IMPLEMENTATION OF A COMPUTER VISION BASED ADVANCED DRIVER ASSISTANCE SYSTEM IN TIZEN IVI

Gorka Velez, Orti Senderos, Marcos Nieto
Researchers, Vicomtech-IK4
Oihana Otaegui
Head of Intelligent Transport Systems and Engineering Unit, Vicomtech-IK4
Paseo Mikeletegi 57, 20009, Donostia-San Sebastián, Spain
Tel: +34943309230, {gvelez,osenderos,mnieto,ootaegui}@vicomtech.org

Geoffroy Van Cutsem
Senior Technical Marketing Engineer, Intel Corporation SA
Veldkant, 31, 2550 Kontich, Belgium
Tel: +3234500851, geoffroy.vancutsem@intel.com

ABSTRACT

There are many reasons for programming in Tizen: it is based on standards, is open, it has industry support and multiple profiles. In this paper we test the potential of one this profiles: Tizen IVI (In-Vehicle Infotainment). For this purpose, we present the implementation of two Computer Vision based Advanced Driver Assistance Systems (ADAS) in a Tizen IVI platform: a Lane Departure Warning (LDW) and a Driver Fatigue Detection (DFD). The results show that both application implementations are capable of achieve a real-time performance. Moreover, the process of porting an existing application to a native Tizen IVI application can be really smooth if some considerations are followed.

Keywords: Advanced Driver Assistance System, Tizen, Computer Vision, Algorithm implementation

INTRODUCTION

The Advanced Driver Assistance Systems (ADAS) market is expected to grow significantly in the following years, from a demand of 60 million ADAS units in 2013 to a demand of more than 100 million units in 2018. The market research firm ABI Research predicts that the 60% of the world's cars and that 80% of the North American and Western European cars will include features like smartphone connectivity and built-in Internet in five years' time (1). All these data confirm the increasing importance of ADAS for the automotive industry.

Among the different options, computer vision based solutions have gained a strong foothold due to their advantages over other type of sensors like LIDAR or radar, such as lower costs, higher resolution or better adequacy to embedded systems and cost-effective platforms. However, the development of an embedded vision device for ADAS is not straightforward. Several requirements must be taken into account: real-time computational performance, low cost, small size, low power consumption and short time-to-market. The automotive industry has proposed

different solutions to solve this problem, as it is usual on still non-mature markets. Some have used programmable hardware to accelerate the processing of the implemented computer vision algorithms. However, ADAS applications not only need to do image processing, but they also need to communicate with other devices and offer a usable user interface. Developing a vision based ADAS application in a FPGA or an ASIC is a too cumbersome task that prolongs the developing cycle. Here we propose software solution as the best option to obtain a short development cycle.

In the past, rigid custom built proprietary solutions dominated the market. However, these solutions were unable to keep up with the pace of innovation of Information and Communications Technologies (ICT). Nowadays, leading vendors are starting to develop initial Linux based systems. Although QNX and Microsoft are still leading the market with their proprietary operating systems (OS), Linux-based OS presence is expected to grow significantly on the following years. Following this trend towards open source solutions, Tizen IVI appeared as a new option for next generation in-vehicle infotainment systems built on Linux.

In order to study the potential of Tizen IVI, we have implemented two computer vision based ADAS applications. The following three sections describe the platform, the selected applications and their implementation. Then, the obtained results are presented and discussed. Finally, the conclusions are explained.

PLATFORM DESCRIPTION

Software

Tizen is an open and flexible operating system built from the ground up to address the needs of all many connected device profiles such as Smartphones, Cameras, Wearables, SmartTV, In-Vehicle Infotainment and others. In our case we have chosen Tizen IVI (In-Vehicle Infotainment), as it is designed specifically for the automotive market (2). More specifically, we have used the Tizen IVI 3.0-M2-March2014 version.

