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

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

***Abstract*—For autonomous cars, a key element is the lane detection and tracking. It is used as a main component of the lane follow algorithm. In this report, an algorithm for solving this task using Particle filter is proposed. A car in scale 1:10 with a single camera in a closed environment is used for gathering data and testing the algorithm. The main objective is to reach a high accuracy in lane detection with a high velocity of execution so the algorithm can be implemented in a real-time system. It produces two splines that represent the lane markers to be used in the control of the vehicle. The interesting contribution of this report is in the merge of the detection and tracking in the particle filter in the same step thanks to the weighting function based directly on the colour of the pixels of the probability density matrix. Using a ground truth manually created an analysis of the effect of the parameters on the algorithm is made. The results pointed out that the algorithm is able to find the lane makers with enough accuracy in the simplified environment.**

***Index Terms*—Lane detection, lane tracking, particle filter, autonomous driving, real-time.**

I. Introduction

Advanced Driver Assistance Systems (ADAS) and automatic driving system are used in diverse contexts, from moving robots in industries to the more complex task of autonomous car. It is common that the environment contains lane that the robots have to follow and so a lane detection algorithm is a key element for such systems. For the previous reason, found the lane position with respect to a vehicle is a very actual topic and it needs continuous improvements for more robust and light algorithms for running in embedded systems with limited computational power.

Lane detection and lane tracking are two fundamental tasks for autonomous driving. The first one refers to the analysis of an image to identify the lane. The second concept permits to introduce the history knowledge in the detection of the lane and so it permits to obtain more accurate and fast results.

These two tasks provide data to the control algorithm of the autonomous system for following the street and remaining in the correct position relative to the lane markers. The algorithm created is planned to be used inside a system for autonomous driving that will be developed for the Bosch Future Mobility challenge [1] and it is tested on the hardware provided by Bosh.

The objective was to create a light algorithm that could do lane detection and tracking in a Raspberry Pi using images obtained by a PiCamera in real-time. In literature this is obtained using three elements: road feature extraction, post-processing and lane tracking, as described in [2]. In this paper, a lighter method is proposed by merging detection and tracking in the particle filter.

The method implemented consist of two main parts. The first part is the image pre-processing that aim to create a probability density function(PDF) of lane markers starting from pictures. The second module is the particle filter that using the output of the first module do the lane tracking and detection. The computer vision part was implemented using an external library, instead, the second part was totally implemented by the creators of this project.

The outcome is an analysis of the accuracy of the algorithm an analysis of how different parameters of the computer vision(blurry, IPM) and of the particle filter (number of particles, number of interpolation points, particle representation) influence the accuracy and the execution time.

The paper is structured in seven sections of which the first one is this introduction. In section II an analysis of the research made in this field is reported. Section III describes the method used and the theory behind the algorithm in details: the first part regards the preprocessing of the data and the second part is focused on the particle filter implementation. Section IV contains a detailed description of the implementation made and an analysis of the technology and choice made. In section V the experimental results are reported. The report is closed by a conclusion section and an analysis of possible future improvements of the system implemented.

II. Related works

This project is aimed to be part of a wider system for creating an autonomous car able to drive in a simulated urban system and so a fundamental requisite for the vehicle is to do lane following. The principal aim was so to develop an algorithm based on Monte Carlo localization algorithms to do lane detection and tracking. The idea that guides our choices was to reduce the computer vision part of the system to make a fast algorithm that could run in small devices. In Literature the most common approach is the one summarized by McCall and Trivedi in [2] where three main steps are well defined: lane feature extraction, lane detection and lane tracking.

Before analysing the three elements in details, a definition of how the road is modelled is needed. Using a

model for the road permits to reduce the complexity of the algorithm since the use of information from the model will permit to reduce the space of solutions. In fact, without a model, every line in the pictures would be considered. The model usually used is a straight lane or curve lane. The second one is more general since the first one can well model only straight roads.

A. Lane Feature extraction

This step is a fundamental one for the system since it extracts information from the pictures that will be used in the following steps. The quality of this phase will significantly influence the outcome. A bad work in this stage will result in the wrong final results, though the other stages are working well.

The most common approaches used in this stage are edge-based techniques. They aim to recognize sharp discontinuity in images based on the intensity and contrast of the colour of the pixel in the pictures [3]. There are multiple techniques to perform the edge detection like Canny, Sobel, Prewitt or Roberts that works with the maximums and minimums of the first derivative of the image for adjacent pixels and the Laplacian methods instead that works with the value that approaches zero of the second derivative of the image [4]. Those approaches can be improved in performance by simplifying the problem and considering only vertical or almost vertical line and so comparing the pixels only with others in the same rows [5].

A different system for Lane Feature extraction is the usage of Machine Learning [6] that provides good result but it requires that the training set contains all the possible cases otherwise the system will not work in every situation.

B. Lane Detection

On this stage the *a priori knowledge* of the road model is used to reduce the number of lines detected in the previous step. One of the most common techniques used is the Houg Transform [7][8]. The high usage is due to the good result and the speed for lines detecting with the limit of the possibility to find only straight lines. The RANSAC method instead is very powerful for removing outliers and it works well with noises [6].

An alternative to the classical methods is the usage of Dynamic Programming that permit to have good performance for the real road with false candidate, noisy as shadows and other occlusions [9].

