TUM

TECHNISCHE UNIVERSITÄT MÜNCHEN
INSTITUT FÜR INFORMATIK

ADAS on COTS with OpenCL: A Case Study with Lane Detection

Kai Huang, Biao Hu, Long Chen, Alois Knoll, Zhihua Wang

TUM-I1637

ADAS on COTS with OpenCL: A Case Study with Lane Detection

Kai Huang, Biao Hu, Long Chen, Alois Knoll, Zhihua Wang

Abstract—The concept of autonomous cars is driving a boost for car electronics and the size of automotive semiconductor market is foreseen to double by 2025. How to benefit from this boost is an interesting question. This article presents a case study to test the feasibility of using OpenCL as the programming language and COTS components as the underlying platforms for ADAS development. For representative ADAS applications, a scalable lane detection is developed that can tune the trade-off between detection accuracy and speed. Our OpenCL implementation is tested on 14 video streams from different data-sets with different road scenarios on 5 COTS platforms. We demonstrate that the COTS platforms can provide more than sufficient computing power for lane detection in the meanwhile our OpenCL implementation can exploit the massive parallelism provided by the COTS platforms.

Index Terms—Advanced Driver Assistance Systems (ADAS), Commercial of the shelf (COTS), OpenCL, FPGA, GPU

1 INTRODUCTION

AUTOMOTIVE IC sales had been estimated to represent 7.3% of the total $287.1 billion IC market in 2015 [22]. The IC Market Drivers Report states that from 2014 through 2019, the automotive IC market is forecast to increase at an average annual growth rate of 6.7%, highest among six major end-use applications – computer, consumer, communications, automotive, industrial/medical, and government/military – and two points more than the 4.3% CAGR forecast for the total IC industry over the same time period. More specific, the automotive IC market is forecast to grow 7% to $22.2 billion in 2016 and continues increasing to $29.2 billion in 2019 [22]. The driving force of this boost in automotive electronics is the concept of autonomous cars. According to the consensus of most Tier-1 OEMs, fully autonomous cars will present to market by 2018-2020 [27]. Their arrival will help double by 2025 the size of todays automotive semiconductor market. In additional, according to Linley Gwennap, president of the Linley Group, an expected $10,000 premium for a self-driving car will be paid for lots of cameras and processing horsepower. The question here is how Tier- 2 and 3 suppliers, particular for car electronics suppliers, to benefit from this boost.

Freescale, one of the leading car electronics suppliers, states two critical issues on the path toward autonomous driving: the lack of open standards for Advanced Driver Assistance Systems (ADAS) development and safety guarantee of consumer-focused silicon chips for safety-critical autonomous applications [16].

Responding to the current lack of open standards and to reverse the trend toward closed, proprietary ADAS systems which inhibit development and design innovation, Freescale has announced it will soon introduce an Open Computing Language (OpenCL)-based automotive development environment, targeting its own silicon. The motivation of Freescale' move is to reduce ADAS R&D overhead and free their customers to focus more on ADAS innovation, and more importantly, ADAS safety, according to Bob Conrad, Freescale SVP and General Manager, Automotive MCUs [16]. OpenCL [20] is an open, royalty-free standard maintained by the non-profit technology consortium Khronos Group. As OpenCL is designed for cross-platform, parallel programming, and improving speed and responsiveness for a wide spectrum of applications in numerous markets, we consider as well OpenCL could be a suitable programming model for ADAS development.

Regarding the underlining platforms that execute ADAS applications, although Freescale claims that consumer-oriented silicon solutions designed to enhance gaming graphics or run smartphone apps are not safe enough to ensure autonomous driving-quality and reliability, we consider the mass-production commercial off-the-shelf (COTS) components will be the de-facto carriers for ADAS. The reasons are multi-fold. Firstly, the time-to-market and development cost can be significantly reduced by using COTS, compared to traditional dedicated ECU/ASIC-implementations. Secondly, autonomous driving requires significantly higher computing power than those available via special-purpose controllers equipped with safety features. Nowadays COTS platforms, on the other hand, can provide massive computing power and the performance per price can be even a magnitude higher than automotive-specialized ones [18]. Third, COTS platforms start to support OpenCL for heterogeneous computing, which allows ADAS developers to exploit the computing power of COTS in the meanwhile trades off different design objectives, e.g., performance and energy consumption. Therefore, the use of high-performance commodity COTS in safety-critical systems becomes desirable. The safety issue that Freescale believes to hinder consumer-oriented silicon solutions for ADAS can and will be solved by the technology advance. A short survey on COTSs for safety-critical systems is presented in Section 2.1 to justify this claim.

To test the feasibility of using OpenCL for ADAS development on COTS platforms, this article conducts a case study. We choose five different COTS platforms which are not specifically designed for automotive markets, i.e., Nvidia GeForce GTX 660 Ti and Quadro K600, Altera Stratix V A7 and a resource-restricted Cyclone V SoC FPGA,

and Redmi Note 2 mobile phone. All these five platforms support OpenCL such that source code written in OpenCL can be executed on them without major modifications. This helps for a fair comparison. In addition, Nvidia Jetson TK1, which is specially designed and targeted for automotive market, is used for comparison as well. With this case study, we try to answer following questions: 1) Can OpenCL be a standard for cross-platform development and applications developed with OpenCL be smoothly applied on different COTS platforms? 2) Can an ADAS application, which is developed using OpenCL, exploit the parallelism provided by COTS platforms? 3) Can COTS platforms provide enough computing power for ADAS applications? For the safety issue, it is not a focus of this article and we left for the related work for a brief discussion.

For ADAS applications, lane detection is chosen. Lane detection is one of the most basic functions for 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. For example, as the software in a car is expected to run up to 1 GB of software [7], it is preferable to design a scalable algorithm to leave rooms for multiple ADAS applications on single ECU [8]. The scalability here means the demanding power of the algorithm can be tuned to trade off between the accuracy and speed. The reason is that lane detection, at certain circumstance, can be particularly computationally demanding due to, e.g., varying light conditions, traffic on the road that obstructs the lane markings, and the shadows cast by buildings or trees. Therefore, such scalability has practical use in real life. On the other hand, with this capability to scale the computing demand, we can use this algorithm to exploit the limits of the tested COTSs. The contributions of this article are summarized as follows:

- A scalable lane detection approach is developed that can tune the trade-off between the detection accuracy and computing power. Lane markings from live road stream can be either detected or tracked by our approach. For the case of tracking, Particle Filter [9] 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. By changing the number of the used particles, the algorithm can make a trade-off between the tracking accuracy and computing power (aka frame rate).
- With an OpenCL implementation, our lane detection is tested on five different COTS platforms with 14 video streams from different data-sets representing different road scenarios. We demonstrate that our OpenCL implementation can exploit the massive parallelism provided by the COTS platforms. With an average deviation fewer than 5 pixels, the average frame rates can reach about 600 fps on Nvidia GeForce GTX 660 Ti and Altera Stratix V A7 for the 14 test videos. The peak frame rates for certain videos on the GeForce GPU can reach over 1000 fps. On the low-budgeted Quadro K600, the average frame rates can be more than 200 fps. Even on the resource-restricted Altera Cyclone V SoC FPGA and Redmi Note 2 mobile, the average frame rates can still reach real time at acceptable accuracy.
- We profiled and analyzed the execution of our application and show the detailed timing of the execution on the 5 COTSs platforms. In addition, we extended existing OpenCL runtime environment to support Altera FPGA and demonstrate the heterogeneous execution of our lane detection application with a combination of Nvidia GeForce GPU and Altera Stratix FPGA.
- To compare the performance between COTSs and automotive-specific hardware, Nvidia TK1 is chosen, which is specially designed and targeted to automotive market for ADAS applications. Since OpenCL is not supported by TK1 for the moment, we re-implemented our lane detection with CUDA and ran all the test cases with the CUDA implementation on TK1. By comparing the measurement results, we demonstrated the usefulness of COTSs platforms for ADAS applications.

The rest of this paper is organized as follows: The next section reviews related work in the literature. Section 3 provides a short introduction of OpenCL. The details of our algorithm is presented in Section 4. Section 5 describes the experimental setup and Section 6 presents experimental results. Section 7 concludes the paper.

2 Related Work

2.1 COTS on safety-critical systems

Early discussion on building systems with COTSs can be dated back to [6], where, from a software engineering perspective, some basic understanding of how developing systems with COTS products were described and why and what new capabilities were being identified. From the perspective of hardware platforms, Kotaba et al. [26] systematically analyzed the effects of COTS multi-core platforms – the applications compete for resource access, typically arbitrated in a non-explicit manner by specific hardware implementation, which causes non-deterministic temporal delays on execution – and suggested mitigation techniques, where possible.

