Real-time lane detection and tracking using particle filtering in an autonomous car 1:10 scale

De Martini Alessandro *970804-2479, E–mail: aledm@kth.se*
Morettini Simone *961202-1254, E–mail: simonemo@kth.se*

***Abstract*—This paper presents a Lane-detection-and-tracking algorithm based on a particle filter algorithm. The method we present has been tested on a vehicle model equipped with a Raspberry Pi and camera module. A lane-detection system is an important component of many autonomous vehicles, in fact, it is the base for many control algorithms (e.g., lane following, lane centring). The particle filter algorithm is used both for lane detection and lane tracking, and it permits to analyse in real-time each video frame collected by the monocular forward-looking camera. The filter outputs are two spline markers. A new weight algorithm based on the colour of pixels in images is the main contribution of the paper. The proposed lane detection algorithm is verified and evaluated under various test scenario. From the experimental results, we conclude it provides sufficient accuracy and reliability for a scale 1:10 vehicle application that runs in a protected environment.**

***Index Terms*—Particle Filter, Lane detection, Intelligent vehicle, Autonomous driving.**

I. Introduction

Autonomous guided vehicles (AGVs) have found applications in many industries. Their utilization varies from hospitals as transportation for patients to automatic warehouses. In most applications, the vehicle has to navigate in a not structured environment. All the information for pathfinding and navigation control are acquired using one or more camera mounted on the vehicle. All the images acquired are analysed using an algorithm specific for each application. Even if autonomous vehicles are still deployed only in industries, they are starting to enter the private market. Many companies are currently investing on them and they are planning to lunch a fully autonomous vehicle (L4) within the next years. However, currently, many *advanced driver-assistance systems (ADAS)* are helping to enhance the safety of driving using several modern technologies. Detecting and localizing lanes from a road image is a fundamental component of all the vehicles already described.

Lane detection has been an active topic in the research community for many years [1]. Many algorithm for detection and tracking, from Machine learning [2] to Hough transform [3] to probabilistic fitting and Kalman filtering [4] has been proposed. One of the most important features of these algorithms is the computational speed. Most of the time the embedded system where they are implemented requires to work in real-time. Therefore, this paper presents a real-time algorithm implemented in an autonomous vehicle provided by Bosch [5].

The implementation consists of two parts, *lane detection* and *lane tracking*. Both of them are implemented using a Monte Carlo localization algorithm which permits the creation of a lane model over the image provided by the camera mounted on the ego-vehicle. Going into some details, while the first part of the algorithm refers to the lane recognition using a bunch of different splines, the second permits to have an accurate result on the lane revelation introducing the knowledge of the past.

The benefit of our contribution is the possibility of implementing in Python an algorithm with low computational time on a Raspberry Pi 4.

The algorithm we present is constituted of three main blocks. The first part is the image pre-processing, where the image coming from the camera streamer is prepared for the followed parts. After that, the particle filter plays an important role both for the second and the third stages (lane detection and lane tracking). The most important part of our contribution will be the filter weighting which consists of a comparison between pixels under the artificial lane created by the filter and the real one captured from the camera. Finally, our system, implemented in Python is tested in real-time on a Raspberry Pi 4. All the algorithm performances should be analysed considering different filter parameters (blurry, line model, IPM image transformation usage).

The rest of the paper is organized as follows. Section II summarise the related research for lane detection and tracking. Section III deeply describe from the theoretical point of view the two parts of the algorithm. Section IV describes how the algorithm is implemented in Python and what are the main parameters which characterize it. Section V describes the filter parameters tuning and the experimental results for the proposed algorithm. The final section provides the conclusions and some ideas for future works.

II. Related works

Lane detection is a necessary component of an autonomous system environment. Our research started while we were working for an international challenge organized by Bosch [5]. The competition aims to build a working autonomous vehicle that performs better than the others, according to a scoring system based on different car ability (parking, lane-following, passing a roundabout).

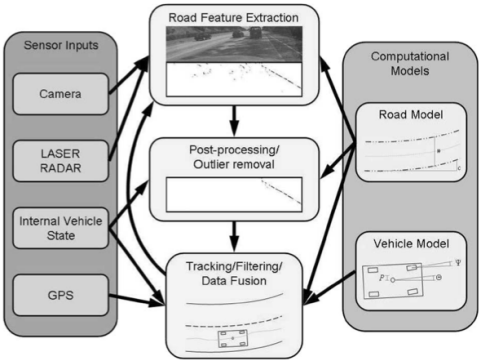


Fig. 1: Generalized flowchart of lane detection systems [6].

Therefore, lane detection is one of the bases for making the vehicle working. As another challenge, we decided to create an algorithm which mostly relies on a Monte Carlo Localization algorithm, less based than possible on computer vision algorithms.

Lane detection algorithms have been deeply studied with many different methods in the past. McCall and Trivedi [6] proposed a common diagram for generalizing the system flow of these algorithms (Fig. 1). In general, almost all the algorithms developed until now followed this flowchart.

A. Road Modelling

Most approaches propose at least one road model for representing lane markers. If the model chosen is right, it could significantly boost the performances by eliminating the outliers ignoring evidence generated by noise. The model could be represented either by a straight line or by a curve.

B. Lane Feature extraction

While road modelling is considered as a supporting step for the algorithm the *Lane Feature Extraction* is essential and can either completely change positively or negatively the final result of the whole system. This stage may also depend on the condition of the working environment. Many techniques have been collected by the research community.

The edge-based techniques is the most popular approach and rely on the contrast present between the road surface and the lines [7] [8]. These generally performed goodly, however sometimes contrast fool and some auxiliary steps may be additionally applied to increase their effectiveness (Inverse Perspective Mapping (IPM), blurring, thresholding) [9]. Even if, the Image processing we wrote is based on *Edge Detection* other techniques are available. For example Machine learning [10] or Stereo vision techniques [11] could also be applied for Lane Feature extraction.