An effective and simple method is the one used in [10]. It is based on a set of hypothesis created randomly. The hypotheses are lines that are candidate solution to represent the lane markers and they are weighted based on the likelihood that they are a real lane marker. The tracking stage will reduce the number of hypotheses needed and make this approach work.

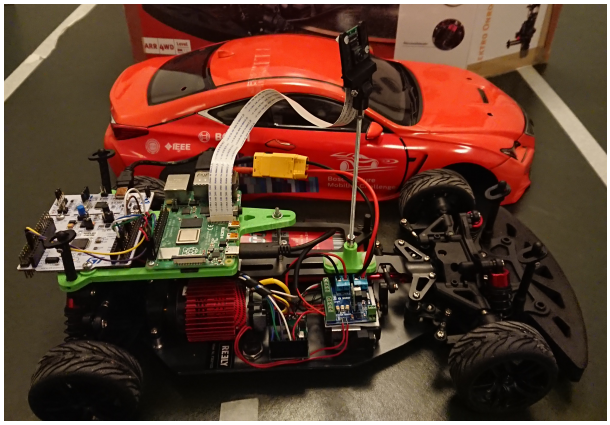


Fig. 1: Vehicle used equipped with the a Raspberry Pi 4 Mobel b and the PiCamera v2.1.

C. Lane Tracking

This step permits to use the state of the vehicle in the algorithm. The previous step works only on one picture, now in this step, the data from the past are used to approximate the lane with higher accuracy.

One of the most common approaches for Lane tracking is the Kalman Filter [11][12]. The advantages of using this filter are the velocity and accuracy that it provides. The disadvantage is that the result is not accurate when data is noisy or when it runs on complex and unexpected situations as it is common for lane tracking as it could be crossroad, obstacles or bends.

A better alternative is the Particle Filter. It uses a set of randomly generated particle that approximate the solution. The advantage is that it can describe complex states than the Kalman Filter. Usually, the Particle Filter is used only for tracking as in [13], but in [10] it was also used for lane detection.

However, the Lane Tracking is not always used as in [14] where the lane was detected on a single picture without using information for the past state of the vehicle.

III. Method

The system takes as input images obtained from a camera sensor, one at a time and produces as output two candidate lines that identify the street lane. The camera sensor is fixed on a forward locking position on top of a vehicle as reported in Fig. 1. The input is processed to produce a PDF of the street lines. So the initial colour image is transformed in a greyscale image where the intensity of a pixel indicates the probability that the pixel appertain to a street line. The particle filter used is the classic one. It starts with the initialization than sampling, weighting and resampling. The weighting is made using the PDF obtained in the initial step. All the steps are reported in the flowchart in Fig. 2.

A. Image processing

To extract the best PDF, the input image is processed through multiple steps as shown in Fig. 3.

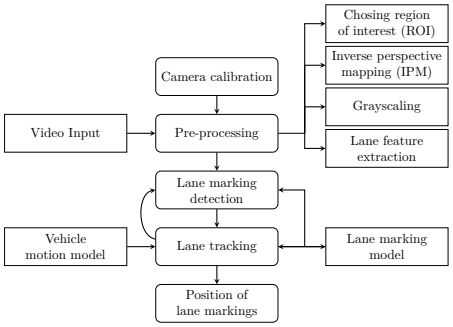


Fig. 2: Lane detection and tracking flowchart.

Initially, the image is cut so only the part of the image that contains the street is selected. The cut parameters are manually chosen before executing the algorithm. Choose the cutting points of the figure can be made without automation since the camera is fixed on the car and so the interesting part of the figure for controlling the car will be always fixed. The second cut operation is made to obtain two images, one that will contain the left line and one that will contain the right line.

An implemented module is used for transforming the image as if it was taken from a camera above the street parallel to it using an inverse perspective mapping(IPM)[15]. In this way, the perspective due to the camera position in the car is decreased. The transformation is made using a 3x3 matrix that does a remapping of the picture as shown in Equation (1).

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

Then the brightness and the contrast of the images are edited to facilitate the extraction of the lines and reduce the light reflections parts of the image that can cause a wrong PDF. Finally, a filter is applied to extract only the part of the picture that contains white and yellow colours that are the possible colours of the lines in the dataset used.

