A Scalable Lane Detection Algorithm on COTSs with OpenCL

Kai Huang*†, Biao Hu†, Jan Botsch†, Nikhil Madduri‡, and Alois Knoll†

∗School of Mobile Information Engineering, Sun Yat-Sen University
†Chair of Robotics and Embedded Systems, Technical University Munich, Germany
Email: ∗huangk36@mail.sysu.edu.cn ‡nikhil.madduri@gmail.com †{kai.huang,hub,jan.botsch,knoll}@in.tum.de

***Abstract*—Road lane detection are classical requirements for advanced driving assistant systems. With new computer technologies, lane detection algorithms can be exploited on COTS platforms. This paper investigates the use of OpenCL and develop a particle-filter based lane detection algorithm that can tune the trade-off between detection accuracy and speed. Our algorithm is tested on 14 video streams from different data-sets with different scenarios on different COTS hardware. With an average deviation fewer than 5 pixels, the average frame rates for the 14 videos can reach about 400 fps on both GPU and FPGA. The peak frame rates for certain videos on GPU can reach almost 1000 fps.**

I. Introduction

Lane detection is one of the most basic functions for Advanced Driver Assistance Systems(ADAS) and it has grabbed significant attention in research since mid-1980's. New development on computer technologies, however, has allowed new perspectives on designing a good lane detection algorithm. On the one hand, the powerful but low-cost commercial-off-the-shelf (COTS) semiconductor products boost the usage of COTS for domain-specific applications. In the automotive industry, it is very attractive to replace traditional dedicated ECU/ASIC-implementations with COTS in order to reduce the overall manufacture cost. On the other hand, as the software in a car is expected to run up to 1 GB of software [2], it is preferable to design a scalable algorithm to leave rooms for multiple ADAS applications on single ECU [3]. The scalability here means an algorithm can consider the trade-off between the accuracy and demanded computing power.

This paper presents a scalable lane detection and tracking algorithm based on Particle Filter [4] and OpenCL. The proposed method is vision-based and requires no knowledge of any physical parameters like position and orientation of the camera. Lane markings in a video stream can be either detected or tracked by the algorithm. For the case of tracking, Particle Filter is used. The choice of Particle Filter, rather than the most popularly used Kalman Filter is that Particle Filter is non-linear and particularly suitable for parallel processing, as each particle can be processed independent of the others. In addition, by changing the number of used particles, the algorithm can make a trade-off between the tracking accuracy and computing power.

The choice of OpenCL is because it is an industry standard for parallel computing supporting heterogeneous hardware. Hence source code written in OpenCL can be executed on GPUs, FPGAs, and Multi-Core CPUs alike, without major modifications. This helps in developing software that runs on heterogeneous COTS hardware for future automobiles, replacing the traditional dedicated ECUs. With the OpenCL implementation, our algorithm is tested on both GPUs and FPGAs. With an average deviation fewer than 5 pixels, the average frame rates can reach about 400 fps on Nvidia GeForce GTX 660 Ti and Altera Stratix V A7 for different kinds of road scenarios. On the resource-restricted Altera Cyclone V SoC FPGA, the average frame rates can still reach 25 fps.

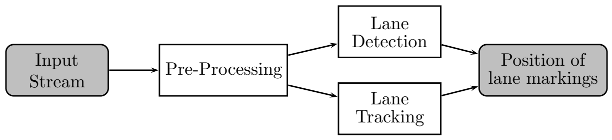


Fig. 1: Algorithm overview

Related Work: To capture the non-linear dynamics of lane tracking, Particle Filter is typically used. In [6], lane tracking is conducted by a statistical Hough Transform with Particle Filter. Stratified resampling is used to resample the particles based on their weights and the high weight particles after resampling are chosen as the detected lines. This algorithm was implemented in MATLAB on Intel Core 2 Duo (2.2 GHz) machine and the throughput of the algorithm is as low as 1.0 Hz. There are also learning approaches, e.g., [5] uses around 200 particles to track the lane markings with a separate particle filter dedicated to each of the lane markings. The lane tracking algorithm in their case was reported to be working on 240x320 images at 25 Hz on a 4 GHz processor. While the major advantage of a particle filter being its ability to capture non-linearity and Kalman Filter can provide optimal solution if a part of the solution is linear, [7] provides a very innovative approach to combine both, where, the lane tracking problem is split into two separate sub-problems. The states that are linear are processed by Kalman filter while Particle Filter estimates the non-linear states. This algorithm was tested on Intel Atom CPU N270 (1.6 GHz) and the performance obtained is up to 30 Hz. In [1], an FPGA implementation is presented where the frame rate can reach 40 fps for 752x320 images.