C. Lane Detection

Referring to Fig. 1 this stage is also called Post-Processing and it is necessary to estimate the line marking positions. It is based on the knowledge about line markings received after the *Lane Feature Extraction* and the proposed road model. The most common techniques in literature used at this stage are Hough Transform and RANdom SAmple Consensus (RANSAC).

a) ***Hough Transform****:* Because of its speed and effectiveness in line detecting Hough Transform is an extremely popular technique used at this stage. The research of Wang et al. [8], Aly [12] and Duan et al. [3], all apply this technique in their approach. The disadvantage of the Hough Transform is that it is extremely useful for detecting straight lanes in the image, but it fails when bends occur. Therefore, many authors used it in combination with other techniques for improving the results. In [8], Wang et al. used an algorithm called CHEVP, which is made by the combination of Canny edge detector and Hough Transform.

b) ***RANSAC****:* This method is famous for outlier removal and model robust fitting. In some research [12], RANSAC is used in combination with Hough Transform for a line or spline fitting. In [10], Kim used this approach to find the hypotheses of lane markers among many noises.

c) ***Multiple random hypotheses****:* This method is less popular than the ones just described, but it seems to be extremely simple and effective for finding the best lane markers hypotheses in the processed image. The idea is to generate many hypotheses over the image and assign a weight to each sample to describe how good it represents the line. The advantage of this method, as shown in [9], is that the Lane Detection output could be directly used in the tracking algorithm.

D. Lane Tracking

The lane tracking stage is not always present in Lane detection algorithms. For example, in [12] the approach presented works on a single frame instead of presenting a tracking stage. However, with a tracking algorithm application, the speed and the accuracy of the system has been improved significantly. Particle Filter and Kalman Filter algorithms are often used at this stage.

a) ***Kalman Filter****:* Considering a strong set of assumption, the Kalman Filter [13] could work with high speed and accuracy in many situations. However, in reality, a lane detection algorithm has to deal with many not ideality, such as: road high curvature; noise left after feature extraction; false or missing evidence in lane detection, etc. In these cases, the Kalman Filter result becomes very inaccurate.

b) ***Particle Filter****:* Particle Filter could provide a solution overcoming Kalman Filter limitations. Its effectiveness and speed depend on the number of particles chosen. Among all the authors Berriel et al. [9] used the particle filter for both line detection and tracking, while others, such as [10] used the particle filter only in the Lane Tracking stage.

As described in this section a huge number of algorithms

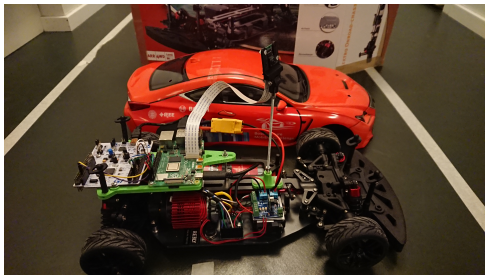


Fig. 2: Vehicle equipped with the Raspberry Pi Camera.

could be applied for dealing with lane detection problems. However, some factors in any research should be considered before deciding which algorithm to use: system objectives, working environment, and sensing modalities. Considering these factors and the aim of the Bosch Challenge [5] we decide to consider for our algorithm an Edge-based technique for feature extraction and a particle filter for both line detection and tracking. The road modeling instead has been kept general and open for an accuracy evaluation in the conclusions.

III. Lane detection Method

The lane detection system described in this section works on a sequence of images coming from a monocular forward-locking camera mounted on the top of the vehicle as shown in Fig. 2. For each image analysed singularly the algorithm outputs informations describing lane markers. Since the image would be split in the right and in the left part the filters outputs two markers, one for each side. Fig. 3 describes how the lane marker tracking system works. Our method processes one image at a time and each lane marker individually. Initially, each image is pre-processed for extracting lane marking evidence, removing unnecessary parts, and correcting perspective distortions. Finally, a particle filter algorithm has been implemented to average both the contributions of the currently found evidence and the lane markers detected in the previous frame.

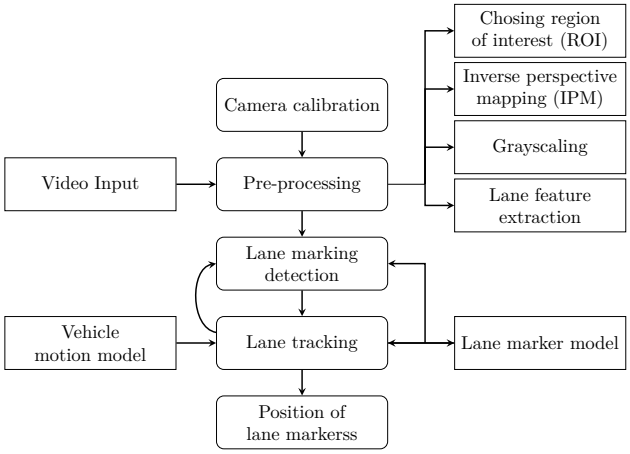


Fig. 3: Lane marker tracking system flowchart.

Before going into the flowchart details would be necessary to mention that camera calibration is not a part of the working system. It includes everything related to the image acquisition such as position, view angle and internal parameters of the camera (focal length, scaling factors and lens distortion). Correctly defining all those parameters could significantly help to increase the accuracy of the system. Since these parameters are usually empirically decided before, many papers do not mention this part.

A. Pre-processing

The pre-processing stage input is a sequence of frames captured by the monocular forward-locking camera. It extracts the evidence of lane markers from the raw input image for providing it to the next stage. Each frame is firstly cropped as shown in Fig. 4a for choosing the region of interest (ROI). The ROI is useful to remove irrelevant parts of the image (e.g. sky, horizon, etc.), the road region, in fact, is quite small compared to the whole image. The reduction of the data that should be processed increases the speed of the system.