Inline with the aforementioned guideline, there is generic work on providing temporal isolation on COTS platforms without specifying concrete application domains. In [34], a run-time fixed-size weighted DMA transaction was suggested to provide temporal separation for hardware-based I/O virtualization on PCIe. With such virtualization, secure and safe sharing of I/O subsystems can be guaranteed for COTS multi-core systems. In [37], an integrated approach is presented for temporal partitioning and WCET analysis of COTS multi-core systems. Workload monitoring at basic block level at runtime is used to ensure that the respective process does not introduce further interference on the affected resource. In [18], a so-called software coded processing technique, i.e., detecting transient, permanent, and systematic hardware execution errors by arithmetic encoding of variables, constants, and operations, is deployed to demonstrate that COTS hardware can be used in safety-critical applications.

There is also work targeting specific safety-critical domain, e.g., automotive, railway, and aerospace. Shen

et al. [45] demonstrated that a teleoperation system can be exclusively composed of low-cost COTS components yet still meet the high performance demands of remotely driving a car on the road with wireless communication over 3G/4G data networks. In [15], an attempt to design an automotive ECU using a Xilinx Zynq-7000 FPGA, rather than an MCU-based platform, which conforms to both the AUTOSAR and ISO 26262 standards, was presented. Cohen et al. [12] showed the feasibility of hard real-time and parallel execution of safety-critical tasks on the Xilinx Zynq COTS. A train signaling usecase provided by Alstom Transport was programmed in an extension of the synchronous dataflow language HEPTAGON and a software stack and composition methodology were designed to enable hard real-time control code to be isolated from timing interference. Further, a recent book [24] examines radiation effects for FPGA and GPU on aerospace applications. The authors concluded that even FPGA and GPU are very sensitive to radiation, such COTS platforms can still meet the reliability requirements with the help of certain fault-tolerance techniques.

There are also quite a few recent research projects working on deterministic multi-core architectures, e.g., MERASA [51], parMERASA [50], ACROSS [43], RECOM-P [42] and T-CREST [47]. The MERASA architecture uses either round robin [40] or IABA arbitration [39] to guarantee an upper bounded for inter-core interference. The parMERASA approach presents time predictable on-chip ring architectures to achieve composable WCET on a many core architecture [38]. The RECOMP project uses IDAMC NoC [33] to support mixed critical applications. The ACROSS project employs time triggered NoC for time predictable access to the shared memory. Similar to ACROSS, the T-CREST project aims to build a time predictable NoC architecture [48, 46].

From semiconductor industry, Freescales latest Qorivva MPC5643L was the industrys first multi-core to achieve a formal ISO 26262 certificate for ASIL-D functional safety capability by an independent third-party accredited certification body. Infineon in 2012 introduced a 32-bit TriCore^{TM} multi-core processor to meet safety and powertrain requirements of automotive industry. In November 2015, Altera announced a lockstep solution for its Nios® II embedded processors to enable the implementation of SIL-3 safety designs in Altera FPGAs in full compliance with functional safety standards IEC 61508 and ISO 26262 [49]. Nvidia and Huawei are also new comers of automotive industry that announced entering this market [35, 13] and we expect soon the announcement of their safety-critical solutions.

From all these R&D activities, We believe that the concern of Freescale on the safety issue of consumer-oriented silicon solutions for ADAS can and will be solved by the technology advance. COTS platforms will be the de-facto carriers for ADAS applications and thus we focus on the performance and programming issues in this article.

2.2 Lane detection

Numbers of researches have already been done in developing lane detection systems for the past two decades. Vision based sensing is among the most popularly used techniques. For vision sensing, one of the most commonly used methods for lane detection is Hough Transform [21] to detect lane markings [3, 5]. Hough Transform suffers from its own problems like grid dimension, quantization errors, and noise [41], to name a few.

To overcome the problems of Hough Transform, different alternatives have been proposed, e.g., a randomized line detection [10] that picks up three random edge points and finds the best candidate, color based segmentation methods that use color information in the image and extract the lines [11, 30], and steerable filter [31]. These methods are designed for a sequential implementation on a CPU and not quite suitable for parallel implementation. Compared to the aforementioned methods, our method combines particle based and gradient based approaches for detecting the lines. Therefore, the accumulator array as in the case of HT need not be maintained, which can reduce the memory transfers between GPU/FPGA and the Host CPU.

For lane tracking, the most frequently used techniques are the Kalman and Particle filters. In [29], a Kalman Filter is used to track the lanes after the detection. An ego vehicle lateral offset is predicted and the lane lines are selected based on this predicted value. Once the lines are selected, the ego vehicle lateral offset is updated. Though the performance of their algorithm was reported to be providing up to 94.34 Hz on a PC (based on AMD Athlon 64 3000+), there are several strong assumptions made about a constant ego vehicle lateral offset, lane width, and the width of lane markings. This raises questions about the robustness of the algorithm on different situations. In [31], lane tracking is conducted by combined using Kalman Filter updated by Hough transform. The throughput that was reported for their system is around 30 Hz. Kalman Filter provides an optimal solution as long as the probability distribution of the states to be estimated are linear and the noise is Gaussian but it fails when the problem is non-linear. In this regard, there is a modified version of the Kalman Filter known as Extended Kalman Filter used in [32] and [52] especially for lane tracking problem.

To capture the non-linear dynamics of lane tracking, Particle Filter is typically used. In [28], 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. However, the throughput of lane tracking algorithm is as low as 1.0 Hz. There are also learning approaches, e.g., [19] 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, [36] 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 [4], an FPGA implementation is presented where the frame rate can reach 40 fps for 752x320 images.

In our approach, we use Particle Filter to perform lane tracking after the lane detection step. While we follow a weak model, i.e., no assumptions about the lane width and lane marking size, that makes our algorithm quite robust in different road conditions, our main aim is to implement the application with OpenCL to exploit the parallelism of COTS platforms. We will demonstrate, giving sufficient computing power, an algorithm even with weak model can still achieve both high accuracy and speed.

3 OpenCL Overview

OpenCL (Open Computing Language) [20] is a standard for heterogeneous high performance computing managed by the non-profit technology consortium Khronos Group. It defines a high level abstraction layer for low level hardware instructions. This enables cross-platform execution from general purpose processors to massively parallel devices without changing the source code.

3.1 OpenCL framework

The OpenCL specification provides specific view and rules to organize the execution of applications. An OpenCL application runs on a *host* system which is connected to one or more accelerators *devices*. The host coordinates the execution and the devices are capable of executing C-like OpenCL code as accelerators. The computing acceleration by OpenCL is achieved by the means of the following four models in OpenCL framework.

Platform Model OpenCL abstracts the combination of host and devices as a single *platform*. Any platform has only one host and one or more than one devices. The host is often a CPU of a system. The device can be any accelerators, such as GPU and FPGA. A device is considered to be made of blocks of several *compute units* and each compute unit is made up of several processing element. According to this model, a processing element is the smallest entity and it resembles a *thread* in traditional parallel programming terminology.

Execution Model There are two kinds of executions of OpenCL, i.e., *host execution* and *device execution*. The host execution is mainly responsible for assigning jobs to devices, and making use of results from devices. The device execution is called the *kernel* that is called by host and executed in parallel in device. The kernels are mostly written in the *OpenCL C* programming language, a dialect of the C99 standard and compiled with a vendor-specific compiler, but native kernels are optionally supported as well. They describe the sequence of instructions within a single execution instance, called a *work-item*. Work-items that are grouped in the same *work-group* are executed concurrently.

Memory Model The memory hierarchy consists of four distinct regions for the kernels: *Global* memory that can be written and read by all work-items in all work-groups. *Constant* memory, a region of global memory that is initialized by the host and does not change during the execution. *Local* memory that is only accessible to work-items within the same work-group. *Private* memory owned by a single work-item and not visible by others.

Program Model OpenCL supports *Data-Parallel* and *Task-Parallel* models. Data-Parallel model involves a kernel operating on multiple elements of a data structure concurrently. Task-Parallel model can be defined as a set of independent programs operating concurrently.

3.2 OpenCL on GPUs and FPGAs

OpenCL has been implemented on a wide range of GPU devices, such as Nvidia and ATi GPU. In recent years, FPGA providers also start to provide tools to generate hardware design specification from OpenCL kernels, e.g., Altera Complete Design Suite [14] or Xilinx AutoESL [44]. To allow multiple OpenCL implementations from different vendors to co-exist on the same system, there is a `cl_khr_icd` extension [25]. It defines an *installable client driver (ICD) loader*, a unifying library that acts as a mediator between the different platforms. This enables the use of multiple heterogeneous devices in a single process without interference between implementations on different devices. Without this mechanism, the overheads of multiple processes and inter-process communication (IPC) between them are required.

4 Lane Detection Implementation

This section presents in details our lane detection approach, an extension of our preliminary work in [23]. The application 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. Note that the proposed method is vision-based and requires no knowledge of any physical parameters like position and orientation of the camera.

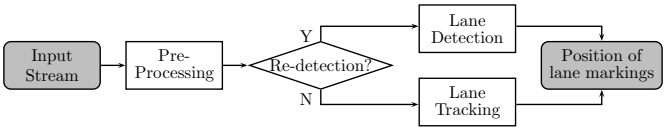


Fig. 1. Algorithm overview.