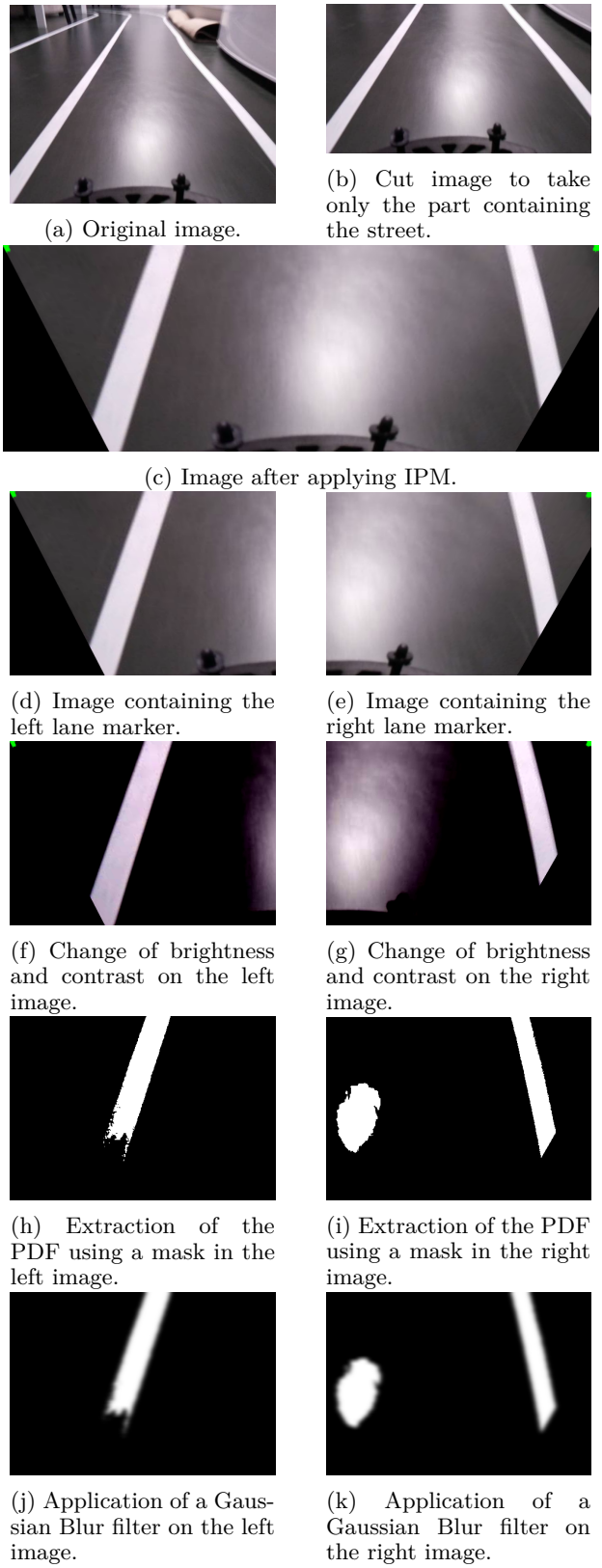
The last step of the image processing is the application of a Gaussian Blur filter to remove some noisy and to transform the mask obtained in a probability density matrix that will permit the particle filter to converge better.

B. Particle filter

The objective is to identify the street lines and so a particle is a line and it is represented as a set of points as in Equation (2).

$$p = \{(x_1, y_1), (x_2, y_2), ..., (x_{N_c}, y_{N_c})\} \quad N_c \geq 2 \tag{2}$$

where $(x_i, x_i), 0 < i \leq N_c$ are the coordinates of the point that identify the line. If $N_c = 2$, then a particle is a straight line, instead, if $N_c > 2$ the points are interpolated using a spline or an open polygonal chain. Therefore a particle is also defined by the *order* of interpolation and



(a) Original image.

(b) Cut image to take only the part containing the street.

(c) Image after applying IPM.

(d) Image containing the left lane marker.

(e) Image containing the right lane marker.

(f) Change of brightness and contrast on the left image.

(g) Change of brightness and contrast on the right image.

(h) Extraction of the PDF using a mask in the left image.

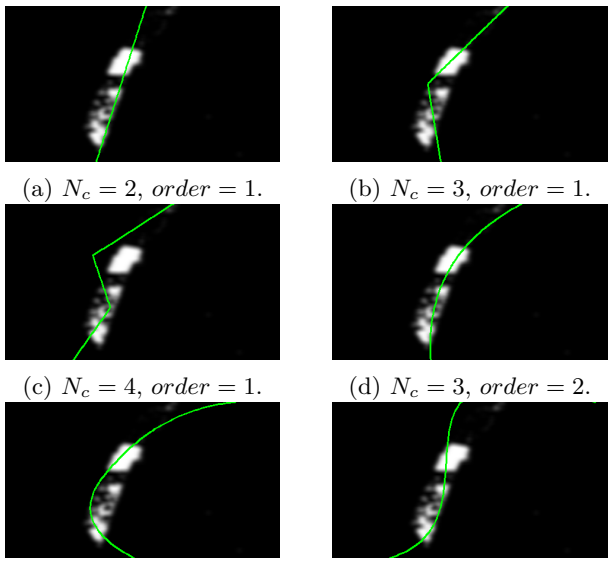
(i) Extraction of the PDF using a mask in the right image.

(j) Application of a Gaussian Blur filter on the left image.

(k) Application of a Gaussian Blur filter on the right image.

Fig. 3: The image pre-processing steps from the acquired picture to the two final PDF.

the number of control points N_c and by changing them different particles can be obtained as shown in Fig. 4. The number of particles used in the filter, in this report, will be denoted with N_p. The algorithm is implemented in a way that both straight lines and splines can be used. Using the particle filter on the straight lines permits to obtain a faster execution because is less computationally expensive to do the calculations but the spline has the advantage of approximating better curves instead of straight lines.



(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. 4: Particle with different orders and number of control points drawn on top of a PDF of a lane marker.

1) Particle initialization: This first step of the algorithm creates the initial set of particles by generating the points that describe each particle. The ordinate of the points is fixed so the points are uniformly spread for all the heigh of the image using equation (3). This choice comes from the model of the road and so from the fact that lane markers will go across the whole picture from top to bottom. The abscissa is selected randomly using a Gaussian distribution with mean $image\_width/2$ and standard deviation $1/3 * image\_width$.

$$y_i = (i-1) * image\_heigh/(N_c - 1) \quad 0 \leq i \leq N_c \quad (3)$$

At the end of this step, the weight of each particle is set to $1/N_p$.