Tizen IVI architecture definition is driven by requirements gathered from the automotive industry, both directly and also via industry alliances and initiatives such as GENIVI (3) and AGL (4). Tizen IVI is designed to control the display of the centre console as well as the backseat screens. It can access all the car sensors and implements a strong security mechanism to isolate control of some critical parts. It is also designed to interact and integrate with the passengers' mobile devices. Tizen IVI is not only useful for infotainment applications, as it can also integrate a wide range of automotive applications and services, such as ADAS, traffic management, remote diagnosis or remote vehicle monitoring and control.

Hardware

The use of mobile phones for ADAS purposes is very tempting, as they are already familiar for the user and they have all the necessary applications and connections configured. However the use of mobile phones while driving can dramatically increase crash risk. On the other hand, an in-

vehicle computer based system works even when the phone does not, has more flexible inputs, and more and better data access.

The Tizen IVI OS was finally installed on a Nexcom VTC 7120-BK computer (Figure 1). This fanless computer is specially designed for in-vehicle applications, and features an Intel® Celeron® Processor 847E 1.1GHz, 2GB of RAM, two high speed interface for storage, 2.5" SATA and CFast, and several I/O options.

Figure 1 Nexcom VTC 7120-BK in-vehicle computer.

The captured images are obtained from an IDS UI-2210SE camera. This camera has a CCD sensor from Sony which delivers a resolution of 640 x 480 pixels.

IMPLEMENTED APPLICATIONS

There are many ADAS applications that use computer vision technologies. Two of them, Lane Departure Warning (LDW) and Driver Fatigue Detection (DFD), were chosen to be implemented in Tizen IVI as case studies. LDW was chosen because it is a mature integrated technology already in the market that is representative of applications based on a forward looking camera. On the other hand, DFD was chosen because it is a not yet widely commercialized application whose real-time implementation is far more challenging (5).

Lane Departure Warning (LDW)

The objective of a LDW application is to warn the driver for unintentional lane departures. Thus, the solution must be able to detect and track in real-time the road's lane markings and determine the position of the vehicle inside its own lane. In our case, we have implemented a vision system that uses perspective assumptions and geometric road models and that has been shown to provide excellent results in real-time (6).

Figure 2 shows the output of the computer vision algorithm. The two white thick lines represent the detected lanes. After the computer vision stage, there is a semantic analysis that has as a result the information to activate or deactivate the lane departure warning signal.

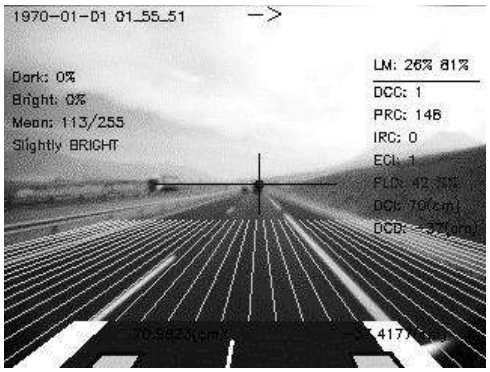

Figure 2 Lane detection.

Driver Fatigue Detection (DFD)

In the implemented DFD solution, computer vision is used to determine the attention of the driver, defining its level of drowsiness or fatigue by means of analysing biometrics like eyelid closure, blinking speed and frequency. The application is focused on a user-based detection and tracking of the driver eyes using paired eyes-model.

Figure 3 shows a screenshot of the DFD solution. The blue boxes define the detected eyes, and the green boxes represent the regions of interest.

Figure 3 Driver Fatigue Detection.

IMPLEMENTATION METHOD

The Tizen platform provides two different types of frameworks for application development: the Web framework can be used to develop HTML5/JavaScript applications, while the native framework can be used to develop core middleware and applications. In our case we have used the native framework to develop our two ADAS applications, as computer vision algorithms are very performance demanding.