4.1 Pre-processing

The pre-processing is the first step for 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 to a grayscale space. 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 sub-steps for pre-processing is shown in Fig. 2.

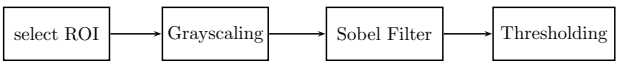


Fig. 2. Steps for pre-processing.

Fig. 4 in the experiment section shows images from KITTI-ROAD Dataset [17]. As the figures shown, only a small part of the images is of actual interest, i.e., the ROI showing the street. 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 a 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. More 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 a maximum value. 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 FPGA or GPU.

In the pre-processing step, every pixel of this ROI needs to be pre-processed. The settings of local work items and global work items depend on the device architecture. For the Nvidia GPUs listed in Tab. 1, the size of local work items is set to be 16×4, and the size of global work items is set to be $(\lceil \frac{ROI\_WIDTH}{16} \rceil \cdot 16) \times (\lceil \frac{ROI\_HEIGHT}{4} \rceil \cdot 4)$, because this setting matches the architecture of Nvidia GPU better. For the Altera FPGA, the sizes of both local work item and global work item are default to be 1 according to Altera OpenCL guidance [1].

4.2 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, lane detection is only performed when the lane tracking algorithm fails to track the markings.

4.2.1 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:

$$s_X = \frac{x_{bottom} - x_{top}}{\text{ROI\_HEIGHT}}, \qquad x_i = x_{top} + s_X \cdot i \tag{1}$$

In this manner, one can access the intensity value of any pixels for a given line X.

4.2.2 Intensity weight

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 an *intensity weight* to the candidate lines, which expresses how close the line is placed to a real lane marking. The intensity weight of a line is determined by summing up the intensities of all pixels in the line within the ROI. Further, also the pixels in an adjustable neighborhood around a line (left and right of the actual line) are added to this intensity weight. This accounts for the width of a lane marking. The intensity weight is therefore calculated as

$$w^I = \sum_{i=1}^{\text{ROI\_HEIGHT}} \sum_{j=1}^{2N_n} \text{intensity}(x_i,\, j), \tag{2}$$

where $2N_n$ is the neighborhood used to decide the width of the lane marking and x_i from Eqn. 1.

4.2.3 Detection

Keeping in mind that lane markings are represented by pixels with high intensities, this method results in candidate lines with higher intensity weights, if they are close to a

lane marking. The candidate line with the highest intensity 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 timing performance of the lane detection and its quality. Using more candidate lines will result in a better detection, but also consumes more computing power.

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 other 32% of the lines might overlap to other regions. This empirical choice of parameters proved to detect lane markings in all kind of positions from our later experiments.

4.2.4 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 intensity weights can be done in parallel. Hence, an OpenCL kernel is used to compute the intensity 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.

In this kernel, only the intensity of sampled lines need to be computed. Hence, according to the OpenCL guidance of both Nvidia and Altera, the local work size is set to be 8 and the global work size is set to be equal to the number of sampled lines. For every work item in the kernel, an indexed `sample_from_Gaussian` based on the work item id is loaded. The intensity of a sampled line is thus computed. The intensities of all sampled lines will be sent back to the host and the host will select the lines with the highest intensity as the best lines.

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 intensity weight amongst the good lines will be called the best line and represents the actual lane marking.

4.3 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.

4.3.1 Particle Filter Basics

Particle filter is a stochastic computational technique based on *Bayesian Signal Processing* and *Markov Chain Monte Carlo* simulation models. According to [9], Bayesian Signal Processing is concerned with the estimation of the underlying probability distribution of a random signal in order to perform statistical inferences. A Particle Filter implements the Bayesian recursion equation

$$P(X|Y) = \frac{P(Y|X)P(X)}{P(Y)} \tag{3}$$

where X is the desired state variable that is inferred from the measurement Y, the term $P(X|Y)$ [1] is called *posterior* probability distribution, $P(Y|X)$ is *likelihood*, $P(X)$ is *prior* probability distribution, and $P(Y)$ is *evidence*.

Eqn. 3 is implemented by sampling a number of weighted particles $i = 0, 1, \ldots, N$ from the prior distribution. Each particle has a state X_i. In addition, each particle is assigned an *importance weight* w_i. The importance weight expresses the likelihood that X_i is identical with the true state of the state-space model given the measurement Y, hence $w_i = P(Y|X_i)P(X_i)$. Inserting this to Eqn. 3 yields

$$P(X_i|Y) = \frac{P(Y|X_i)P(X_i)}{P(Y)} = \frac{P(Y|X_i)P(X_i)}{\sum_{i=0}^{N} P(Y|X_i)P(X_i)} = \frac{w_i}{\sum_{i=0}^{N} w_i}. \tag{4}$$

A particle filter is usually implemented in three consecutive steps in an iterative manner:

- Prediction update: The state X_i of each particle is updated in the same manner as the true state is expected to change.
- Importance weight update: The importance weight w_i is calculated for each particle.
- Resampling: This step is introduced that randomly samples particles according to their importance weight and copies them to a new, equally sized set. This yields a set with a higher density of good particles and avoids degeneration of the particle set.

A particle filter requires little or no assumptions on the model, only periodic (indirect) measurements of the true state. Indirect position measurements in the form of images are the only available information in a vision-based lane tracking algorithm, making a particle filter a promising tool for lane tracking.

4.3.2 Particle Filter setup

To use Particle Filter for the tracking. We match Eqn. 3 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 obtain, the best lines from the previous frame can be interpreted as observations of the lane markings in the current frame,

1. The notation $P(A|B)$ denotes a conditional probability, namely the probability of event A occurring subject to the condition that event B has already occurred.

assuming that the positions of the markings do not change significantly between two consecutive frames.

Following the aforementioned matching, below present the three steps to implement a Particle Filter.

4.3.3 Prediction update

Since the particles are already given in form of the good lines, sampling candidate lines from a normal distribution like in the detection phase is not necessary. Instead, prediction update step is introduced. This step is required, 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 the vehicle moves). The particles need to move the same distance in order to be a valid prior distribution in the new frame.

The distance the lane markings shift is not known. Therefore the particles are shifted by a random value sampled 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 optimal case. In our work, σ is empirically set to $1/16$ ROI_WIDTH.

4.3.4 Importance weight update

For each particle of the prior distribution the importance weight is calculated. This is done by applying Eqn. 4. As previously mentioned, 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

$$w^i_{X_t} = \frac{1}{\sqrt{2\pi\sigma^2}} \mathrm{e}^{-\frac{1}{2}(\frac{X_t-\mu_f}{\sigma_f})^2} \tag{5}$$

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 pointwise distance of line X_i and line Y. It yields the area between the two lines.

The evidence $P(Y)$, or marginal likelihood, is required in order to put the importance weight of different particles in relation to each other. The evidence describes the overall probability of the observation Y and is simply calculated as

$$P(Y) = \sum_i^N P(Y|X_i)P(x_i) = \sum_i^N w_i. \tag{6}$$

Therefore, the importance weight update is $w_i^{updated} = P(X_i|Y)$ of Eqn. 4.

4.3.5 Resampling

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 to prevent a degeneration of the set. The pseudo code for the resampling is shown in Algo. 1.

Algorithm 1 RESAMPLE

```
idx ← rand()%N_p; β ← 0.0                    ▷ N_p = #particle
for i = 1 : N_p do
    β+ = rand()%(2 * w_max);
    while β > w_idx do
        β− = w_idx; idx ← (idx + 1)%N_p;
    end while
    particle(i) ← particle(idx);
end for
```

Here, idx is an index drawn randomly from the particle indexes. The weight w_{max} belongs to the particle with the maximum weight after the update step. Variable β is assigned a random weight which is within $2 \times w_{max}$ and used to ensure that any particle with weight less than the value of β is skipped and all other particles are kept in the new set. As the new set contains the same amount of particles as the old set after the *resampling* step, there can be multiple copie of some particles from the old set in the new set. Usually, the higher the weight of a particle, the higher the probability of multiple copies of that particle in the new set. Nevertheless, some of the particles can still be completely missed out in the new set even if they have high weight.

4.3.6 Best lines update

Although the best lines of current frame are considered as the lane markings of next frame in updating the importance weight of particles, the best lines of current frame are not the real lane markings of next frame because the lane markings also move with the movement of vehicle. The collected intensity of a resampled good line still represents the closeness between the resampled good line and the real lane markings. Hence, the reampled good lines with the highest intensity weights are chosen as the best lines in each frame and continuously be used as the lane markings of next frame in updating the importance weight and resampling the particles.

4.3.7 Implementation

The prediction and importance weight update 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 GPU or FPGA. The procedure is similar to lane detection but with a different mean and deviation values mentioned in Sec. 4.3.3. The resampling and best lines update, in contrast, are dependent on information from all particles and are implemented on the host. The job of a work item in the lane tracking kernel is to compute the importance and intensity of a particle line. Therefore, the group work size is set to be the number of all particles so that every particle is used. The local work size is set to be 8 because this can fully utilize the accelerating device.