2) Line sampling: In this step, the particle filter creates new particles by changing the x-coordinate of all the points of the particles. It does not act on the y-coordinate since it is always expected to have two lines that cross all the input image vertically. The x-coordinate is updated using a new value evaluated from a Gaussian distribution as expressed in equation (4). When the car turns, the points on the image that corresponds to the points more father from the car will move faster. To take that in consideration, σ is evaluated using the equation (5).

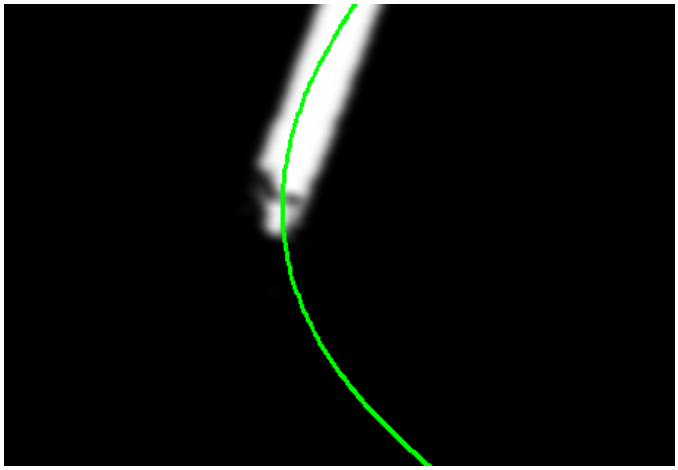


Fig. 5: Weighting example. The particle will receive half of the maximum weighting since the half upper part is above white pixels and the bottom part is above black pixels.

$$x_i^{(new)} = \mathcal{N}(x_i, \sigma_i) \quad 0 < i <= N_c \quad (4)$$

$$\sigma_i = ((N_c - 1) - i) * 40/N_c + 5 \quad (5)$$

3) Line weightings: This step is central to have a particle filter that converges to the right line. The approach used is computationally simple. The weight of a line is equal to the sum of the pixel colour of the line projection on the PDF as shown in Fig. 5. As a result of the previous image processing, the PDF is composed of pixels that have a value in the range $[0, 1]$.

A particle is weighted by passing through two steps. Firstly M points are created by interpolating the particle control points as expressed in equation (6).

$$Q = \{q_0, q_1, ..., q_M\} = interpolate(p, M) \quad (6)$$

Secondly, weights are evaluated by summing the value of the PDF at the coordinates of the points of Q as expressed in equation (7). The iteration over p and q are used for giving a thickness to the lines and help the algorithm to converge.

$$W_i = \sum_{i=0}^{M} \sum_{j=-1}^{1} \sum_{k=-1}^{1} pdf(q_{i,x} + j, q_{i,y} + k) \quad (7)$$

4) Line resampling: The resampling was made using the Systematic re-sampling [16]. This approach consists in choosing an initial random number and then select the particle using a fixed distance. It was chosen because is fast since only one random number is needed and because it guarantees a better variance than totally random sampling and so it reduces the possibility of too fast convergence.

5) Line extraction: From the set of particles, two methods were implemented to extract the final line to use as an approximation of the real line. The first method consists to simply select the particle with the highest weight and so the one that overlaps more the street lane marker. The second method instead consists in creating a new particle where the x-coordinates of the points are the averages of the x-coordinates of all the points of the particle set as expressed in Equation (8).

$$p_{approx} = \{(avg(p_{i_{x_1}}), y_1), (avg(p_{i_{x_2}}), y_2), ..., (avg(p_{i_{x_{N_c}}}), y_{N_c})\} \quad (8)$$
$$0 <= i <= N_p$$

where avg makes the average of a list of number, N_p is the number of particles and $p_{i_{x_j}}$ indicates the x coordinate of the $j-$th point of the i-th particle. The average method works only if the algorithm is converging in a region otherwise it create a totally wrong line.

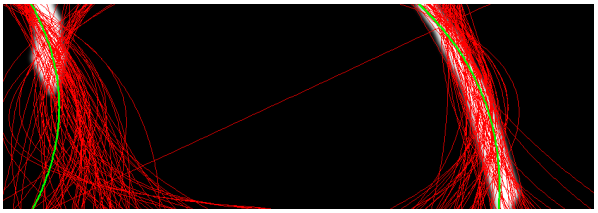
6) Reset mechanism: The particle filter implementation was extended including a module to reset the set of particles. If the particle that approximate the whole set has a weight lower than a certain threshold, 20% of the maximum weight, it means that the algorithm diverged. When the algorithm is not able to converge back to a good state after multiple instants, the set of particles is initialized from scratch using the initialization function. That's important when the lane markers are lost due to fast change of direction, obstacles or other unexpected situation of the street. This threshold mechanism permit also to set the state of the particle filter. If the weight of the best particle is above the threshold the filter is enabled, otherwise it is disabled and no approximation is provided.

C. Evaluation system

For comparing the execution of the algorithm on different datasets and with different configuration an evaluation system is implemented. It works by using the same principle of the weighting of the particle filter. The different is on the PDFs used to weight a particle. In the weighting, the PDFs were automatically created from the image processing module. For the evaluation, the PDFs were manually created using external computer graphics software and with adjustment made by a human person. The human correction guarantee that only the pixels that contain a street line are white and that everything else is black.

