Optimising computer vision based ADAS: vehicle detection case study

Marcos Nieto[1] ✉, Gorka Vélez[1], Oihana Otaegui[1], Seán Gaines[1], Geoffroy Van Cutsem[2]

[1]Vicomtech-IK4, Paseo Mikeletegi 57, San Sebastian, Spain
[2]Intel Corporation SA, Veldkant, 31, 2550 Kontich, Belgium
✉ E-mail: mnieto@vicomtech.org

Abstract: Computer vision methods for advanced driver assistance systems (ADAS) must be developed considering the strong requirements imposed by the industry, including real-time performance in low cost and low consumption hardware (HW), and rapid time to market. These two apparently contradictory requirements create the necessity of adopting careful development methodologies. In this study the authors review existing approaches and describe the methodology to optimise computer vision applications without incurring in costly code optimisation or migration into special HW. This approach is exemplified on the improvements achieved on the successive re-designs of vehicle detection algorithms for monocular systems. In the experiments the authors observed a ×15 speed up between the first and fourth prototypes, progressively optimised using the proposed methodology from the very first naive approach to a fine-tuned algorithm.

1 Introduction

Computer vision is a reality for the advanced driver assistance systems (ADAS) industry. Its continuous growth in the last decade is due to factors such as the reduction of hardware (HW) costs, improved performance of applications and algorithms, and the increased market opportunities of car makers [1].

This industry, however, imposes some strong requirements that must be satisfied to reach the mass market. Namely, the legal aspects, user acceptance, low power consumption, size and weight of the devices, integration with onboard computers, and very reduced time to market (TTM) development.

Although existing prototype ADAS and automated vehicles have been successfully demonstrated (DARPA Grand and Urban Challenge, Google driverless car, EUREKA Prometheus, to name a few), they usually do not meet those criteria, particularly because of the large volume and cost of the sensors used.

Therefore, there are two main challenges for the developers of ADAS, which both relate to optimisation steps needed to move from research to market: (i) algorithm performance optimisation; and (ii) development cycle optimisation for low TTM.

The first relates to the need to migrate laboratory algorithms to work within an embedded platform, real-time being the main goal to achieve. The solution to this problem may be achievable with long, expensive, re-programming tasks, where the code is analysed and migrated to VHDL for field-programmable gate arrays (FPGA), or parallelised with general-purpose computing on graphics processing units (GPGPU) or digital signal processors (DSP) primitives. The required development cost goes against the low TTM premise: this type of migration and redeployment involve high financial costs and time for development teams. In general it also involves enormous efforts of coordination between the multidisciplinary teams, time extensions, etc.

Several work methodologies have emerged to find an appropriate trade-off. One is the so-called co-design methodology [2], which basically consists in a coordinated work plan for HW and software (SW) teams around the flexibility of reconfigurable HW platforms, where the FPGA logic can be reprogrammed and adapted to changes in the SW design. However, this option still requires the presence of multidisciplinary teams and coordination effort. Another option is the adoption of open standards platforms and

systems, providing integration interfaces, already validated by the industry itself, which allow the development team to concentrate on the optimisation of the algorithm [3]. Here we consider the latter option as the best to obtain an adequate trade-off.

This paper contributes presenting a design and development methodology that addresses the two main challenges for ADAS developers: algorithm performance optimisation and development cycle optimisation. Following this proposed methodology, computer vision algorithms can be optimised without the need to recode or include code optimisation procedures. The result are incrementally optimised prototypes implemented in an open source platform that ensures rapid development cycles. To illustrate this, we also present a case study where the proposed design and development methodology is applied for a vehicle detection algorithm.

The rest of the paper is organised as follows. Section 2 provides an overview of related work about implementation platforms, design and development methodologies and vehicle detection algorithms. Section 3 describes the proposed design and development methodology and Section 4 presents a case study to show how the proposed methodology is suitable for an ADAS vision application. Section 5 presents a discussion, and finally, Section 6 concludes the paper.

2 Related work

2.1 Implementation platforms

Different implementation platforms have been proposed to meet the requirements of computer vision based embedded systems: high computational performance and reliability, and low cost, size, power consumption, and TTM. Traditionally, DSPs have been used for image signal processing. DSPs offer single-cycle multiplication and accumulation operations, in addition to parallel processing capabilities and integrated memory blocks. There are many examples in the related literature that use DSPs [4–6].