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 of both or either will lead to higher accuracy. Nevertheless, both numbers can be used to tune the trade-off between speed and accuracy of our algorithm.

4.4 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:

- Lane markings do not cross.
- Minimum distance between any detected lane markings. In this work, 20% of the width of the ROI.
- 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 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.

5 Evaluation Setup

This section describes the setup of our case study. Details of the chosen COTSs and test videos are depicted. In addition, the OpenCL runtime environment are presented.

5.1 Test cases

We choose five different COTS platforms, i.e., an Nvidia GeForce GTX 660 Ti GPU and an Altera Stratix V A7 FPGA (Nallatech PCIe385n_a7 card) mounted on a desktop with an Intel Intel Xeon processor, an Nvidia Quadro K600 on another desktop with an Intel Core 2 Quad desktop, a standalone Altera Cyclone V SoC FPGA, and a Redmi Note 2 mobile phone. The Cyclone SoC is embedded with a Cortex-A9 dual-core as the host. The Redmi Note 2 mobile uses the MediaTek Helio X10 octa-core on 2 GHz [2], with an on-chip PowerVR G6200 GPU. The detailed specifications of these platforms are listed in Tab. 1.

The choices of COTSs for evaluating the application performance are follows. First of all, the COTS platforms should be diverse to test the heterogeneity support of OpenCL. From this regard, GPUs and FPGAs, which represent two structurally different programming platforms, are chosen. In addition, a mobile GPU is tested as well. Secondly, we intend to test the scalability of our approach and extract the limits for a wider range of platforms, from powerful high-end to low-budgeted low-cost platforms. For high-end platforms, GTX 660 Ti GPU and Altera Stratix V A7 FPGA are chosen. For budgeted platforms, Quadro K600 GPU and Altera Cyclone V SOC FPGA are chosen. The Redmi Note 2 mobile is another type of low-budgeted platform, as the mobile itself costs around $100. In addition, Nvidia Jetson TK1, which is specially designed and targeted to automotive market for ADAS applications, is chosen for further comparison.

To test our lane detection application on these five COTS platforms, 14 video streams from different data-sets with different scenarios are used. The detailed information of these video streams is listed in Tab. 2. These videos can respectively represent different road situations, including night, blurred lane, and broken lanes. In these 14 video streams, the third one and the fourth one come from Caltech Lanes Dataset, while all others are self-recorded. The original videos and the lane detection results can be seen in this link[2].

5.2 OpenCL runtime

The Nvidia GeForce GTX 660 Ti GPU and Altera Stratix V A7 FPGA (on the Nallatech PCIe385n_a7 card), are mounted on an Intel Intel Xeon desktop via PICe bus. This allows a heterogeneous execution of our application, i.e., distributing the workload around GPU and FPGA during runtime.

In order to use different devices together, an Installable Client Driver (ICD) loader is needed. The ICD loader libraries act as a proxy between the user program and the actual implementations. It employs the C programming language library *dl* that is used on Linux to load libraries at runtime. Due to the absence of ICD loader in Altera AOCL 13.0sp1, a self-developed ICD loader was developed and implemented. The relationships between the libraries are shown in Fig. 3, where different vendor implementations can be loaded at runtime without the symbol conflicts.

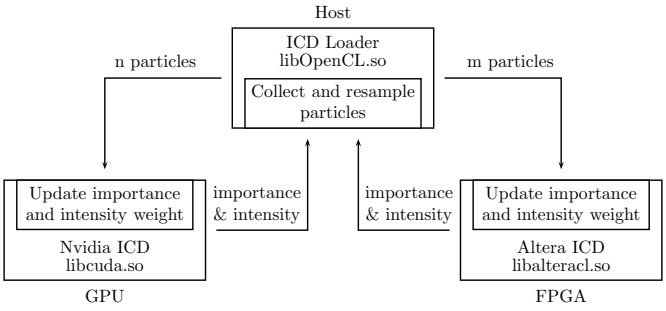


Fig. 3. Relation of ICD libraries in the case of lane tracking.

The communication between the host and devices is organized in a server-client model, as shown in Fig. 3. The host distributes the workload between the devices and collects the results when they are ready. This distribution is adjustable at runtime because frames are processed one after another independently.

6 Evaluation Results

This section presents experimental results and analyzes these results from different aspects.

6.1 Accuracy

First of all, the detection capability of our approach is evaluated. In order to do so, the KITTI-ROAD Dataset [17] is used for the evaluation, which consists of two independent marked subsets, 96 frames for urban marked two-way roads and 94 frames of urban marked multi-way roads. Our approach can detect lane markings in 92% of the images

2. https://www.youtube.com/playlist?list=PLMf5knkexT9JL9Hl6YgMk7HWL7_Bs_Zh9

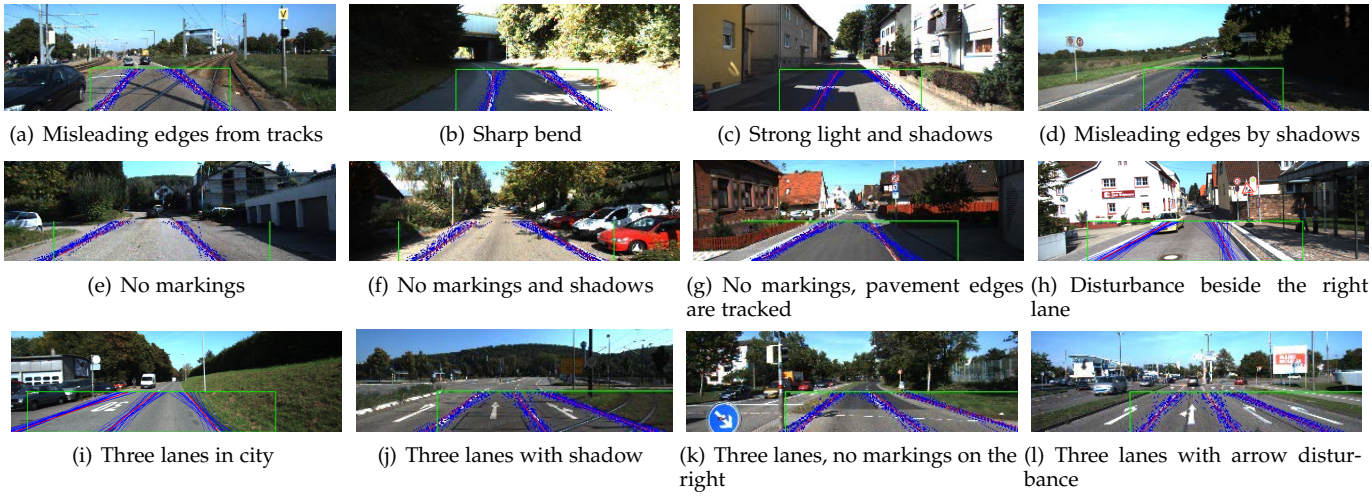
TABLE 1
Hardware specification of the COTS platforms.

	FPGA1	FPGA2	GPU1	GPU2	GPU3
Model	Nallatech 385-D5	Cyclone V SoC	GeForce GTX 660 Ti	Quadro K600	PowerVR G6200
Architecture	Stratix V 5SGXA7	ST 5CSTD6	Kepler GK104	Kepler GK 107	Mediatek Helio X10
Driver version	13.1	13.1	304.88	340.58	Android 5.0
Host CPU	Intel Xeon E31225	Cortex-A9 dual-core	Intel Xeon E31225	Intel Core 2 Quad Q9300	Cortex-A53 Octa-core
Host connection	PCIe	AMBA AXI	PCIe	PCIe	MTK MCSI
PCIe Generation	3.0	N/A	3.0	2.0	N/A
PCIe lanes	8x	N/A	16x	16x	N/A
On-board memory	8GB DDR3	1GB DDR3	2GB DDR5	1GB DDR3	2 GB DDR3
Max. Freq. (MHz)	600	210	915	876	700
Max. GFLOPS	294.7	56	2,460	336.4	89.6
Max. Power (W)	25 (board)	~4 (chip)	150 (board)	41 (board)	~4 (chip)
OpenCL version	1.1	1.1	2.0	1.2	1.2

TABLE 2
The detailed information of the 14 test videos.