II. Method and Implementation

This section presents in details our algorithm. The algorithm contains three parts, namely video pre-processing, lane detection, and land tracking, as shown in Fig. 1. The video stream showing a road and the area surrounding it will be processed in two subsequent steps frame by frame. First, information on the lane markings is amplified and extracted from each frame in the pre-processing step. Then, depending on whether previous estimates of the position exist or not, the exact position of the lane markings is detected or tracked in a lane detection or lane tracking steps, respectively.

A. Pre-processing

The pre-processing is the first step for both detecting or tracking street lanes. A region of interest(ROI) is selected from an incoming image. This region is pre-processed to provide the required information on the lane markings. First the image is transformed into grayscale. Then a Sobel filter detects edges in the image and finally a thresholding step removes minor disturbances and strengthens the appearance of the lane markings.

The ROI contains all essential information for the subsequent steps and the rest of the image can be discarded. The size of the ROI is a parameter that determines the efforts for later-on lane detection/tracking. The smaller the ROI chosen, the less computing power is needed. In this work, the size and position of the ROI are kept adjustable, so that the algorithm can be tested in a wide range of scenarios. Once the ROI is selected, only the area within the region is processed in the subsequent steps. The ROI is then transformed to a grayscale format, where each pixel reflects the intensity of the pixel in the original image. In principle, dark pixel will receive lower intensity values and bright pixels will receive higher values. The grayscale is performed for each pixel of the ROI individually. In this manner, lane markings are substantially brighter than the road they are printed on.

Afterward, a Sobel filter is applied to the grayscaled image to produce a new image, where only transitions and edges (such as the lane markings) of the original image are present. Technically, each pixel in the new image describes the gradient of the original image at that position. A strong gradient will be represented by a high absolute value and a soft gradient by a value close to zero. With the outcome from the Sobel filter, there may be still noises from e.g., varying colors of the street material and shadows. Therefore, a thresholding step is introduced to set the intensity of those pixels, whose gradient falls below a certain threshold, to zero, otherwise maximum. This step will make the lane detection possible under difficult conditions, e.g., when rain or fog lead to blurred images. In those situations the edges of the lane markings might be less visible and the thresholding will make them brighter.

The pre-processing displays a high potential for parallel computations. Grayscaling and Thresholding can be applied to each pixel independently. The use of the Sobel Filter requires knowledge of eight neighbouring pixels. This provided, all pixels in the ROI can be pre-processed in parallel. Therefore, an OpenCL kernel was developed such that the pre-processing can be performed on FPGAs or GPUs.

B. Lane Detection

After the pre-processing, the lane markings are indicated by a band of pixels with high intensities in the ROI. It is the task of the lane detection or lane tracking algorithm to extract the exact positions of the lane markings. Lane detection is performed whenever no estimates on the lane markings are available, for example the very first frame that is processed. After detecting the lane markings, detection is only performed when the lane tracking algorithm fails to track the markings.

1) Lane Marking Representation: To represent the lane markings, i.e., the band of pixels with high intensities in the ROI, we assume all lane markings within the ROI are straight lines. This is due to the fact that the ROI captures only a small section of the street ahead and, on straight roads and even in moderate bends, the straight-line-assumption always holds. In sharp bends or other exceptionally routed roads where the lane markings might exhibit a bend, the ROI can then be split horizontally into two or more regions, yielding sub-regions with straight lane markings.

With the straight-line assumption, a lane marking can be defined by two points, one at the top and the other at the bottom of the ROI. The position (or state) of a line can thus be expressed as $X = \binom{x_{top}}{x_{bottom}}$, as the height of the ROI is known and the y-values of these two points are constant and given by the respective line number in the ROI. The slope s_X of the line and any other point on the line with y-value i can be determined by $s_X = \frac{x_{bottom} - x_{top}}{\text{ROI\_HEIGHT}}$, and $x_i = x_{top} + s_X \cdot i$. In this manner, one can access the intensity value of any pixels for a given line X.