Secondly, an inverse perspective mapping is applied [14] for transforming the camera image to a bird's-eye-view image (Fig. 4b) . Its benefit is the removal of the perspective effect so that in IPM images lane markers are vertical and parallel. The IPM is a geometrical transformation that remap one figure into the other according to (1). A 3x3 homography matrix (H in (1)) is required for transforming the points from the original image to bird's-eye-view.

$$(x', y', z') = H \cdot [x \ y \ 1] \tag{1}$$

Since the ground plane is assumed to be constant in the input image, H should be estimated using four fixed, manually selected, points. These points should be two on the left and two on the right of the lane markers. The image is then converted from colour to grayscale for reducing the number of channels improving the performances. Moreover, at this stage, the brightness and the contrast are corrected to reduce the ground reflection. As a final stage, a blurring is applied both to help the filter with dashed lanes and because it is useful in the particle filter weight stage. The IPM image, cropped, grayscaled and blurred is the final result of the pre-processing stage. It is expressed as a matrix, in (5) it is called *pdf*.

B. Particle filter

Particle filters [13] or Sequential Monte Carlo (SMC) methods are stochastic sampling approaches used for estimating the posterior density of the state-space. It approximates the posterior by a finite number of parameters. All the particles are sampled from a given distribution and each of them has a weight that represents its probability of being sampled from the density function. Even if this representation is approximate, since it is nonparametric, it could represent a much broader space of distribution than Gaussians.

(a) Cropped highlight.

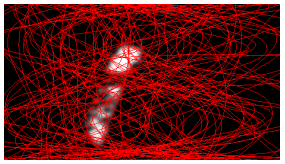
(b) IPM Image.

Fig. 4: Two phases of filter pre-processing.

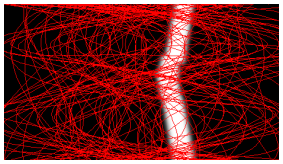
Essentially, the filter consists of four consecutive steps: initialization, prediction, resampling and weighting. First hundreds of lines are randomly sampled from a given distribution, all the weights are set equal at this stage. Subsequentially, the prediction step is applied thanks to a vehicle motion model for estimating the following position of the new particles. The resampling step generates new particles that are evaluated according to an error function. The weight represents the particle probability of being sampled for the next iteration of the filter. Resempling, thus, assures a higher probability of living to the particles with higher weight, while all the particles with lower weights tend to die.

In this paper, two independent particle filters were used, one for each lane marker. Before starting to describe particle filter steps is necessary to explain which line marking model has been considered for the algorithm described.

1) Line marking model: SMC methods permit to deal with many different types of particles because what matters is the idea behind the particle update. For this reason, the particles have been created as splines. Therefore, each of them is defined as an N-dimensional point (N_c), where N represents the number of the control points c of the spline and $c = (x, y)$ with x and y the control points coordinates. A particle is defined by the equation (2), x_c are randomly sampled while y_c are disposed equally spaced according with picture hight h. The number of control points could reach the number of pixels in the image, however, boundaries are restricted as shown in (2)



(a) Left half.



(b) Right half.

Fig. 5: Spline sampling in two binary images.

for computational efficiency constraints.

$$\begin{aligned} p &= \{(x_1, y_1), (x_2, y_2), \ldots, (x_{N_c}, y_{N_c})\} \quad 2 \leq N_c \leq 4 \\ y_i &= i \ \frac{h}{N_c - 1} \qquad 0 \leq i \leq N_c - 1 \end{aligned} \tag{2}$$

2) Particle Initialization: Particle filter initialization creates hundreds of lines over the binary half image. More is the number of particles used, more is the area of the image covered and more are the chances of finding the correct lane. Considering the line marking model described at the previous point we need to estimate the initial lane candidates estimating the position of the spline points in (2). As shown in Fig. 5 the particle filter is initialized with a set of random particles which are expected to cover the whole image. In hight, the image is already covered considering the distribution of y_i in (2). Each x_i will be instead sampled from a Gaussian distribution having the mean (μ_{init}) as the centre of the image and the standard deviation σ_{init} large enough for covering a great part of the image width. The distribution we sampled from is decided to be Gaussian since the line is more likely to be in the center of the image. Fig 5 shows an example of 100 particles initialized in each half of the IPM binary image.

$$\begin{aligned} x_i &= \mathcal{N}(\mu_{init}, \sigma_{init}) \qquad 0 \leq i \leq N_{samples} \\ \sigma_{init} &= \frac{width}{3} \end{aligned} \tag{3}$$

3) Lane Prediction: The lane prediction step, also know as the lane sampling step consists of the prediction of the lane position knowing the previous step output and the motion model of the lane. The lane is not moving in a fixed frame, but it is, if we consider a moving frame based on the camera. Therefore, a random amount sampled from a Gaussian distribution is added to the x coordinates of each control point. Instead, y coordinates remain fixed since the line is in all the image hight. The standard deviation is different for each control point of the spline. The further control points tend to move more than the closer, thus they require a bigger standard deviation. The equation in (4) was found empirically and, given a number of control points between two and four, it assigns to each of them a different standard deviation.

$$\begin{aligned} x_i^{new} &= \mathcal{N}(x_i^{old}, \sigma_i) \qquad 0 \leq i \leq N_c \\ \sigma &= ((N_c - 1) - i) * 40/N_c + 5 \end{aligned} \tag{4}$$

4) Weights Update: The particle filter weighing is fundamental for making the filter converging to the right line. At this stage, a set of particles is already estimated by

(a) Hight score particle. (b) Low score particle.

Fig. 6: Examples of spline weighting.

the previous step, but they all have the same weight. To calculate the weight of a particle an error function should be defined. It should be able to define how a particle is near to the real line. The function proposed in this paper is computationally simple, but effective. Defined a spline (Q) as an interpolation function of N_{int} interpolation points, its score is the number of white pixels it and its neighbour pixels contain. The number of interpolation points is a parameter that together with the number of particles should be selected by evaluating both computational time and accuracy. When the equation of the spline is found we can easily compare each point of it with the binary image. As already specified at the pre-processing stage the binary image is expressed as a matrix *pdf*. At this point should be clear why we decided to blurry the image in the processing. It permits to have pixels values between zero and one, helping to have a wide range of spline weights. In (5) we consider a square $3x3$ pixels around all the spline interpolation points for computing the score.