Videos	1	2	3	4	5	6	7
Total frames	1285	895	250	406	1718	3056	4597
Detection	204	20	13	68	3	70	385
Tracking	1081	875	233	334	846	1442	4212
Resolution	360X640	360X640	480X640	480X768	480X640	480X640	360X480
Scenario	city	dark	bus view	blur lane	high way	outskirts	lane crossing
ROI size	72X512	72X512	96X384	96X384	96X512	96X512	72X384

Videos	8	9	10	11	12	13	14
Total frames	1385	938	2473	1313	2283	1512	2448
Detection	5	6	85	30	80	55	108
Tracking	1380	932	2388	1283	2203	1457	2340
Resolution	480X640	480X640	480X640	480X640	320X480	480X640	480X640
Scenario	night	broken lanes	street road in night	blur lanes in night	broken lanes in night	driving with traffic	light disturbance
ROI size	96X512	96X512	96X512	96X512	64X384	96X512	96X512



(a) Misleading edges from tracks (b) Sharp bend (c) Strong light and shadows (d) Misleading edges by shadows

(e) No markings (f) No markings and shadows (g) No markings, pavement edges are tracked (h) Disturbance beside the right lane

(i) Three lanes in city (j) Three lanes with shadow (k) Three lanes, no markings on the right (l) Three lanes with arrow disturbance

Fig. 4. Lane detection for KITTI-ROAD dataset.

from this dataset and only failed to detect when the Sobel filter creates no edges., Some of the especially challenging images are demonstrated in Fig. 4.

We also evaluate the impact of numbers of candidate lines(denoted as N_c) and particles (denoted as N_p) on the overall accuracy. Since our approach 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 the 14 videos in Tab. 2. The numbers of pixels deviated from the baselines are reported.

The results are shown in Fig. 6(a). From the figure, the first observation is that the accuracy increases when N_p increases. With $N_p = 128$, the average deviation is already below 4 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 approach can achieve high accuracy. The second observation is that when $N_p \geq 512$ the deviation is saturated to 3 pixels. The reason for the saturation at 3 pixels is that particle filter is a statistical result. Since every time particles are randomly distributed on the image, there exists inherent deviation between any two detection results. Although our reference baselines are calculated by a large N_c and N_p, this reference baselines cannot guarantee to be

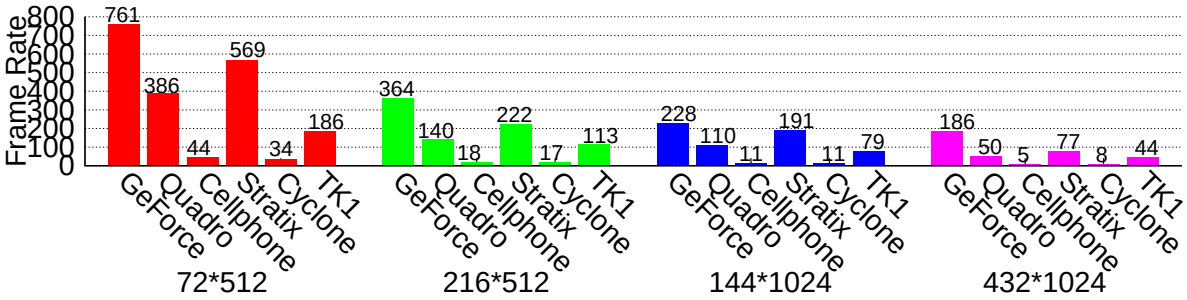


Fig. 5. Frame rate w.r.t. ROI.

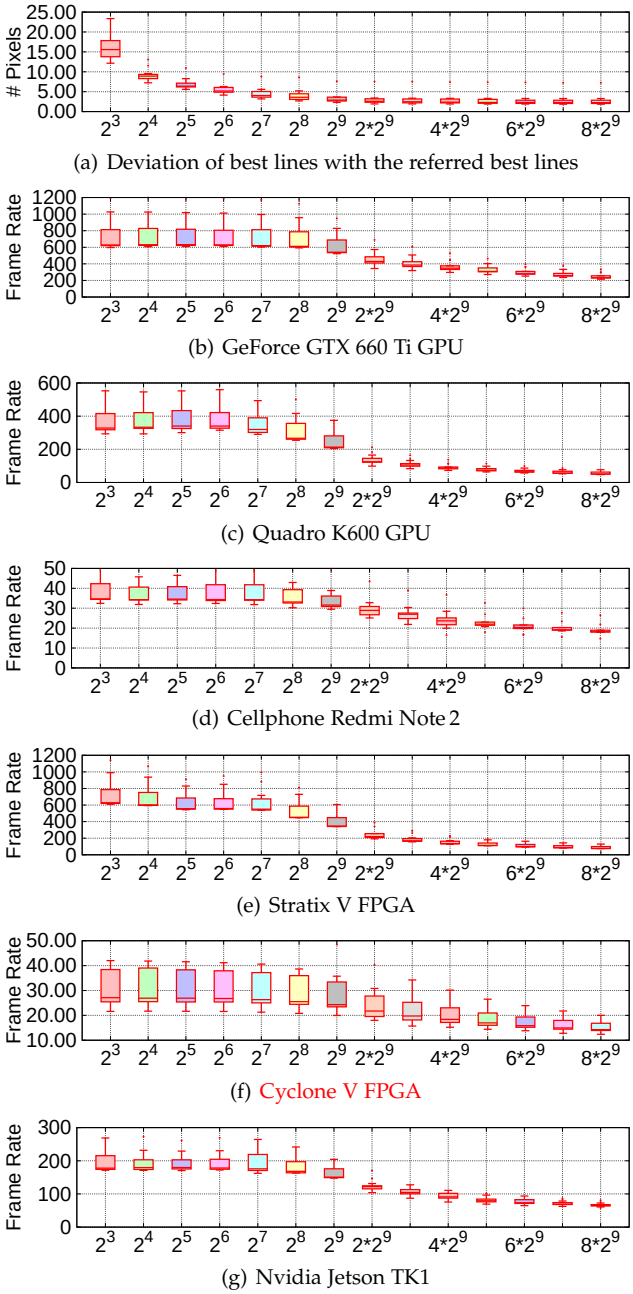
the real lane markings. It can happen that real lane markings are obtained with smaller N_c and N_p during the runtime. Besides, the real lane on the image is not a strictly straight line, which will also result in some deviations between the detected lane and the real lane.

6.2 Performance

We also report the frame rates for the five COTS platforms in Tab. 1 for all N_p and N_c combinations in Fig. 6(a). For each N_p and N_c combination, the 14 videos in Tab. 2 are checked and the corresponding frame rates are recorded for every platform. The results are shown in Fig. 6(b)–6(f). It can be seen that, the GeForce GPU achieves the highest frame rates, as expected among all COTSs (Fig. 6(b)). It can reach over 1000 fps for certain videos when the numbers of particles (N_p) are small. With $N_p \leq 512$, i.e., average deviation less than 3 pixels, the average frame rate can still reach more than 550 fps. For the Stratix V FPGA (Fig. 6(e), the frame rates are lower than the GeForce GPU but still super fast, about average 400 fps for $N_p \leq 512$ and almost 1000 fps for smaller N_p. Considering the Stratix is running at 200 Mhz, such performance is magnificent. The budgeted Quadro K600 GPU (Fig. 6(f)) can also reach relatively high performance, e.g., over 200 fps for average frame rate at $N_p = 512$. For the ultra-low power resource-restricted Redmi PowerVR GPU (Fig. 6(d)) and Cyclone V SoC FPGA (Fig. 6(f)), our algorithm can still reach real time. Considering the power consumption for these two chips is just 4 W, our algorithm is also 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. 5. 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. In general, the drops of the frame rates are sub-linear to the increases of the size of the ROI for devices. Nevertheless, our algorithm can still reach 77 fps on Stratix V FPGA for large ROI.

Furthermore, Nvidia Jetson TK 1 is used as a comparison to evaluate the performance. TK 1 is developed as a small super computer that aims at the use in autonomous driving and robot. Since it does not support OpenCL, our OpenCL code is ported into CUDA code and runs on TK 1. Compared to other GPUs in a PC platform, TK 1 architecture is optimized by using a share memory between CPU and GPU to save the time of data communication. Nevertheless, the frame rates (Fig. 6(g)) can only reach half of those on Quadro K600, though TK 1 and Quadro K600 have similar GFLOPS. Note that we ran the CUDA code on Quadro GPU as well



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

(b) GeForce GTX 660 Ti GPU

(c) Quadro K600 GPU

(d) Cellphone Redmi Note 2

(e) Stratix V FPGA

(f) Cyclone V FPGA

(g) Nvidia Jetson TK1

Fig. 6. Accuracy v.s. frame rate. The X axis is the number of particles N_p. For $N_p \leq 2^9$, $N_c = 2^9$, otherwise $N_c = 2^{12}$.

to compare the OpenCL and CUDA implementations. From this comparison, we found the OpenCL implementation is only 10% slower than the CUDA implementation. This shows that our CUDA implementation is not the cause of