2) Detection: To detect multiple lane markings, the ROI is vertically split into multiple regions, each for one lane marking. To detect one lane marking, a number of randomly placed candidate lines are populated within each region. The actual detection is achieved by assigning a *weight* to the candidate lines, which expresses how close the line is placed to a real lane marking. The weight of a line is determined by summing up the intensities of all pixels in the line within the ROI. Further, the pixels in an adjustable neighborhood around a line (left and right of the actual line) are added to this weight.

Keeping in mind that lane markings are represented by pixels with high intensities, this method results in candidate lines with higher weights, if they are close to a lane marking. The candidate line with the highest weight is selected to represent the lane marking. The number of candidate lines that are sampled in a region is adjustable and, therefore, there is a trade-off between the performance of the lane detection and its quality. Using more candidate lines will result in a better detection, but also consumes more time.

The candidate lines are created by sampling its x_{top}- and x_{bottom}-values following a normal distribution. The mean μ of a normal distribution represents the value we expect to appear (the expectation). In the case of lane detection the most likely position of a lane marking is the center of a region. This is the position of a lane marking, if a vehicle drives on a straight road. The standard deviation σ decides how the candidates are distributed around the expectation. In this work, σ is set to half of the region width. From the definition of the standard deviation of a normal distribution follows that statistically about 68% of the sampled lines will be placed completely within a region and the remaining 32% might overlap to other regions. This empirical choice of parameters proved to detect lane markings in all kinds of positions from our later experiments.

3) Implementation: The presented method for detecting lane markings has the desirable characteristic that the candidate lines are independent from one another. The sampling of the lines and the calculation of their weights can be done in parallel. Hence, an OpenCL kernel is used to compute the weight of each sample line. Notes that the selection of the best lines from the candidate lines requires knowledge of all lines and is therefore performed on the host.

The lane tracking algorithm, which will be described in the following section, requires a set of possible candidates/-particles for each lane marking. Therefore not only the best candidate is stored, but for each lane marking a subset of the candidate lines is kept. In the following this subset will be referred to as the good lines. The line with the highest weight amongst the good lines will be called the best line and represents the actual lane marking.

C. Lane Tracking

Lane tracking differs from lane detection as it uses information from a previous frame to detect the lane markings in a subsequent frame. Hence it does not actually detect the lines, but rather tracks them. The lane tracking algorithm has two sources of information at its disposal, the pre-processed ROI and the set of good lines and best lines from the previous frame. A Particle Filter is employed to keep track of the markings. Each lane marking is tracked separately, which means that multiple instances of the same particle filter are used in the algorithm.

1) Particle Filter setup: We match the classical Particle Filter $P(X|Y) = \frac{P(Y|X)P(X)}{Pr(Y)}$ as follows. The state variable is defined as the position of a lane marking: $X = (x_{top}, x_{bottom})$. The prior distribution $P(X)$ is derived from the good lines from the previous frame. Similarly, observations Y is the actual lane marking positions. Though direct observations cannot be obtained, the best lines from the previous frame can be interpreted as observations of the lane markings in the current frame, assuming that the positions of the markings do not change significantly between two consecutive frames. Following the aforementioned matching, below presents the three steps to implement a Particle Filter.

2) Prediction update: This step is introduced, because the particles (good lines) are representing the lane markings in one frame, but are used as prior distribution in the next frame. In the new frame, the lane markings might have moved slightly because of the movement of vehicle. The particles need to move the same distance in order to be a valid prior distribution in the new frame. As the distance the lane markings shift is not known, the particles are shifted by a random value from a normal distribution with mean $\mu = 0$ and standard deviation $\sigma > 0$. $\mu = 0$ indicates that we expect no shift in an optimal case and $\sigma > 0$ accounts for a deviation from the optimum. In our work, σ is empirically set to $1/16$ ROI_WIDTH.

3) Important weight update: For each particle of the prior distribution the importance weight is calculated. The importance weight of a particle is determined by the numerator: $P(Y|X_i)P(Xi)$. It assesses the likelihood that the predicted particle X_i produces the observation Y. In this work this is determined by fitting the state of the predicted particle to the Gaussian function $w^i_{X_t} = \frac{1}{\sqrt{2\pi\sigma^2}} \mathrm{e}^{-\frac{1}{2}(\frac{X_t - \mu_f}{\sigma_f})^2}$, where $\mu_f = Y$. The standard deviation σ_f represents the measurement noise that accounts for a possible error in the assumption that the position of a lane marking does not change between two frames. The term $X_i - \mu_f = Xi - Y$ represents the distance of two lines, the particle X_i and the observation Y. This distance is calculated by summing up the point-wise distance of line X_i and line Y.

