggang Shen. Lane detection and tracking using b-snake. *Image and Vision computing*, 22(4):269–280, 2004.

[6] Karl Kluge and Sridhar Lakshmanan. A deformable-template approach to lane detection. In *Intelligent Vehicles' 95 Symposium., Proceedings of the*, pages 54–59. IEEE, 1995.

[7] ZuWhan Kim. Robust lane detection and tracking in challenging scenarios. *IEEE Transactions on Intelligent Transportation Systems*, 9(1):16–26, 2008.

[8] Christian Lipski, Bjorn Scholz, Kai Berger, Christian Linz, Timo Stich, and Marcus Magnor. A fast and robust approach to lane marking detection and lane tracking. In *Image Analysis and Interpretation, 2008. SSIAI 2008. IEEE Southwest Symposium on*, pages 57–60. IEEE, 2008.

[9] Lei Xu, Erkki Oja, and Pekka Kultanen. A new curve detection method: randomized hough transform (rht). *Pattern recognition letters*, 11(5):331–338, 1990.

[10] Jiang Ruyi, Klette Reinhard, Vaudrey Tobi, and Wang Shigang. Lane detection and tracking using a new lane model and distance transform. *Machine vision and applications*, 22(4):721–737, 2011.

[11] Massimo Bertozzi and Alberto Broggi. Gold: A parallel real-time stereo vision system for generic obstacle and lane detection. *IEEE transactions on image processing*, 7(1):62–81, 1998.