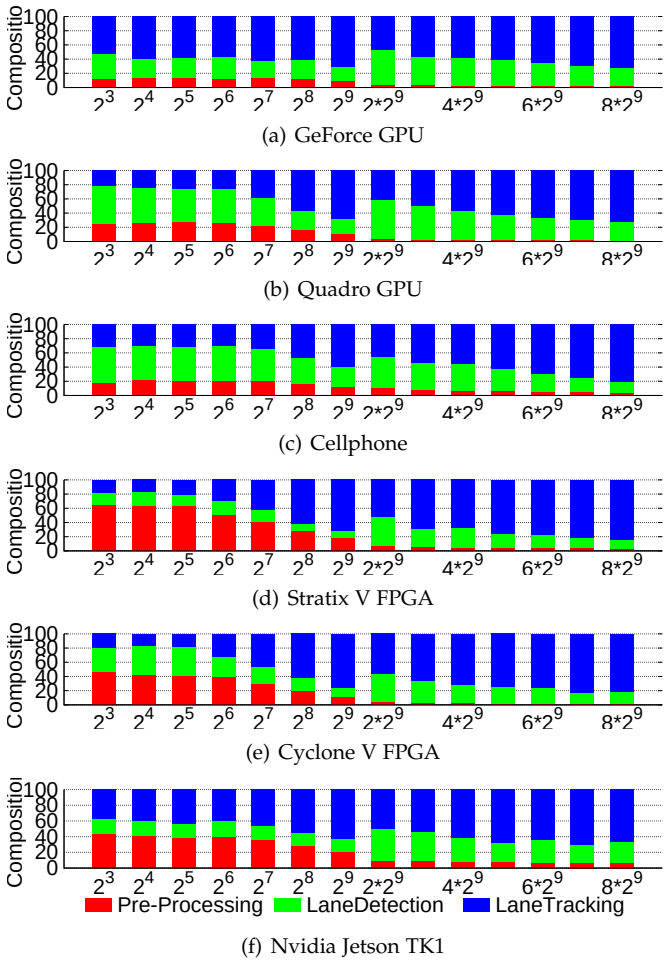


Fig. 7. Ratios of the average computation time of kernels on different platforms.

the low frame rate on TK1. We believe that the bottleneck of TK1 is the host processor, a 32-bit quad-core Cortex A15 and its 64-bit memory data width.

6.3 Timing Analysis

In our approach, there are three OpenCL kernels, i.e., pre-processing, lane detection, and lane tracking. Therefore, we investigate the detailed timing of these three kernels for all the experiments in Fig. 6. The normalized average computation time of all N_c and N_p combinations is shown in Fig. 7. From this figure, we have following observations. In general, given fixed N_c, increasing N_p will increase the timing percentage of lane tracking. The reason is that there are more particles needed to be processed with larger N_p. Another reason is that with less particles, i.e., smaller N_p, the tracking is less accurate and need more times of detection. On the other hand, the workload for pre-processing is fixed for a given ROI. Therefore, the ratios for pre-processing drop further as the number of N_p increases. The second observation is the ratios are different for different platforms, given the same N_c and N_p combination. In general, the two FPGAs have higher percentages for pre-processing than the GPUs. The reason is that more floating point operations are involved during pre-processing.

Lets take a closer look and consider the case $N_p = 256$ and $N_c = 512$. The normalized average total execution time and execution time per frame are shown in Fig. 8 and Fig. 9, respectively. For the total execution time (Fig. 8), the major computation is expected spending on lane tracking. The reason is that average 95% frames are conducted lane tracking. When we consider the computing time for individual frames (Fig. 9), lane detection consumes much more computing power than lane tracking, e.g., more than twice for GPUs, which justifies the need of a lane tracking step, rather than detection for every frames.

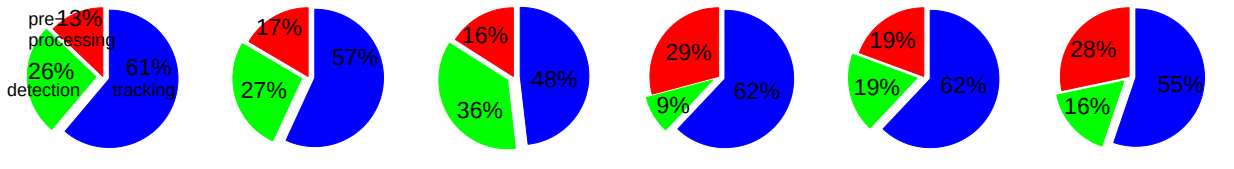
Fig. 10 shows the ratios of timing expense between the host CPU and devices. In general, the hosts have a bigger portion, i.e, about 80%. This means the hosts are always the bottleneck. The main reason for this bottleneck is the conversion of OpenCV structure of image frames into OpenCL peer. This conversion unfortunately cannot be paralleled with the devices. Comparing the Stratix V FPGA and GeForce GPU (Fig. 10(a) and 10(d)), who are mounted on the same machine with Intel Xeon processor, one can find the portion for the host is smaller (i.e., 58%) for Stratix V FPGA. The reason is that the Stratix V FPGA has much small computing resources than the GeForce GPU, i.e., slower. Although almost half computation time is spent on the Stratix V , the processing speed of Stratix V is still very fast as the frame rate for Stratix V is the second highest among all devices. The reason for the high percentage of Stratix V computation time is that the host processor is very powerful. For the case of Cyclone V, the host processor is a dual-core Cortex-A9, which results in the highest ratio for the host.

6.4 Heterogeneous Execution on Multiple Devices

In section 5.2, the framework of heterogeneous execution of OpenCL application using multiple devices is described. This section presents experiments of heterogeneous execution of our lane detection with Altera's Stratix V FPGA and Nvidia's GeForce GTX 660 TI GPU in combination. The results are summarized in Fig. 11.

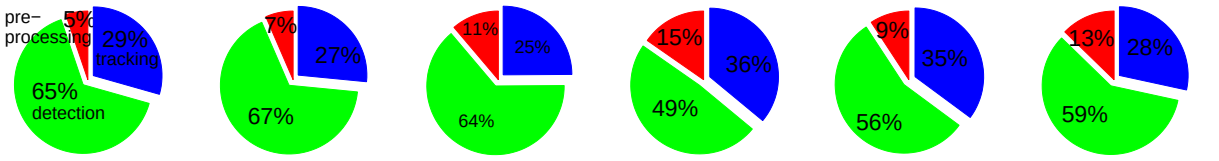
In this experiment, the number of sampled lines are set twice as the number of particles ($N_c = 2N_p$) and we tested four cases, each with different number of particles, i.e., $N_p = 64, 128, 1024$, and 10240. For each N_p, six scenarios are evaluated, i.e., the percentage of N_p computed on FPGA is set to be $0\%, 20\%, 40\%, 60\%, 80\%$, and 100%. When this percentage is 0% and 100%, all computations are solely executed on GPU and FPGA, respectively. In these two cases, there is no ICD overhead during the test because the ICD is not used when only one device is used.

There are three main observations from Fig. 11. First, the frame rates decrease as more workload are on the FPGA for most of the cases. This is because the GeForce GPU has higher computer power than the Stratix FPGA, as shown in Figs. 6(b) and 6(e). Second, the ICD overhead is non-trivial. Comparing scenarios 0% (without ICD) and 20% (with ICD) for cases 64, 128, and 1024, the frame rates drop significantly, i.e., 40.16%, 34.52%, and 27.55%, respectively. While comparing 20% to 40%, 40% to 60%, and 60% to 80%, the drops on frame rates are not significant as 0% to 20%. On the other hand, the frame rates increase again from 80% (with ICD) to 100%(without ICD). Third, the speed-up in computation by distributing workload to both devices is effective only when particles number is sufficiently



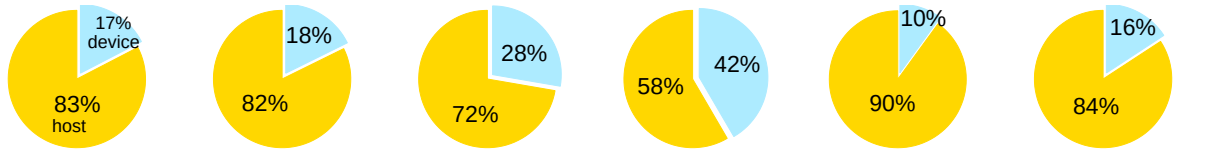
(a) GeForce GPU (b) Quadro GPU (c) Cellphone (d) Stratix V (e) Cyclone V (f) Nvidia Jetson TK1

Fig. 8. Normalized average total execution time of OpenCL kernels for $N_p = 256$ and $N_c = 512$.



(a) GeForce GPU (b) Quadro GPU (c) Cellphone (d) Stratix V (e) Cyclone V (f) Nvidia Jetson TK1

Fig. 9. Normalized execution time of OpenCL kernels for single frame for $N_p = 256$ and $N_c = 512$.



(a) GeForce GPU (b) Quadro GPU (c) Cellphone (d) Stratix V (e) Cyclone V (f) Nvidia Jetson TK1

Fig. 10. Distribution of the computation time between host and devices.