4) Resampling and Picking up best lines: The prediction and importance weight update produce a new set of good lines, where each particle has a normalized importance weight. Finally, a resampling step is performed in order to increase the accuracy of the tracking and prevent a degeneration of the set. The resampling step selects particles from the updated set according to their normalized importance weight and shifts them to a new set with the same number of particles. Since particles are selected according to their importance weight, good particles are more likely to be selected than less accurate particles. A particle can also be chosen multiple times.

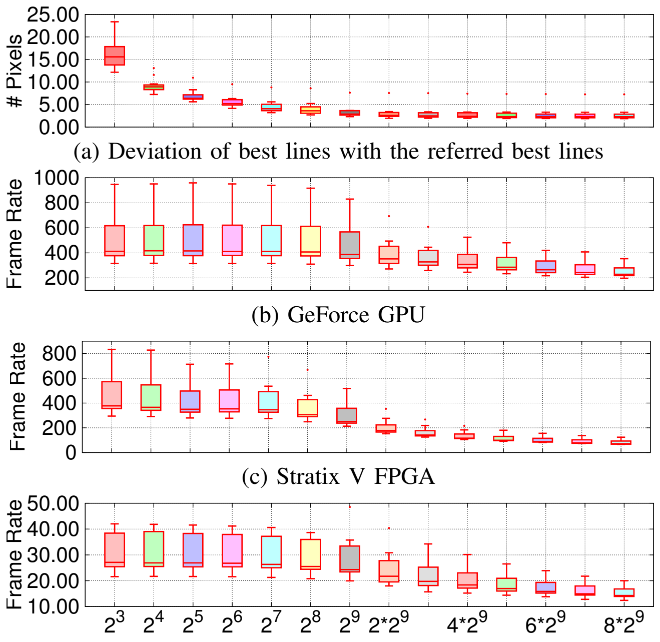
The resampling is used to keep the good lines that are close to the best lines of previous frame. The closeness is represented by the importance weight. However, picking up the best lines of current frame does not rely on the importance weight because the best lines may have some deviations. The collected intensity of a resampled good line still represents the lane marking. Hence, the resampled good lines with the highest intensity weight are selected to be best lines. The selected best lines of current frame will be used as a reference to resample good lines of next frame.

5) Implementation: The prediction, importance weight update, and intensity collection of one particle is not dependent on other particles. Therefore these parts can be performed in parallel and an OpenCL kernel was created that carries out the updates on GPUs or FPGAs. The resampling and picking up best lines, in contrast, are dependent on information from all particles and are implemented on the host. Note the number of particles used for the tracking does not have to be same as the number of sampling candidates used for the detection. The number of sampling candidates is the upper bound for the number of particles. In principle, larger number leads to higher accuracy. Nevertheless, both numbers can be used to tune the trade-off between speed and accuracy of our algorithm.

D. Re-detection Criteria

The switch from lane tracking to detection depends on the actual road scenarios. During lane tracking, a few criteria are checked whether the detected lane marking positions are reasonable and in line with the physical properties of lane markings. If the criteria are not met, a detection step is triggered to re-discover the positions of the lane markings again. The used criteria are a) lane markings do not cross, b) minimum distance between any detected lane markings (In this work, 20% of the width of the ROI), and c) at least 30% of a lane marking are in the ROI.

The last check is important for the lane tracking. The algorithm tracks lane markings based on existing estimates and



(a) Deviation of best lines with the referred best lines

(b) GeForce GPU

(c) Stratix V FPGA

(d) Cyclone V FPGA

Fig. 2: Accuracy v.s. frame rate. The X axis is # particles N_p. For $N_p \leq 2^9$, N_c is always 2^9, otherwise $N_c = 2^{12}$.

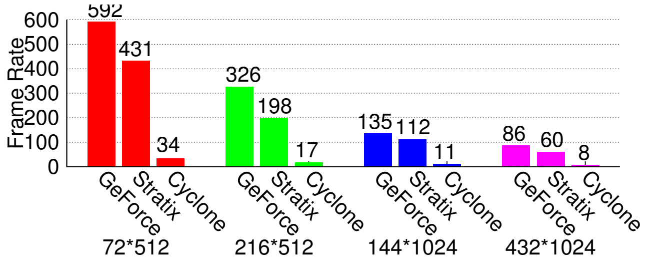


Fig. 3: Frame rate w.r.t. ROI.

it will do so even if the lane markings moves to the boundary or even out of the ROI. No new lane markings are detected in the lane tracking stage. In many scenarios, for example when a car changes the lane on the highway, one lane marking moves out of the ROI and another one moves in. The third check of the re-detection criteria discovers these cases and triggers a lane detection step to discover a new lane marking.