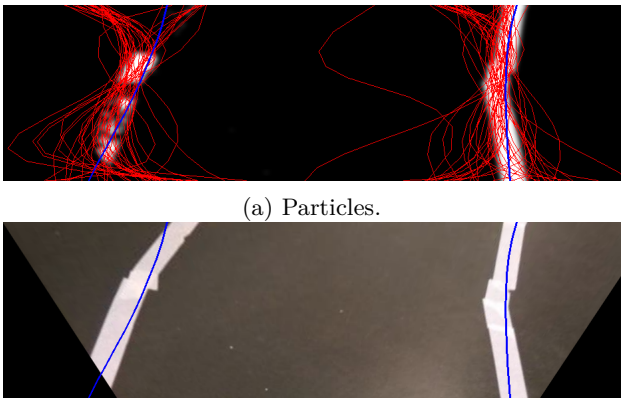
$$\begin{aligned} Q &= \left\{ \left(x_{p_1}, y_{p_1}\right), \left(x_{p_2}, y_{p_2}\right), \ldots, \left(x_{p_{N_{int}}} y_{p_{N_{int}}}\right) \right\} \\ W_Q &= \sum_{i=0}^{N_{int}} \sum_{j=-1}^{1} \sum_{k=-1}^{1} pdf_{x_{p_i}+j, y_{p_i}+k} \end{aligned} \tag{5}$$

5) Particle Resampling: Filter convergence capability highly depends on the resampling step. Resampling the particles which will survive in the next iteration permits to avoid degeneration problems. This phase transforms a particle set of $N_{samples}$ into another particle set of the same size. It is performed considering the particle weights and, whereas, before the resampling step, particles were distributed according to a predicted distribution, after that phase they would be distributed approximately according to the posterior. Since it forces particles back to the posterior it helps the convergence. This paper uses the *Stochastic Resampling* described in [13]. This method permits to spread particles enough avoiding wrong too fast convergences.

6) Threshold restart: To guarantee a good detection and tracking result a re-detection criterion was introduced. The best line predicted should have a score greater than a threshold. When the street rapidly change the tracking phase based on the previous frame would predict the line wrongly. In this case is useful to reinitialize the filter from scratch. The reinitialization happens after a pre-decided number of steps. Another case where it would be useful is in the case where there are no lane markers in the image. The particle filter could assume, in fact, two states: enabled or disabled. When the evidence score in the processed image is low the filter is disabled since it would not be able to detect any line. When evidence increase in the image the filter status switch from disabled to enabled and it restarts. Being restarted means it starts to predict the lane from the particles used in the last iteration. When the error stays high, the filter would be reinitialized according to the threshold restart.

C. Lane marker

As described by the flowchart Fig. 3 the final step of the Algorithm is the output definition. The position of a final lane marker should be defined by the tracking algorithm. The final lane extracted will be used in a control algorithm. Two methods were defined to extract a final lane marker (blue line in Fig. 7). The first simply consists of the selection of the line with the higher score. The second, instead, consists of creating a new particle: its x coordinates are computed averaging all the x coordinates of the particles that survived after the resampling stage. The second method works only if the filter converges, otherwise, it creates a new random particle.



(a) Particles.

(b) Output image.

Fig. 7: Lane marker after tracking stage in IPM image.

IV. Implementation

The implementation of the work was simulated using Python3 since it permits the better integration of this module with all the other algorithms. In particular, most of the image processing techniques in the paper are realized using the Open Source Computer Vision Library (OpenCV). Many coding inspirations have been found in G. Bradski and A. Kaehler book [11]. Moreover, the *Numpy library* was used for facilitating matrix operations and the *Scipy library* for creating the spline interpolation.

The simulation approach is made in real-time to facilitate a real application of the algorithm. In fact, the final implementation has been mounted in the available embedded hardware Raspberry Pi 4 shown in Fig. 2. Our algorithm has been tested to work with input video at

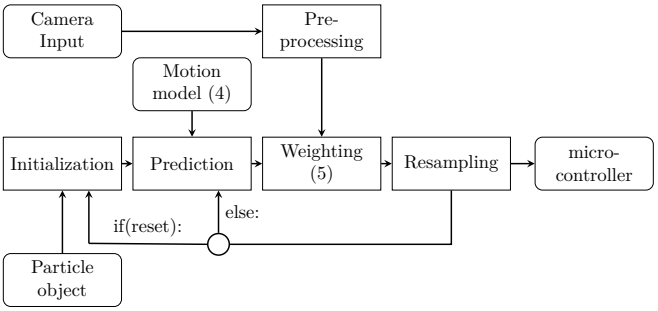
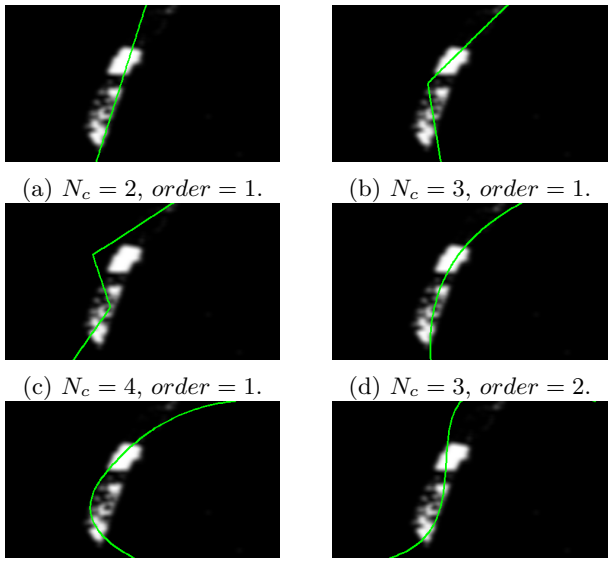


Fig. 8: Particle Filter software structure.

quality 480 p and aspect ratio 4:3. The frame rate was set to 15 fps. The input data provided by the forward-looking monocular camera are processed by the lane detection program and then sent to the micro-controller *NucleoF401RE* and used for the Automated Vehicle Control System.