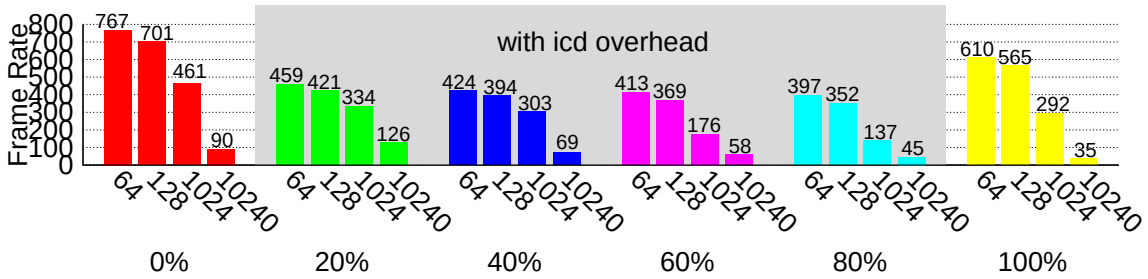


Fig. 11. Heterogeneous execution with Stratix FPGA and GeForce GPU, where the bottom numbers represent the percentage of particles on FPGA.

large where single device cannot cope with the required workload. Comparing scenarios 0% to 20%, only in the case of 10240 particles, the frame rate increases from 90 to 126, which the frame rates drop for all other cases. It indicates that the benefit of distributing workload overcomes the ICD overhead and thus increases the frame rate. In other cases, the workload distribution cannot compensate the ICD overhead and only results in performance decrease.

7 Conclusion

This article presents a case study for running ADAS application on COTSs with OpenCL. From this case study, we draw following conclusions. First, our lane detection developed with OpenCL can smoothly execute on both Nvidia GPUs and Altera FPGAs, individually and jointly. Although getting ideal performance on Redmi mobile needs customized optimization, such optimization does not affect the basic of the source code. Regarding the effort of coding in OpenCL, it took 6 months for a master student without any prior knowledge of parallel programming to develop a working prototype of the lane detection application. It took another 4 months for a bachelor student without no knowledge of OpenCL to optimize the memory consumption as well as the FPGA execution. Therefore, we can confidently say OpenCL can be considered as a viable standard programming model for ADAS development. Second, as shown the experiments, all the examined COTSs can provide more than sufficient computing power for lane detection. With proper manners that can exploit the available parallelism, we believe COTS platforms can provide enough computing power to support ADAS applications, even multiple applications jointly on single platforms. In summary, Using OpenCL to develop parallel ADAS applications on COTS platforms is a viable solution for future ADAS development.

Acknowledgments

The authors would thank Nikhil Madduri, Jan Botsch, Di Zhu, Zhihu Pan, Alibi Rakhmatulin, and Mingyue Cui for their contributions to the paper. This work is partly supported by German BMBF fund: 13N11936 and China SYSU 'the Fundamental Research Funds for the Central Universities': 15lgjc32.

REFERENCES

[1] Altera SDK for OpenCL - Programming Guide. *https://www.altera.com/content/dam/altera-www/global/en_US/pdfs/literature/hb/opencl-sdk/aocl_programming_guide.pdf*, 2015.

[2] GPU GFLOPS for Game Consoles/ARM, X86 SoC. http://kyokojap.myweb.hinet.net/gpu_gflops/, 2016.

[3] M. Aly. Real time detection of lane markers in urban streets. In *IEEE Intelligent Vehicles Symposium*, pages 7–12. Ieee, June 2008.

[4] 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.

[5] A. Borkar, M. Hayes, M. T. Smith, and S. Pankanti. A layered approach to robust lane detection at night. *IEEE Workshop on Computational Intelligence in Vehicles and Vehicular Systems*, pages 51–57, Mar. 2009.

[6] L. Brownsword, D. Carney, and T. Oberndorf. The opportunities and complexities of applying commercial-off-the-shelf components. *Crosstalk*, 11(4):4–6, 1998.

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

[8] 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.

[9] J. V. Candy. *Bayesian Signal Processing: Classical, Modern and Particle Filtering Methods*. Wiley-Interscience, New York, NY, USA, 2009.

[10] T.-C. Chen and K.-L. Chung. A New Randomized Algorithm for Detecting Lines. *Real-Time Imaging*, 7(6):473–481, Dec. 2001.

[11] K.-Y. Chiu and S.-F. Lin. Lane detection using color-based segmentation. In *IEEE Intelligent Vehicles Symposium*, pages 706–711, 2005.

[12] A. Cohen, V. Perrelle, D. Potop-Butucaru, M. Pouzet, E. Soubiran, and Z. Zhang. Hard Real Time and Mixed Time Criticality on Off-The-Shelf Embedded Multi-Cores. In *8th European Congress on Embedded Real Time Software and Systems (ERTS)*, Toulouse, France, Jan. 2016.

[13] W. Cunningham. Bmw adopts nvidia gpu for in-car displays. http://www.cnet.com/news/bmw-adopts-nvidia-gpu-for-in-car-displays/, 2011.

[14] T. Czajkowski, U. Aydonat, D. Denisenko, J. Freeman, M. Kinsner, D. Neto, J. Wong, P. Yiannacouras, and D. Singh. From opencl to high-performance hardware on fpgas. In *Intl. Conf. Field Programmable Logic and Applications (FPL)*, pages 531–534, 2012.

[15] F. Fons and M. Fons. Fpga-based automotive ecu design addresses autosar and iso 26262 standards. *Xcell journal*, 78:20, 2012.

[16] Freescale. OpenCL ADAS development environment addresses safety concerns. *http://automotive.electronicspecifier.com/*, 2015.

[17] J. Fritsch, T. Kuhnl, and A. Geiger. A new performance measure and evaluation benchmark for road detection algorithms. In *Intelligent Transportation Systems-(ITSC), 2013 16th International IEEE Conference on*, pages 1693–1700, 2013.

[18] M. Ghadhab, J. Kaienburg, M. Süßkraut, and C. Fetzer. Is software coded processing an answer to the execution integrity challenge of current and future automotive software-intensive applications? In *Advanced Microsystems for Automotive Applications*, pages 263–275. Springer, 2016.

[19] 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.

[20] K. Group. OpenCL. http://www.khronos.org/opencl/.

[21] P. Hough. Method and means for recognizing complex patterns. *US-Patent:3069654*, 1962.

[22] http://www.icinsights.com/. 2016 IC Market Drivers Report.

[23] K. Huang, B. Hu, J. Botsch, N. Madduri, and A. Knoll. A scalable lane detection algorithm on cotss with opencl. In *In Design, Automation and Test in Europe*, March 2016.

[24] F. Kastensmidt and P. Rech, editors. *FPGAs and Parallel Architectures for Aerospace Applications*. Springer, 2016.

[25] Khronos Group. OpenCL Extension #5: Installable Client Driver (ICD) Loader. https://www.khronos.org/registry/cl/extensions/khr/cl_khr_icd.txt, 2010. Retrieved on 2015.03.15.

[26] O. Kotaba, J. Nowotsch, M. Paulitsch, S. M. Petters, and H. Theiling. Multicore in real-time systems–temporal isolation challenges due to shared resources. In *Workshop on Industry-Driven Approaches for Cost-effective Certification of Safety-Critical, Mixed-Criticality Systems*, 2014.

[27] T. Litman. Autonomous vehicle implementation predictions. *Victoria Transport Policy Institute*, 28, 2014.

[28] 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 (IV)*, pages 993–997, 2010.

[29] X. Liu, B. Dai, J. Song, H. He, and B. Zhang. Real-time long-range lane detection and tracking for intelligent vehicle. In *Sixth Intl. Conf. Image and Graphics (ICIG)*, pages 654–659, 2011.

[30] C. Ma, L. Mao, Y. Zhang, and M. Xie. Lane detection using heuristic search methods based on color clustering. In *International Conference on Communications, Circuits and Systems (ICCCAS)*, pages 368–372, 2010.

[31] J. McCall and M. Trivedi. Video-based lane estimation and tracking for driver assistance: survey, system, and evaluation. *Intelligent Transportation Systems, IEEE Transactions on*, 7(1):20–37, 2006.

[32] M. Meuter, S. Muller-Schneiders, A. Mika, S. Hold, C. Nunn, and A. Kummert. A novel approach to lane detection and tracking. *2009 12th International IEEE Conference on Intelligent Transportation Systems*, pages 1–6, Oct. 2009.

[33] B. Motruk, J. Diemer, R. Buchty, R. Ernst, and M. Berekovic. Idamc: A many-core platform with run-time monitoring for mixed-criticality. In *High-Assurance Systems Engineering (HASE), 2012 IEEE 14th International Symposium on*, pages 24–31, Oct 2012.

[34] D. Muench, M. Paulitsch, and A. Herkersdorf. Temporal separation for hardware-based i/o virtualization for mixed-criticality embedded real-time systems using pcie sr-iov. In *27th International Conference on Architecture of Computing Systems (ARCS)*, pages 1–7, Feb 2014.

[35] P. Muyanözçelik and V. Glavtchev. Gpu computing in tomorrow's automobiles. http://www.nvidia.com/content/nvision2008/tech_presentations/Automotive_Track/NVISION08-GPU_Computing_in_Tomorrows_Automobiles.pdf, 2008.

[36] 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.

[37] J. Nowotsch, M. Paulitsch, A. Henrichsen, W. Pongratz, and A. Schacht. Monitoring and wcet analysis in cots multi-core-soc-based mixed-criticality systems. In *Design, Automation and Test in Europe Conference and Exhibition (DATE)*, pages 1–5, March 2014.