Using this system, particles can be evaluated numerically with the accuracy that in this report is considered as the percentage of points of a line that overlap a real lane maker in the image. If a line overlaps a lane marker, the accuracy of the particle will be the maximum, 1.0. The accuracy calculated with this method has validity inside this report for comparing different dataset and different configurations but it can't be used for comparing the algorithm with external solutions. In fact, the accuracy is evaluated in a simplified way, for example in some frames one of the two line disappear because of a bending but that is not took in consideration, so the ground truth for one of the line will be zero and so the accuracy will be wrongly zero.

Two particle filter are used in the algorithm, one work on the left part of the image trying to approximate the left lane marker, one the right marker. The final output of the algorithm is presented in Fig. 6 where the particles produced by the two filters are merged in a single image.



(a) Particle drawn above the probability density function. The green lines are the particle that have the higher weight.

(b) Best particles drawn above the image capture by the camera and processed using the IPM.

Fig. 6: Final output of the algorithm.

IV. Implementation

The programming language used to implement the system is Python 3. The choice was taken for the simplicity to integrate the final solution with the other module developed for the realization of the autonomous vehicle and the compatibility with the Raspberry where the system will run.

All the image processing operations were made using OpenCV 4.4.0 because it is very powerful and it permits to have access to a wide range of operations on images simplifying the creation of the system. The library Numpy was used to manage the operation on the PDF matrices. That choice was made because Python does not provide directly operation between matrix and so Numpy permitted to reduce the use of cycles in the implementation for obtaining a faster solution. For interpolating the points presented in the particle the library Scipy was used.

The code was structured using the object programming approach as represented in Fig. 7. This framework was used to simplify the debug, the readability of the code and the analysis of the system implemented. The negative point of this implementation is that it needs some extra loop that could be avoided using matrices representation and by removing some objects.

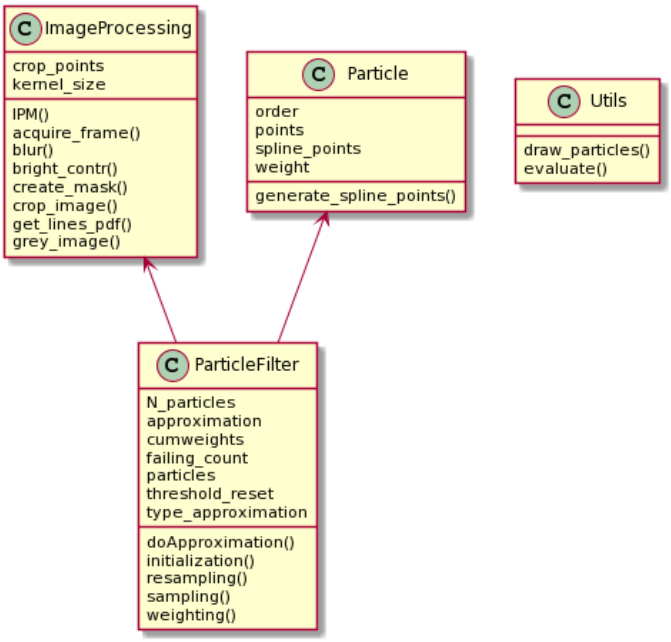
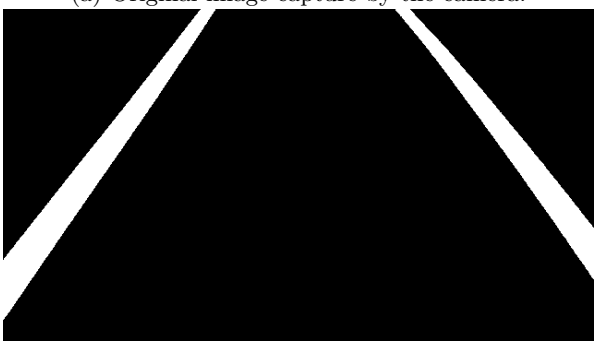


Fig. 7: UML of the software implemented. For every class, only the most important parameters and methods are reported. The Particle Filter is the main object that contains the filter. It contains inside the object ImageProcessing to load the data and a list of object Particle to represent all the particles. Utils contains some methods useful in various part of the code, for example, to display the result.

(a) Original image capture by the camera.



(b) Ground truth of the image 8a

Fig. 8: A frame from the dataset.

V. Results

In this section, the accuracy of a certain frame refer always to the average of the accuracy of the two lines extracted from the two filters.

A. Datasets

To test and validate the system, 4 datasets consisting of a sequence of images capture by the vehicle camera and the corresponding ground truth were realized, as shown in Fig. 8. They were created by moving the vehicle inside a close environment created for the scope. The resulted datasets are simpler than what a real car would encounter in the real world. Some simplifications are because the ground is composed of constant colour and the lane markers are well identifiable because made with white tape in contrast with the dark green of the ground. Moreover, very few obstacles were present along the path of the car. The aspect that increases the complexity of the dataset is the high reflectivity of the ground that made difficult for the algorithm to identify the lanes marker.

The datasets cover different cases:

- Dataset 1: 117 frames, the vehicle move on a straight street delined by a straight continuous line;
- Dataset 2: 394 frames, it's the longest dataset. The vehicle move on a straight line delimited by a segmented line. Moreover, a change of line and a backward movement was performed;
- Dataset 3: 154 frames, the vehicle bends and pass through a crossroad;
- Dataset 4: 84 frames, the vehicle pass through a crossroad and then it bends but the lane is very difficult to identify due to a very high level of the reflection of the ground.

One sample for each dataset is reported in Fig. 9.