As shown in Fig. 8 the algorithm has been implemented in one *main script* with some of the particle filter steps linked to the image pre-processing library. Moreover, all the implementation is based on the *particle object* defined in another file which is in charge of particles management. Each object is constituted of a list of points and a weight. The number of points in the lists depends on the control points number already described in section III. Moreover, this object contains the `generateSpline` method which, having the number of interpolation points (N_{int}) as input, manage the spline usage.



(a) $N_c = 2$, $order = 1$. (b) $N_c = 3$, $order = 1$.

(c) $N_c = 4$, $order = 1$. (d) $N_c = 3$, $order = 2$.

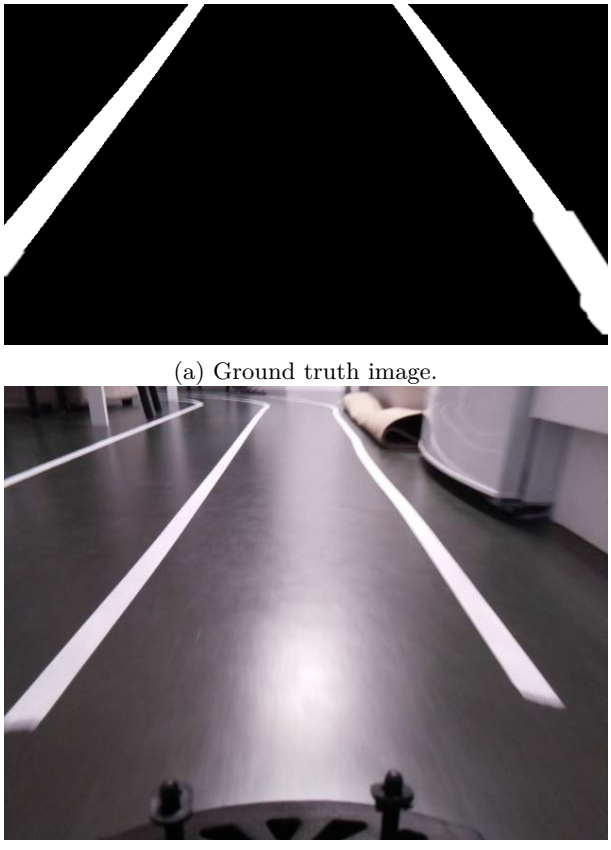
(e) $N_c = 4$, $order = 2$. (f) $N_c = 4$, $order = 3$.

Fig. 9: Different lane markers.

The particle object have an important role in the algorithm we created. It permits to streamline the code and it defines the lane marking model mentioned in Fig. 3 with some input parameters. The object programming approach simplifies the debug, the readability of the code and the analyses of the system implemented, but it adds some extra loop to the normal execution that could be avoided using some matrices. The algorithm has been created for a general marking model. Basically, it could be either a line or a cubic spline just specifying it as a particle filter input. The idea behind this generalization was raised with the necessity of testing the algorithm with different parameters for studying how the accuracy behaves. The input parameters are two: the line order (*order*) and the number of control points (N_c). Considering the number of points varies between two and four and the order between one and three the types of line tested has been six (Fig. 9). The order could not be greater or equal to the number of points.

V. Experimental Results

In order to validate our system, we ran a set of experiments using datasets with different road characteristics. The experimental environment is the one specified the Bosch Challenge guide [5]. Thus, having a dark floor and well defined white lines made the pre-processing stage easier. All the lines were made using white tape which is in contrasts with the floor. Moreover, very few obstacles were present in the experiments performed. However, some complexity was added by the high reflectivity of the floor which sometimes make difficult to find lane markers.



(a) Ground truth image.

(b) Raw image.

Fig. 10: Ground truth used for quantitative results.

Quantitative and qualitative results, under different conditions, are reported. The results are compared with a set of *ground truth datasets* which has been created

modifying the pre-processed images with a line easy to be detected (Fig. 10). Therefore, the lane detection result obtained with the ground truth dataset is the result we aim to obtain with the raw images. The difference between them defines the accuracy of our method.

TABLE I: Dataset Overview.

Dataset	Frames	Dataset	Frames
Dataset 1	117	Dataset 3	154
Dataset 2	394	Dataset 4	84

A. Set-up and Datasets

We used four different datasets with a total of 749 frames (Table I) entirely acquired by us with 640x480 pixel resolution. As already mentioned each dataset consists of different key-points that could be encounter by the vehicle in the competition. In particular:

- **Dataset 1:** Most of the frames consists of a straight lane defined by continuous lane markers.
- **Dataset 2:** The lane detected is mostly straight as in the previous dataset, but now one of the lane markers is dashed. Moreover, a simulated overtaking and a reverse is performed.
- **Dataset 3:** the vehicle turns passing throw another lane marker.
- **Dataset 4:** As in dataset 3 the vehicle turns passing throw other lines, but now a high level of ground reflection is present.

The tests were performed in a laptop with Core i5-5257U (2x2.7 GHz) 4 GB RAM 1867 MHz.

(a) Dataset 1, straight lane.

(b) Dataset 2, dashed line.

(c) Dataset 3, bend.

(d) Dataset 4, reflection.

Fig. 11: Datasets particulars.

B. Tuning

The system was tuned using two measures: execution time and lane detection accuracy. These are the two most significant parameters since they give an idea of the system performances. A set of metaparameters needs to be tuned in order to get the best out of the proposed system:

- *Number of particles ($N_{samples}$):* has a direct influence on algorithm performances and permits to balance the accuracy and the execution time;
- *Number of interpolation points (N_{int}):* since it has a direct influence on the weighting function it is strongly related with the model accuracy;
- *Number of control points and spline order (N_c and order):* N_c defines how the lane output is flexible while the order defines how the spline curvature is;
- *IPM usage:* as described in section III IPM permits to have straighter lines in the image, but does it positively influence system performances?
- *Reinitialization steps:* given a fixed threshold, they are the number of steps that the system should wait to reinitialize the filter.
- *Blur kernel size, brightness and contrast:* these are parameters used at the pre-processing stage (Fig. 3). A blur is added for facilitating the weighting and for helping with dashed lines while contrast and brightness are changed for adjusting the light conditions.