The whole code was written in C++ in a desktop PC with the aim of being cross-platform. It was previously tested in Windows 7 and Ubuntu Linux, and once all the detected errors were corrected, it was ported to Tizen IVI. For this purpose, all the code was recompiled in target platform. In order to develop cross-platform solutions is highly recommended to use CMake. It is used to control the software compilation process using simple platform and compiler independent configuration files. It is designed to support directory hierarchies and applications that depend on multiple libraries.

Figure 4 depicts the architecture and the dependencies of the implemented ADAS applications. Both applications have the same dependencies and were developed using the Viulib software library (7), which is a cross-platform SDK written in C++ that can be used to acquire, process and analyse in real-time video sequences simplifying the building of complex artificial vision applications. The rest of the dependencies can be downloaded from the Tizen repository using the *zypper in* command. OpenCV is used by Viulib for some computer vision functions, Qt is used to build the graphical interface, and FFmpeg and V4L are used by OpenCV to capture frames from the camera.

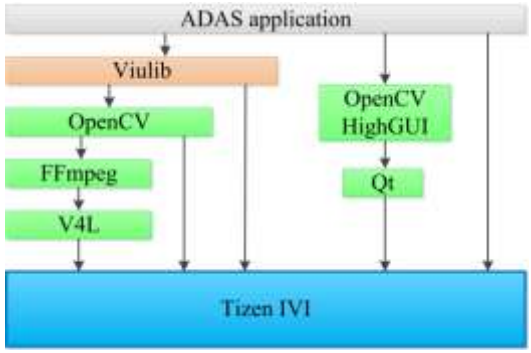

Figure 4 Application's dependencies.

It is worth noting that the Tizen IVI platform is better suited for serial processing, so it is important to avoid as many operations at pixel level as possible. This includes convolution operations, colour conversions or image resizing. In that sense, the number of image filtering stages was minimised as much as possible in our implementations.

RESULTS

Both applications were successfully implemented in Tizen IVI. Table 1 shows the profiling times for LDW implementation. The total computation time is 8.201 ms, which is well below the maximum 40 ms necessary for real time. In this application each frame is processed with a resolution of 320 x 240 pixels.

Table 1 Profiling times for LDW implementation.

Module	Time (ms)
Obtain frame from camera	1.372

Image pre-processing	0.396
Lane markings detection filter	5.71
Perspective histogram	0.656
Tracking	0.046
Parameter adjustments	0.004
Semantic analysis	0.011
Total	**8.201**

The DFD algorithm is not so easy to profile. The processing time of each frame depends on the scene and on driver's behaviour. If the driver moves constantly his head, the tracking time can increase dramatically, and at some point, the track can even get lost. Not only this, the algorithm first attempts to detect opened eyes, and if it fails, it tries to find closed eyes. So the processing time is bigger when the driver has the eyes closed. Additionally, if during 10 frames the algorithm did not find any eye, it would execute a face detection algorithm in order to determine a new region of interest. For all these reasons, and in contrast to LDW, there is no constant execution time for DFD. However, in a normal situation where the driver is getting drowsy, his movements are slow, so the tracking module does not need much time to track the position changes. In this case, the variation on the processing time of each frame is not too high. In order to test if the DFD algorithm could detect on real time if someone is getting drowsy several videos were recorded with different people. In these videos, the head movements of the drivers were smooth, as on real fatigued drivers. Table 2 shows the mean profiling times for the best case, which is the most frequent. That is, when the driver has the eyes opened and they were also detected on the previous frames. Table 3 shows the mean profiling times for the worst case, i.e. when the driver has the eyes closed and a face detection stage is needed.

Table 2 Profiling times for DFD implementation: best case.

Module	Time (ms)
Obtain frame from camera	1.372
Eye detection	3.302
Tracking	6.601
Filter tracking	0.802
Total	**12.076**

Table 3 Profiling times for DFD implementation: worst case.

Module	Time (ms)
Obtain frame from camera	1.372
Face detection	1.342
Eye detection	7.518
Tracking	6.601
Filter tracking	0.802
Total	**17.634**

On both cases the total processing time is below 40 ms, which means that on normal conditions, the algorithm runs in real time.