(a) Dataset 1, straight lane.

(b) Dataset 2, dashed line.

(c) Dataset 3, bend.

(d) Dataset 4, reflection.

Fig. 9: Frames from the datasets.

The following results were obtained by running the algorithm on a laptop with Core i5-5257U (2x2.7 GHz) 4 GB RAM 1867 MHz.

B. Parameters tuning

In the tuning phase, two aspects were considered to compare different configurations: the execution time and the accuracy obtained. The accuracy has to be maximized to have a more correct solution and the time need to be minimized so the system can run in real-time. The parameter that can be tune in the particle filter are:

- *Number of particles* (N_p)*:* it represents the number of particles used in the filter. A high number of particle increase the accuracy but also increase the computation time;
- *Number of interpolation points (M):* it is the number of points that define the line and so the points that will be used in the weighting step of the filter to assign the weight to a particle;
- *Number of control points and spline order* (N_c *and order):* these parameter will influence the representation of the particle and so the accuracy for the detection and tracking;
- *Reinitialization steps:* it is the number of steps after which the particle filter reset itself because is not able to create particles with weights higher than the threshold;

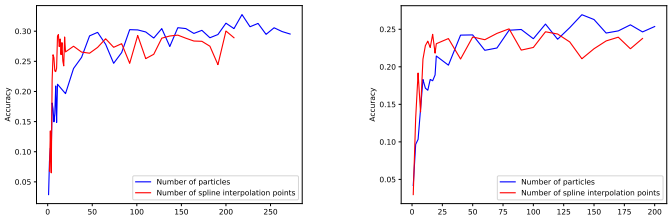
instead for the image pre-processing:

- *IPM usage:* it permit to remove the effect of the camera perspective to have the picture as it would be taken by above the street without distortion;
- *Blur kernel size, brightness and contrast:* these parameters are responsible in the creation of the PDF and they permit to create a system that can work with different light conditions.

For tuning mainly the first two dataset were used because of their size and the amount of different condition presented. The third one is introduced for tuning the the blur kernel size, brightness and contrast. The fourth one is not used for tuning.

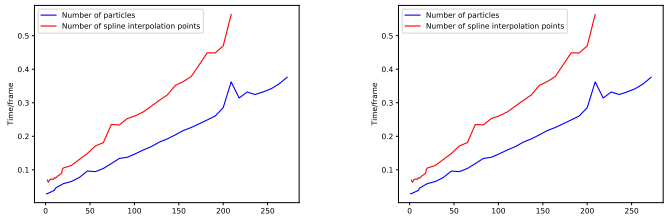
1) Number of particles: A different number of particles were used in the particle filter in the range 0 to 270 for the first dataset and 0 to 210 in the second dataset, in both cases with an increasing step of 10 particles. The resulting accuracy obtained are represented in Fig. 10 and the execution time in Fig. 11 with the blue lines. From the data is possible to realize that after a certain amount of particle, $N_p > 50$, the accuracy converge and so it's not useful to have too many particles. From the time analysis instead is possible to see that there is a linear dependency and increasing the number of particles increase the execution time. Considering the constraint of the real-time execution and the fact that the camera used work at 15 fps only when the execution time is lower than 0.071 s/frame can be considered for the execution on the vehicle.

2) Number of interpolation points: The number of interpolation points is important to have a correct weighting of the particles but at the same time because of the weighting algorithm, they cause a fast increase in computational time. Using the same approach for the number of particles, different number for interpolation points were used. In 10 is reported the accuracy obtained by changing the number of interpolation point and in Fig. 11 the resulted time, in both Fig.s the red line identify the interpolation points variable. From the accuracy result, it is possible to notice that convergence is reached by increasing of a small amount the number of interpolations points, so there is no need to have a big number of them but a value between 10 to 20 is enough.



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

Fig. 10: Accuracy N_p and M dependency.



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

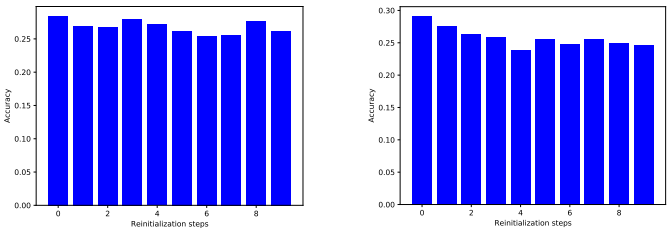
Fig. 11: Time N_p and M dependency.

The optimal parameters for having a real-time execution are reported in table I.

TABLE I: Dataset Overview.

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

3) Number of control points and spline order: These two parameter influence what the particle represents. A higher number of control points and a higher-order permit to represent more complex lines but at the same time make the algorithm more complex and more difficult to converge. The results using different configurations are reported in table II. The element of the table with *None* represent the configuration that can not be made because the order can not be greater or equal than the number of points. The best performance by considering both datasets are reached with three control points, instead, about the



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

Fig. 12: Average accuracy using different number of reinitializations steps.

order, it is difficult to choose one because the difference it is too small to say which one is better.

TABLE II: Accuracy using different value for the order and the number of control points.