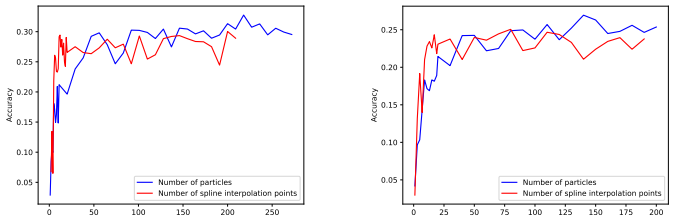
Tuning experiments were not executed in all the datasets. Sometimes, in fact, only the first two datasets was used because they are more meaningful and they contain more frames then the others. Moreover, the execution time was considered only with the first two metaparameters because all the others do not excessive influence its value.

1) Number of particles: We tested a range of particles between 10 and 500 considering both the execution time and the accuracy. In Fig. 12 the blue line shows how the accuracy looks like for the first two datasets in a limited range of particles. Instead, the *execution time-N_{sample}* dependency is represented in Fig. 13 and, differently from Fig. 12 it shows a linear dependency. Thus, firstly the real-time constrain should be consider: receiving as input 14 fps from the camera the processing time should be less or equal then 0.071 s/frame. Moreover, the direction of the blue line in Fig. 12a have an asymptote for $N_{sample} > 56$, while the one in Fig. 12b presents an asymptote after $N_{sample} > 50$.

2) Number of interpolation points: The number of spline interpolation points has been evaluates for values between 2 and 300 with the first two datasets, their results are collected in Fig. 12 and in Fig. 13 and they are represented with the red line. Fig. 13 clearly show a linear relation *time-N_{int}*. However, differently from the blue line, the execution time increases more rapidly with less benefit on the accuracy side. Considering the real-time behaviour of the system and the asymptote a good choice is $N_{int} = 11$ for the first dataset and $N_{int} = 17$ for the second.

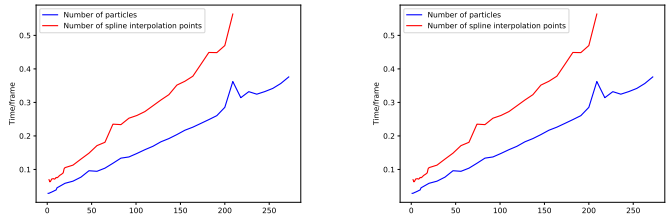
Table II shows how much is the computational time with the chosen parameters.

3) Number of control points and spline order: In order to discover which kind of lane marker is more suitable for our experiment, a simulation with all the types shown in Fig.



(a) Dataset 1. (b) Dataset 2.

Fig. 12: Accuracy N_{sample} and N_{int} dependency.



(a) Dataset 1. (b) Dataset 2.

Fig. 13: Time N_{sample} and N_{int} dependency.

TABLE II: Execution time with chosen parameters.

Dataset	**Particles**	**Interpolation points**	**Execution time [s/f]**
Dataset 1	56	11	0.075
Dataset 2	50	17	0.082

9 has been performed, for the first two datasets. Table III underlines that the best performances are achieved with three control points both with the first and the second order.

TABLE III: Accuracy without IPM usage.

Dataset 1	$\mathbf{N_c = 2}$	$\mathbf{N_c = 3}$	$\mathbf{N_c = 4}$
Order = 1	0.26	0.31	0.29
Order = 2	None	0.31	0.28
Order = 3	None	None	0.27

Dataset 2	$\mathbf{N_c = 2}$	$\mathbf{N_c = 3}$	$\mathbf{N_c = 4}$
Order = 1	0.34	0.31	0.28
Order = 2	None	0.28	0.2
Order = 3	None	None	0.24

4) IPM usage: The results collect after a simulation using the first two datasets with or without the IPM usage shown that the eagle-eye view is not as effective as it seems. The main problem is that the four fixed points should change each time the camera modifies the street view. IPM really boosts performances in some moments, for example, if the lane markers are straight the algorithm converges really fast. However, on average (table IV), performances remains equal. Sometimes the IPM transformation cut an important part of the image and the filter could not converge.

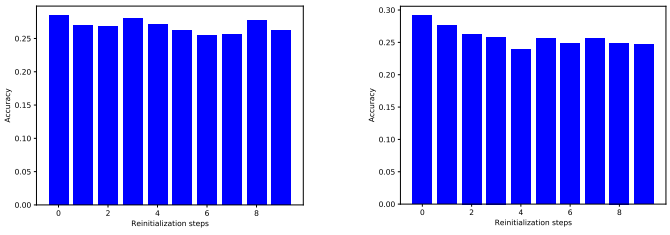
TABLE IV: Accuracy with IPM usage.

IPM - Dataset 1	$\mathbf{N_c = 2}$	$\mathbf{N_c = 3}$	$\mathbf{N_c = 4}$
Order = 1	0.30	0.30	0.27
Order = 2	None	0.29	0.24
Order = 3	None	None	0.30

IPM - Dataset 2	$\mathbf{N_c = 2}$	$\mathbf{N_c = 3}$	$\mathbf{N_c = 4}$
Order = 1	0.30	0.29	0.25
Order = 2	None	0.28	0.23
Order = 3	None	None	0.08

5) Reinitialization steps: As described in section III the threshold is the metaparameters which permits to activate or deactivate the filter, it works according to the image score. It also permits us to decide whether to reinitialize the filter or not. Since the threshold is a fixed number (20% of the maximum weight) what we need to decide for the filter usage is: if the score is under the threshold, after how many steps should we reinitialize the filter?

Intuitively, if the number of steps is too large the filter is never reinitialize. On the contrary, if the number of step is zero the filter could not converge because it is continuously reinitialized. Fig. 14 collects the results acquired with the first two datasets and with the parameters decided in table II. Moreover, the type of lane approximation was set as a cubic spline with four control points. Fig. 14 shows different performances with different *reinitialization steps* and, in particular, underline that in more complex environments (dataset 2) is preferable to reinitialize the filter rapidly. Considering that a number equal to zero is not allowed we find out it is preferable to maintain its value between two and five for both the datasets.