DISCUSSION

The proposed two ADAS application have been successfully implemented in a Tizen IVI platform. Both application implementations were capable of achieve a real-time performance. In our implementation methodology, we first developed and tested the applications on a desktop PC running Ubuntu Linux. Then, we recompiled all the code in the target platform: an in-vehicle computer running Tizen IVI. This process can be straightforward if you stick to using only standard C++ code and you do not use any dependency that cannot be installed in Tizen IVI. In our case, we have used our own cross-platform library, Viulib, and some other libraries that can be downloaded from Tizen IVI repository. This repository is publicly available and contains all the essential packages for developing ADAS applications.

ADAS software applications usually run on top of an operating system. The performance decreases when using such an application instead of using a standalone application, but it is absolutely worthwhile. Operating systems offer considerable savings in development time and in maintenance of the system. Additionally, when using an operating system the programmers can focus on the specific computer vision algorithms without having to care about other low level details. The number of programming errors is reduced when using a higher abstraction level and the obtained source code is more portable.

The use of an open source operating system has important advantages compared to proprietary solutions. First of all, the development cost is lower, as there is no need for paying licensing fees. Furthermore, the platform is publicly assessed and rated by different partners and potentially also by academic institutions. Consequently, there is a higher confidence on the product because more people can inspect the source code to find and fix possible vulnerabilities. Finally, automotive suppliers can reach market faster, due to the time-saving advantage of reusable open source code.

In the particular case of Tizen, it has some other additional advantages. Tizen is backed by a large group of industry operators and device manufacturers such as Samsung, Intel, Fujitsu, LG, Orange or Vodafone. Moreover, as it was said before, Tizen IVI architecture definition is driven by requirements gathered from the automotive industry, both directly and also via industry alliances and initiatives such as GENIVI and AGL. So it can be said that it has a strong industry support.

Tizen was designed with multiple device profiles in mind. There are currently multiple profiles under active development: In-Vehicle Infotainment (IVI) systems, Smartphones, SmartTVs, cameras, Smart Watches, printers… This makes Tizen a strong candidate for being de facto standard operating system for Internet of Things implementations, which would make the developing of ADAS applications in Tizen IVI even more attractive.

CONCLUSIONS

The process of porting an existing application to a new operating system can become a really challenging task, especially if it was not planned previously. In order to avoid problems is important to keep in mind that in the future the same application could run in a different system. In that sense, using cross-platforms libraries can help in minimizing the burden of porting. In the

case of Tizen IVI, it supports many cross-platform libraries that run on Windows or on different Linux distributions. This facilitates enormously the porting or developing of native applications in Tizen IVI, as many of available libraries and tools are already well known.

There are many reasons for programming in Tizen: it is based on standards, is open, it has multiple profiles and it has industry support. In this paper we present the implementation of two computer vision based ADAS application in an in-vehicle computer that runs Tizen IVI. The results obtained show that both application implementations are capable of achieve a real-time performance, demonstrating the potential of this platform.

REFERENCES

(1) Ron Schneiderman, "Car makers see opportunities in infotainment, driver-assistance systems", *IEEE Signal Processing Magazine*, 2013, 30(1):11–15
(2) https://wiki.tizen.org/wiki/IVI
(3) http://genivi.org/
(4) http://automotive.linuxfoundation.org/
(5) Marcos Nieto, Oihana Otaegui, Gorka Vélez, Juan Diego Ortega, Andoni Cortés, "On creating vision-based advanced driver assistance systems", IET Intelligent Transport Systems, 2014, DOI: 10.1049/iet-its.2013.0167
(6) Marcos Nieto, Andoni Cortés, Oihana Otaegui, Jon Arróspide, Luis Salgado, "Real-time lane tracking using Rao-Blackwellized particle filter", Journal of Real-Time Image Processing, 2012, DOI: 10.1007/s11554-012-0315-0
(7) "Viulib 13.10", http://www.vicomtech.org/viulib/