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) Reinitialization steps: Number of reinitialization steps from 1 to 9 were used to see how it influence the average accuracy. A lower threshold is good in a very noisy situation because every time the algorithm diverges there is a new initialization of the particles. In fact, in the Fig. 12b the accuracy is better with a lower number of reinitialization step because the dataset 2 is complex and contains multiple points were the algorithm is not able to track the lane markers. At the same time, a too small number is not acceptable because the filter would reset himself too often creating low-performance with respect to the execution time. Moreover, a continuous reset would change the system from a lane tracking to a lane detection algorithm. So we decided to use four reinitialization steps for all datasets.

5) IPM usage: In table III the accuracy using IPM is presented. Comparing these results with the one in table II is possible to notice that the IPM does not influence so much the accuracy of the algorithm. That different from the expectation. In some frames, IPM helped to converge faster but on average its effect is not visible.

6) Blur kernel size, brightness and contrast: These three parameters about image processing are not directly related to the particle filter. They were analysed using dataset 1, 2 and 3 because it is interesting to see if they can generate good probability density functions for every situation. Dataset 4 was not used because it contains too much reflection and noisy that would affect too much the average accuracy making impossible to tune the parameters.

TABLE III: Accuracy using different values for the order and the number of control points using IPM.

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

The function used for applying the blur to the PDF matrix use kernels that are two-dimensional arrays of pixels. The *kernel_size* parameter refer to the size of the kernel matrix. Knowing that the lines that the vehicle should meet would be vertical the kernel used was not square but rectangular with the high of the kernel being a multiple of the width so $final\_kernel\_size = (kernel\_size * multiplier, kernel\_size)$ where *multiplier* is the variable to make the kernel not square. Three experiments were made: the first one using $multiplier = 3$, the second one using a $multplier = 7$ and the third one with $multiplier = 3$ but using a lower brightness and a higher contrast to remove noisy. The result of the three experiment with different *kernel_size* are reported in table IV.

TABLE IV: 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

Experiment 3 should give better results because the different value of contrast and brightness remove more noisy but actually, it is worst of the other two experiments. That happens because the more strict parameters remove the noisy and also part of the lane marker making more difficult for the algorithm to converge due to the lack of information. A frame of Experiment 2 is reported in Fig. 13a and a frame of Experiment 3 is reported in Fig. 13b. In the first case, there is noisy due to reflection, in the second case instead the noisy is removed together with the correct information of the lane markers. About the kernel matrices, it is possible to notice from table IV that the size does not influence significantly the accuracy.

C. Outcomes

The tuning phase permit to extract the follow parameters to obtain the best result from the algorithm:

(a) Frame with high noise.

(b) Frame filtered with loss of information about the lane marker.

Fig. 13: Different kind of brightness and contrast.

- $N_{s_{fast}}$*:* 50;
- M_{fast}*:* 17;
- $N_{s_{slow}}$*:* 100;
- M_{slow}*:* 25;

- N_c*:* 3;
- *Spline order:* 2;
- *IPM:* No;
- *Reinitialization steps:* 4;
- *Blur kernel size:* 21x7;
- *Brightness:* -60;
- *Contrast:* 100;

where the keyword $fast$ refer to the parameter to have the algorithm run in real-time and *slow* refer to the parameter that give the best accuracy but using more computational time and so it can not be run in real-time.

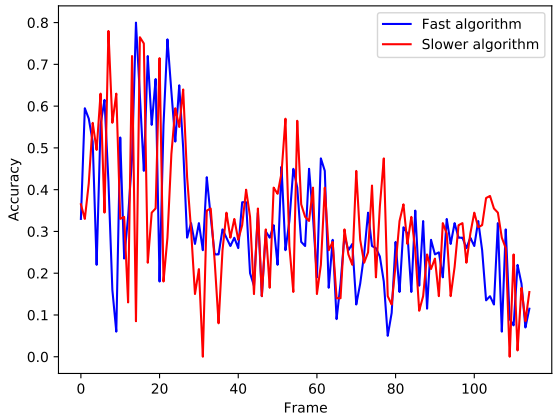
In Fig. 14 there are the graphs of the accuracy obtained by using the algorithm on all the four datasets. The real-time execution, as expected, performs worst in some case but overall the performance is not so inferior to the slower algorithm that uses more particle and more interpolation points. The behaviour of the accuracy reflects the dataset as expected. In 14a the accuracy is high at the beginning because there is a straight lane and then it decreases because of a bend. Similar behaviour is visible in Fig. 14b where the straight lane is present at the beginning and after the middle of the dataset. In Fig. 14d instead, the initial accuracy is very low because of the reflection of the street but then it increases because the lane markers become more distinguishable from the rest of the street.

VI. Conclusion

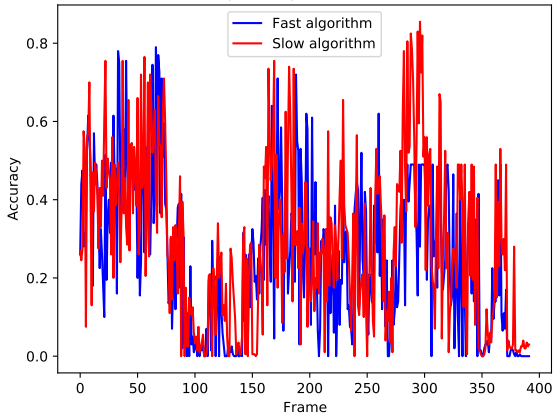
In this report, an algorithm for lane detection and tracking is proposed using a particle filter. The algorithm works and can find and track the lane markers in a simple environment like the one presented in [1]. In the real world, the algorithm would probably have problems to create a correct PDF due to different light conditions and colours of the pictures. Even some limitations, the algorithm can provide an estimation of the lane in real-time as was the objective of this project.