(a) Dataset 1. (b) Dataset 2.

Fig. 14: Reset threshold and accuracy relation.

6) Blur kernel size, brightness and contrast: The last step to perform for finish the tuning part is to define how the blur should be. The tuning has been performed with datasets 1, 2 and 3 since was preferable to have a grate number of differences in the different images. The last dataset was not use because the too high reflection compromised the results. The parameters used for all the datasets are:

- *Number of particles:* 56;
- *Number of interpolation points:* 11;
- *Number of control points:* 4;
- *Spline order:* 3;

- *IPM:* Not used;
- *Reinitialization steps:* 4;

The blur is a function performed in `image pre-processing` class (Fig. 8) which works on the *pdf matrix*. Therefore, it accepts two inputs *blur(kernel size*$*B_c$, *kernel size)* . For what concerns the two first blur experiments brightness and contrast have been kept equal and the hight coefficient (B_c) set equal to 3 in the first experiment and to 7 in the second, *brightness: -60, contrast: 100*. On the contrary in the final simulation brightness and contrast have been changed to brightness: -120, contrast: 127 while B_c was still equal to 3. The *blur kernel size* in all the simulations has been changed from 3 to 19 with an interval of 2. Having B_c greater permits to have a higher blur in y-direction permitting to understand better the dashed lines. For the three experiments, the more significant results are collected in table V.

TABLE V: Accuracy with different blur kernel matrices.

Kernel Size:	3	5	7	9	11	15	17	19
Experiment 1	0.26	0.24	0.24	0.25	0.24	0.21	0.23	0.20
Experiment 2	0.22	0.21	0.25	0.18	0.25	0.24	0.24	0.25
Experiment 3	0.13	0.13	0.14	0.121	0.14	0.14	0.13	0.14

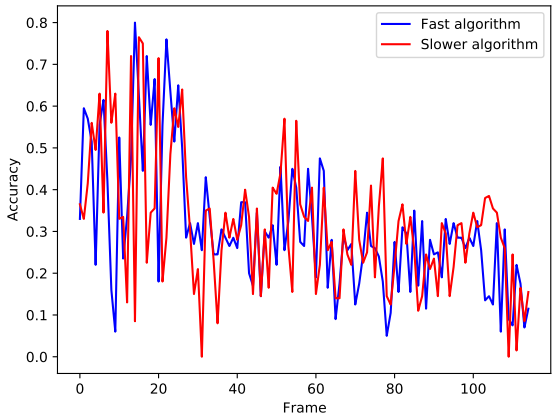
What emerged from the table V is not expected at all. The third experiment performed worst than others even if it should delete all the floor reflection. Moreover, between $B_c = 3$ and $B_c = 7$ the best solution seems to be the first. These results raised out due to some problems visible at the debug phase. Fig. 15b clearly shows that in the third experiments no more reflection is present, however, also the lines are in part deleted. Choosing the brightness and the contrast as in the first experiment, instead, underline the line presence. Finally, for reflection reasons, the high coefficient could not play his role and performances do not change too much.

(a) Dataset 1. (b) Dataset 2.

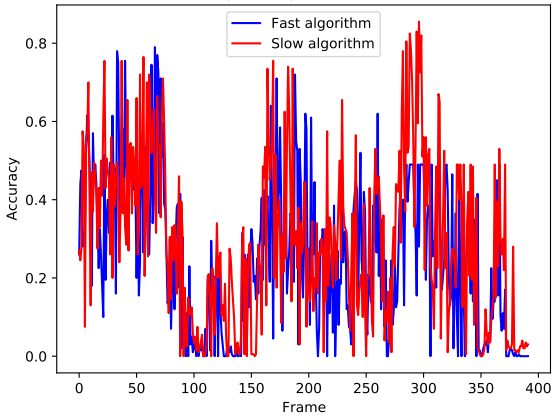
Fig. 15: Different kind of brightness and contrast.

C. Results

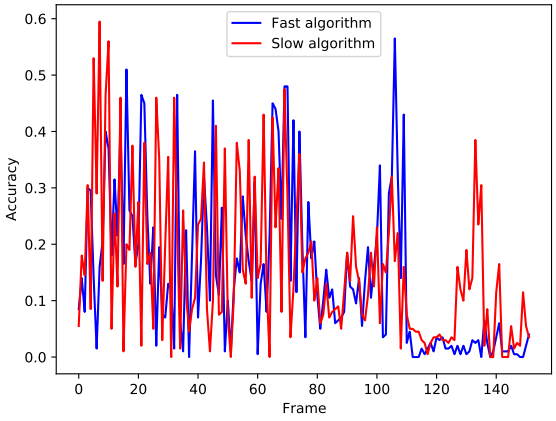
From the above-mentioned tuning, the best parameters, and experiments were run in all the datasets. Two types of experiments have been realized: the first is performed considering a real-time environment (blue lines in Fig. 16), the second privileging a higher accuracy (red lines in Fig. 16). The final results could be analysed both from the quantitatively and the qualitatively point of view.



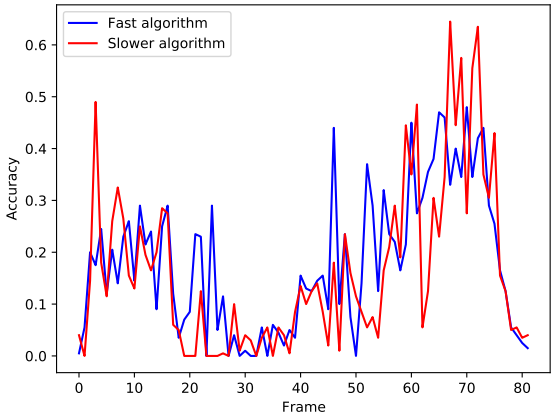
(a) Dataset 1.



(b) Dataset 2.



(c) Dataset 3.



(d) Dataset 4.

Fig. 16: Final accuracy with all the datasets.