[38] M. Panić, G. Rodriguez, E. Quiñones, J. Abella, and F. J. Cazorla. On-chip ring network designs for hard-real time systems. In *Proceedings of the 21st International Conference on Real-Time Networks and Systems*, RTNS '13, pages 23–32, New York, NY, USA, 2013. ACM.

[39] M. Paolieri, E. Quiñones, F. J. Cazorla, G. Bernat, and M. Valero. Hardware support for wcet analysis of hard real-time multicore systems. *SIGARCH Comput. Archit. News*, 37:57–68, June 2009.

[40] M. Paolieri, E. Quinones, F. J. Cazorla, and M. Valero. An analyzable memory controller for hard real-time cmps. *Embedded Systems Letters, IEEE*, 1(4):86–90, Dec 2009.

[41] J. Ponce and D. Forsyth. *Computer Vision A Modern Approach*. 2012.

[42] P. Pop, L. Tsiopoulos, S. Voss, C. F. Oscar Slotosch, U. Nyman, and A. R. Lopez. Methods and tools for reducing certification costs of mixed-criticality applications on multi-core platforms: the recomp approach. In *WICERT 2013 proceedings*, 2013.

[43] C. E. Salloum, M. Elshuber, O. Hoeftberger, H. Isakovic, and A. Wasicek. The across mpsoc – a new generation of multi-core processors designed for safety-critical embedded systems. In *Digital System Design (DSD), 2012 15th Euromicro Conference on*, pages 105–113, Sept 2012.

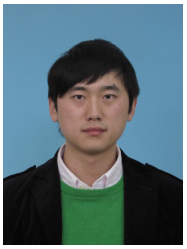
[44] K. Shagrithaya, K. Kepa, and P. Athanas. Enabling development of opencl applications on fpga platforms. In *IEEE 24th International Conference on Application-Specific Systems, Architectures and Processors (ASAP)*, pages 26–30, 2013.

[45] X. Shen, Z. J. Chong, S. Pendleton, G. M. James Fu, B. Qin, E. Frazzoli, and M. H. Ang. *Intelligent Autonomous Systems 13: Proceedings of the 13th International Conference IAS-13*, chapter Teleoperation of On-Road Vehicles via Immersive Telepresence Using Off-the-shelf Components, pages 1419–1433. Springer International Publishing, Cham, 2016.

[46] R. B. Sorensen, M. Schoeberl, and J. Sparso. A light-weight statically scheduled network-on-chip. In *NORCHIP, 2012*, pages 1–6, Nov 2012.
[47] J. Sparso. Design of networks-on-chip for real-time multi-processor systems-on-chip. *2010 10th International Conference on Application of Concurrency to System Design*, 0:1–5, 2012.
[48] J. Sparso, E. Kasapaki, and M. Schoeberl. An area-efficient network interface for a tdm-based network-on-chip. In *Design, Automation Test in Europe Conference Exhibition (DATE), 2013*, pages 1044–1047, March 2013.
[49] K. Taylor. Altera functional safety package combines fpga flexibility with lockstep processor solution to reduce risk and time-to-market. http://newsroom.altera.com/press-releases/nr-altera-functional-safety-yogitech.htm, 2015.
[50] T. Ungerer, C. Bradatsch, M. Gerdes, F. Kluge, R. Jahr, J. Mische, and J. F. *et al*. parmerasa – multi-core execution of parallelised hard real-time applications supporting analysability. In *Digital System Design (DSD), 2013 Euromicro Conference on*, pages 363–370, Sept 2013.
[51] T. Ungerer, F. Cazorla, H. Casse, S. Uhrig, I. Guliashvili, M. Houston, and F. K. *et al.* Merasa: Multicore execution of hard real-time applications supporting analyzability. *Micro, IEEE*, 30(5):66–75, Sept 2010.
[52] K. Zhao, M. Meuter, C. Nunn, D. Muller, S. Muller-Schneiders, and J. Pauli. A novel multi-lane detection and tracking system. In *IEEE Intelligent Vehicles Symposium (IV)*, pages 1084–1089, 2012.

Kai Huang Kai Huang joined Sun Yat-Sen University as a Professor in 2015. He was appointed as the director of the Institute of Cyber Physical Systems of School of Data and Computer Science in 2016. He was a senior researcher in the Computer Science Department, the Technical University of Munich, Germany from 2012 to 2015 and a research group leader in fortiss GmbH in Munich Germany in 2011. He obtained his Ph.D. degree in ETH Zurich, Switzerland in 2010. his MSc from University of Leiden, the Netherlands in 2005, and his BSc from Fudan University, China in 1999. His research interests include techniques for the analysis, design, and optimization of embedded systems, particularly in the automotive domain. He was awarded the Program of Chinese Global Youth Experts 2014 and was granted the Chinese Government Award for Outstanding Self-Financed Students Abroad 2010. He was the recipient of Best Paper Awards ESTIMedia 2013, SAMOS 2009, and General Chairs' Recognition Award For Interactive Papers in CDC 2009. He is a regional editor for Elsevier Journal of Circuits, Systems, and Computers, and have served as a member of the technical committee on Cybernetics for Cyber-Physical Systems of IEEE SMC Society since 2015.

Biao Hu Biao Hu received the B.Sc. degree in Control Science and Engineering at Harbin Institute of Technology, received the M.Sc. degree in Control Science and Engineering at Harbin Institute of Technology from Tsinghua University. He is working toward the PhD degree in Department of Computer Science at the Department of Informatics of the Technische Universität Mnchen. His research interests includes the autonomous driving, OpenCL computing in heterogeneous system, the scheduling theory in real-time systems, and safety-critical embedded systems.



Long Chen received the B.Sc. degree in communication engineering and the Ph.D. degree in signal and information processing from Wuhan University, Wuhan, China, in 2007 and in 2013, respectively. From October 2010 to November 2012, he was co-trained Phd Student at National University of Singapore. From 2008 to 2013, he was in charge of environmental perception system for autonomous vehicle SmartV-II with the Intelligent Vehicle Group, Wuhan University. He is currently an Assistant Professor with the School of Data and Computer Science, Sun Yat-sen University, Guangzhou, China. His areas of interest include computer vision, point cloud processing and unmanned autonomous vehicles.

Alois Knoll Alois C. Knoll received the diploma (M.Sc.) degree in Electrical/Communications Engineering from the University of Stuttgart, Germany, in 1985 and his Ph.D. (summa cum laude) in Computer Science from the Technical University of Berlin, Germany, in 1988. He served on the faculty of the Computer Science department of TU Berlin until 1993, when he qualified for teaching computer science at a university (habilitation). He then joined the Faculty of Technology of the University of Bielefeld, where he was a full professor and the director of the research group Technical Informatics until 2001. Since autumn 2001 he has been a professor of Computer Science at the Department of Informatics of the Technische Universität Mnchen. He was also on the board of directors of the Central Institute of Medical Technology at TUM (IMETUM); between April 2004 and March 2006 he was Executive Director of the Institute of Computer Science at TUM. Between 2007 and 2009, he was a member of the EUs highest advisory board for information technology, ISTAG, the Information Society Technology Advisory Group, and a member of its subgroup for Future and Emerging Technologies (FET). In this capacity, he was actively involved in developing the concept of the EUs FET Flagship projects, and he was one of the authors of the original FET-Flagship report. His research interests include cognitive, medical and sensor-based robotics, multi-agent systems, data fusion, adaptive systems, multimedia information retrieval, model-driven development of embedded systems with applications to automotive software and electric transportation, as well as simulation systems for robotics and traffic.

Zhihua Wang Zhihua Wang received the B.S., M.S., and Ph.D. degrees in electronic engineering from Tsinghua University, Beijing, China, in 1983, 1985, and 1990, respectively. In 1983, he joined the faculty at Tsinghua University, where he is a Full Professor since 1997 and Deputy Director of Institute of Microelectronics since 2000. From 1992 to 1993, he was a visiting scholar at Carnegie Mellon University. From 1993 to 1994, he was a Visiting Researcher at KU Leuven, Belgium. From September 2014 to March 2015, he was a Visiting professor in Hong Kong University of Science and Technology. His current research mainly focuses on CMOS RF IC and biomedical applications. His ongoing work includes RFID, PLL, low-power wireless transceivers, and smart clinic equipment with combination of leading edge CMOS RFIC and digital imaging processing techniques. He is co-authors of 11 books and book chapters, more than 118 paper in international Journals and over 350 papers in international Conferences. He is holding 75 Chinese patents and 4 US patent. Prof. Wang has served as Deputy Chairman of Beijing Semiconductor Industries Association and ASIC Society of Chinese Institute of Communication, as well as Deputy Secretary General of Integrated Circuit Society in China Semiconductor Industries Association. He had been one of the chief scientists of the China Ministry of Science and Technology serves on the expert committee of the National High Technology Research and Development Program of China (863 Program) in the area of information science and technologies from 2007 to 2011.