VII. Future improvements

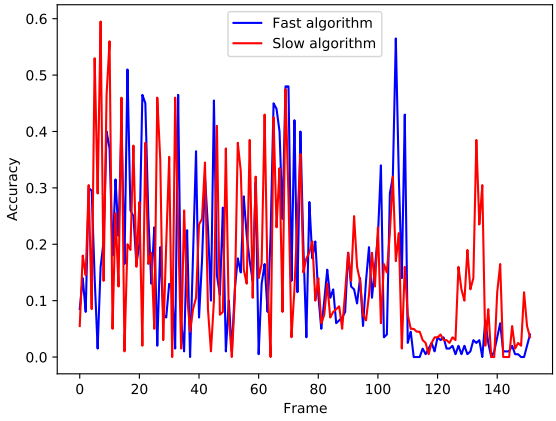
The project could be improved in different aspects. The sampling stage of the particle filter could be changed by including odometry information from the vehicle. In particular, the information about the turning of the steering wheel could be used to sample with less randomicity. A



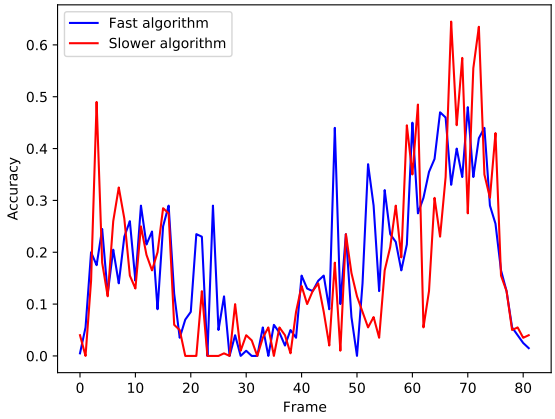
(a) Dataset 1.



(b) Dataset 2.



(c) Dataset 3.



(d) Dataset 4.

Fig. 14: Final accuracy frame by frame in all the datasets using the best parameters found.

simple model, for example, could be to sample more lines in the direction of the turning of the steering wheel.

Another improvement regards the implementation of the algorithm. The code was implemented by balancing velocity of the execution and capacity of editing and debugging but it can be improved by removing some object and using matrices to represent the particles and so creating a faster algorithm. Moreover, other languages faster than Python, as C could be used and a GPU for image processing.

Another direction that should be taken would be on the image pre-processing. In this paper, the blur, brightness, contrast and a fixed mask were used to extract the PDF from the images but the three parameters were chosen at the beginning of the execution. That's an optimal solution if the environment is without noise and in perfect light conditions. A better pre-processing of the image would not be fixed but the parameters would adapt to the pictures.

Finally, the scientific validity could be increase by running the algorithm on an external dataset realized in a real street with a more detailed ground truth as The KITTI Vision Benchmark Suite[17]. Using that dataset would permit to compare this solution to other research as in [18].

References

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

[2] 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.

[3] M. Samuel, M. Mohamad, S. mad saad, and M. Hussein, "Development of edge-based lane detection algorithm using image processing," *JOIV : International Journal on Informatics Visualization*, vol. 2, p. 19, 01 2018.

[4] W. Phueakjeen, N. Jindapetch, L. Kuburat, and N. Suvanvorn, "A study of the edge detection for road lane," in *The 8th Electrical Engineering/ Electronics, Computer, Telecommunications and Information Technology (ECTI) Association of Thailand - Conference 2011*, 2011, pp. 995–998.

[5] M. Nieto, J. Arróspide, and L. Salgado, "Road environment modeling using robust perspective analysis and recursive bayesian segmentation," *Mach. Vis. Appl.*, vol. 22, pp. 927–945, 11 2011.

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

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

[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] D.-J. Kang and M.-H. Jung, "Road lane segmentation using dynamic programming for active safety vehicles," *Pattern Recognition Letters*, vol. 24, no. 16, pp. 3177 – 3185, 2003. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0167865503001843

[10] 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.

[11] E. D. Dickmanns and B. D. Mysliwetz, "Recursive 3-d road and relative ego-state recognition," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 14, no. 2, pp. 199–213, 1992.

[12] J. Xiao, L. Luo, Y. Yao, W. Zou, and R. Klette, *Lane Detection Based on Road Module and Extended Kalman Filter*, 02 2018, pp. 382–395.

[13] N. Apostoloff and A. Zelinsky, "Robust vision based lane tracking using multiple cues and particle filtering," in *IEEE IV2003 Intelligent Vehicles Symposium. Proceedings (Cat. No.03TH8683)*, 2003, pp. 558–563.

[14] M. Bertozzi and A. Broggi, "GOLD: a parallel real-time stereo vision system for generic obstacle and lane detection," *IEEE Transactions on Image Processing*, vol. 7, no. 1, pp. 62–81, Jan. 1998, conference Name: IEEE Transactions on Image Processing.

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

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

[17] J. Fritsch, T. Kuehnl, and A. Geiger, "A new performance measure and evaluation benchmark for road detection algorithms," in *International Conference on Intelligent Transportation Systems (ITSC)*, 2013.

[18] A. Kumar and P. Simon, "Review of lane detection and tracking algorithms in advanced driver assistance system," *International Journal of Computer Science and Information Technology*, vol. 7, pp. 65–78, 08 2015.