- $N_{samples_{fast}}$: 56;
- $N_{int_{fast}}$: 11;
- $N_{samples_{slow}}$: 100;
- $N_{int_{slow}}$: 25;

- *Number of control points:* 3;
- *spline order:* 2;
- *IPM:* Not used;
- *Reinitialization steps:* 4;
- *Temporal blur kernel size:* 7;
- *Brightness:* -60;
- *Contrast:* 100;

1) Quantitatively: To evaluate the method quantitatively the main parameter used is the accuracy which is found as described at the beginning of this section. The execution time is another quantitative parameter that has been already analysed before. Finally, the lane detected percentage is also interesting, however it is strongly related to the accuracy. From the algorithm definition if the accuracy is more than 50 % the line is detected. Fig. 16 shows how the accuracy looks like for each frame in all the datasets. The execution time spent on finding the blue trend in Fig. 16 is ten times the one used for the red one. The first and the second datasets show high levels of accuracy, both with straight lanes and bend with a low radius of curvature. Analysing the final two datasets, instead, performances are really low due to the high reflection and the crossing lines. Moreover, the parameter tuning has been mainly performed with the first and the second dataset.

2) Qualitatively: As observed in the images presented in the paper, in particular in Fig. 7 and Fig. 9 the proposed system can adapt well to different kind of lane markers, this is the main motivation for choosing a spline model. The estimated lines are most of the time overlapped to the ones captured by the camera and this is sufficient for being the micro-controller input.

VI. Conclusions and future directions

This paper presents an approach to estimate lane markers from a single image. The algorithm we proposed aims to be used as a base for much autonomous driving application. The experiments we performed, even if they do not completely represent real-life experiments are a good approximation for them. They showed that our system can detect lanes in most of the frames for what concern straight lines and bends with a low radius of curvature.

However, the model described presents some limitations that seem to be reasonable for an application in the environment presented in [5].

Most of the ideas for future directions regard the *pre-processing stage* of the algorithm. In particular, the IPM application did not show the expected results. This happened mainly because we used a static homography matrix which presents some problems when the lane trend changes. Some of the camera parameters like the height and the viewing angle could be better selected according to the four points of the IPM transformation. Moreover, some of the results we described have been ruined by the high reflection of the floor and even if the floor proposed in the challenge is black the picture evidence map could be better performed applying a *threshold-based technique* [9]. From the particle filter side, an improvement could be performed by adding a better car motion model in (4) using the information provided by some encoders. Finally, for increasing the scientific validity of our project some tests using a standardize image ground truth [15] in real datasets [16] could be performed.

Even if the method presents the limitations just presented it can provide reliable estimates with a high level of confidence and with execution times low enough for real-time applications.

References

[1] Y. Wang, D. Shen, and E. K. Teoh, "Lane detection using spline model," *Pattern Recognition Letters*, vol. 21, no. 8, pp. 677–689, Jul. 2000.

[2] Z. Kim, "Robust Lane Detection and Tracking in Challenging Scenarios," *IEEE Transactions on Intelligent Transportation Systems*, vol. 9, no. 1, pp. 16–26, Mar. 2008, conference Name: IEEE Transactions on Intelligent Transportation Systems.

[3] D. Duan, M. Xie, Q. Mo, Z. Han, and Y. Wan, "An improved hough transform for line detection," in *2010 International Conference on Computer Application and System Modeling (ICCASM 2010)*, vol. 2. IEEE, 2010, pp. V2–354.

[4] B. Dorj, S. Hossain, and D.-J. Lee, "Highly curved lane detection algorithms based on kalman filter," *Applied Sciences*, vol. 10, no. 7, p. 2372, 2020.

[5] Bosch future mobility challenge, "BFMC's Challenge." [Online]. Available: https://boschfuturemobility.com/

[6] J. C. McCall and M. M. Trivedi, "Video-based lane estimation and tracking for driver assistance: survey, system, and evaluation," *IEEE transactions on intelligent transportation systems*, vol. 7, no. 1, pp. 20–37, 2006.

[7] P. Lindner, S. Blokzyl, G. Wanielik, and U. Scheunert, "Applying multi level processing for robust geometric lane feature extraction," in *2010 IEEE Conference on Multisensor Fusion and Integration*. IEEE, 2010, pp. 248–254.

[8] Y. Wang, E. K. Teoh, and D. Shen, "Lane detection and tracking using b-snake," *Image and Vision computing*, vol. 22, no. 4, pp. 269–280, 2004.

[9] R. Berriel, E. Aguiar, V. Vieira, and T. Oliveira-Santos, *A Particle Filter-Based Lane Marker Tracking Approach Using a Cubic Spline Model*, Aug. 2015.

[10] Z. Kim, "Robust lane detection and tracking in challenging scenarios," *IEEE Transactions on Intelligent Transportation Systems*, vol. 9, no. 1, pp. 16–26, 2008.

[11] G. Bradski and A. Kaehler, *Learning OpenCV: Computer vision with the OpenCV library*. " O'Reilly Media, Inc.", 2008.

[12] M. Aly, "Real time detection of lane markers in urban streets," in *2008 IEEE Intelligent Vehicles Symposium*. IEEE, 2008, pp. 7–12.

[13] S. Thrun, W. Burgard, and D. Fox, *Probabilistic robotics*, ser. Intelligent robotics and autonomous agents. Cambridge, Mass: MIT Press, 2005, oCLC: ocm58451645.

[14] R. Hartley and A. Zisserman, *Multiple View Geometry in Computer Vision*, 2nd ed. Cambridge University Press, Mar. 2004.

[15] A. Borkar, M. Hayes, and M. T. Smith, "An efficient method to generate ground truth for evaluating lane detection systems," in *2010 IEEE International Conference on Acoustics, Speech and Signal Processing*. IEEE, 2010, pp. 1090–1093.

[16] A. Geiger, P. Lenz, C. Stiller, and R. Urtasun, "Vision meets robotics: The kitti dataset," *The International Journal of Robotics Research*, vol. 32, no. 11, pp. 1231–1237, 2013.