III. Experiments

Accuracy: First of all, we evaluate the impact of numbers of candidate lines(denoted as N_c) and particles (denoted as N_p) on the overall accuracy. Since our algorithm is a stochastic approach, the lane markings detected from a $N_c = 2^{14}$ and $N_p = 2^{12}$ setting are used as baselines. We evaluate different N_p and N_c combinations. Each combination is tested with 14 independent videos, including the Caltech Lanes Dataset and some self-recorded videos. The numbers of pixels deviated from the baselines are reported. The results are shown in Fig. 2a. From the figure, the first observation is that the accuracy increases when N_p increases. With $N_p = 256$, the maximum deviation is already below 5 pixels. Considering the neighborhood N_n is set to 10 pixels, i.e., a lane marking is represented by 20 pixels, we can conclude that our algorithm can achieve high accuracy. The second observation is that when $N_p \geq 512$ the deviation is saturated to 3 pixels. Larger N_p is needed when the ROI is set larger.

Performance: For all aforementioned N_p and N_c combinations, the 14 videos are tested on three COTS devices and Fig. 2b–2d report the corresponding frame rates. It can be seen that, the GPU can obtain an average 400 fps when $N_p \leq 512$ and the peak frame rates for certain videos can reach almost 1000 fps. For the Stratix V FPGA, the frame rates are lower than the GPU but still considerably fast, over 300 fps for $N_p \leq 256$. One the other hand, our algorithm can still reach real time on the resource-restricted Cyclone V SoC FPGA. Considering the power consumption for the Stratix V and Cyclone V are just 24 W and 4 W, respectively, we can conclude that our algorithm is super energy efficient.

In addition, we also investigate the influence of ROI size on the frame rate. Four different ROI sizes are tested and the results are shown in Fig. 3. In these cases, $N_p = 256$ and $N_c = 512$ are adopted. From the figure, It can be seen that the ROI size has big influence on the frame rate. Nevertheless, our algorithm can still reach 60 fps on Stratix V FPGA for considerably large ROI.

IV. Conclusion

This paper presents a vision-based lane detection algorithm. It requires no knowledge of any physical parameters like position and orientation of the camera and is hence very flexible. The algorithm is tested with 14 videos on three different kinds of COTS and demonstrate super fast speed, high accuracy, and super energy-efficient. The algorithm is so efficient such that traditional techniques like correcting the perspective with inverse perspective mapping, are simply unnecessary. Based on this work, we suggest that ADAS can be benefited from COTS and OpenCL can be used for ADAS application developments.

References

[1] X. An, E. Shang, J. Song, J. Li, and H. He. Real-time lane departure warning system based on a single FPGA. *EURASIP Journal on Image and Video Processing*, 2013(1), 2013.

[2] M. Broy, I. Kruger, A. Pretschner, and C. Salzmann. Engineering automotive software. *Proceedings of the IEEE*, 95(2):356–373, Feb. 2007.

[3] M. Buechel, J. Frtunikj, K. Becker, S. Sommer, C. Buckl, M. Armbruster, A. Marek, A. Zirkler, C. Klein, and A. Knoll. An automated electric vehicle prototype showing new trends in automotive architectures. In *International Conference on Intelligent Transportation Systems(ITSC)*, September 2015.

[4] J. V. Candy. *Bayesian Signal Processing: Classical, Modern and Particle Filtering Methods.* Wiley-Interscience, 2009.

[5] R. Gopalan, T. Hong, M. Shneier, and R. Chellappa. A Learning Approach Towards Detection and Tracking of LaneMarkings. In *IEEE Transactions on Intelligent Transportation Systems*, volume 13, pages 1088–1098, 2012.

[6] G. Liu, F. Worgotter, and I. Markelic. Combining statistical hough transform and particle filter for robust lane detection and tracking. In *IEEE Intelligent Vehicles Symposium*, pages 993–997, 2010.

[7] M. Nieto, A. Corts, O. Otaegui, J. Arrspide, and L. Salgado. Real-time lane tracking using rao-blackwellized particle filter. *Journal of Real-Time Image Processing*, pages 1–13, 2012.