However, DSPs imply a much higher cost compared with other options such as FPGAs [7]. An FPGA is an integrated circuit designed to be configured by a customer or a designer after manufacture. FPGAs take advantage of high speed operation, especially for parallelisable operations. A similar approach is the

use of application-specific integrated circuits (ASIC). In contrast to FPGAs, which are also used for rapid-prototyping, ASICs are used only for high volume manufacturing and long series due to higher initial engineering cost. In addition, they have another major drawback: they are not reconfigurable. During the last decade, a large number of implementations have appeared both ASIC [8, 9] and FPGA [10–14].

Due to the difficulty of implementing an entire ADAS application in HW, hybrid solutions emerged which combined SW and HW implementations [15–18]. These system on a chip (SoC) devices are usually built by implementing the architecture of a microprocessor in an FPGA or ASIC. One of the most competitive options is the Xilinx Zynq-7000 family, which is able to boot independently of the FPGA. This feature has a number of benefits being the most remarkable that from a SW engineer's point of view, the Zynq-7000 device looks, feels and behaves just such as a general purpose multicore processor. However, it is only worthy if the programmable logic part is utterly required. Otherwise it is much cheaper to use a regular microprocessor.

The use of microprocessors in embedded computer vision-based systems has experienced a significant growth in recent years. Computer vision applications are implemented in a microprocessor in two main ways: as standalone SW, or as a process running on top of an operating system (OS). The first approach obtains better computational results, since it does not have the burden of an OS running on background. However, although the performance decreases when using an application that runs on an OS, there are great savings in development time and in the maintenance of the system.

In the context of SW designed for automotive applications, 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. Today, the main suppliers are starting to develop initial Linux based systems [19]. Although QNX and Microsoft continue to lead the market with their own OS, Linux-based OS presence is expected to grow significantly in the coming years. Following this trend, Tizen in-vehicle infotainment (IVI) [20] and Android Auto [21] appeared as new candidates for next generation IVI systems built on Linux. Tizen IVI is not only designed with IVI in mind but can also integrate a wide range of automotive applications and services, such as ADAS, traffic management, remote diagnosis or remote vehicle monitoring and control.

2.2 Design and development methodologies

Depending on the chosen implementation platform, the appropriate design methodology changes. Although there are some general approaches in the field of engineering design and creative design [22, 23], the applied design methodology must be adapted to fit perfectly with the characteristics of the implementation platform.

In the case of using a DSP, a microcontroller or microprocessor, the obtained solution is a SW application. Professional SW development models are normally based on an engineering perspective and they have traditionally emphasised the following [24]:

- The separation of analysis and design.
- Design as a way to comply with the requirements specification.
- Hierarchical decomposition of the design work.

A classical approach is the waterfall design that was first defined in [25]. In a waterfall design the implementation steps to develop a computer program start with gathering system requirements and finish with the testing and operation stages. All the steps are sequentially followed without iterations. This basic waterfall design has been highly criticised because of the assumption that the use situations and needs can be known and specified before any prototype is built.

More modern approaches have emerged in the field of SW engineering, such as agile development [26], but the focus is still in the coordination of general purpose SW development teams

rather than addressing the specific problems of developing computer vision based embedded systems.

In the case of SoC design methodologies, HW decisions are traditionally taken first. After an initial specification, HW architecture is designed based mainly on the experience of the HW design team and the desired device cost. This methodology has some major drawbacks. First, it can delay SW teams, since in some cases the SW development and testing cannot start until the HW design is available. Furthermore, it can also delay the whole product design chain if a critical HW design error is detected late. Finally, there is a risk for overdesigning or underdesigning the system due to the lack of an initial evaluation of SW's computational requirements, which often depend reciprocally on the HW specifications.

To improve the design chain modern HW/SW codesign methodologies emerged [2, 27]. HW/SW codesign can be defined as the concurrent design of HW and SW to implement a desired function. Using codesign methodologies, before designing the final platform, a preliminary functional SW prototype is developed using a more flexible platform: a standard PC. After analysing and evaluating this SW prototype, the final platform's architecture is designed.

2.3 Vehicle detection

Vehicle detection is critical for in-vehicle systems as it is the basis of pre-crash sensing, collision avoidance and, eventually, autonomous driving. Approaches using active sensors such as laser or radar have provided valid solutions, although with a number of drawbacks, including low spatial resolution, high cost, slow scans, and more importantly, limited flexibility. In contrast, cameras can provide a richer description of the scene, which can be extracted and interpreted by increasingly complex algorithms that combine traffic sign recognition, pedestrian detection, lane tracking, etc.

In a very general way, all techniques for detection and tracking of vehicles consist of two stages: hypotheses generation and hypotheses verification. The former usually implies a quick search, that broadly identifies the image regions likely containing vehicles (this stage can also be accomplished by range sensors); the latter uses more complex algorithms that verify that the selected candidates are vehicles or not [28] based on the visual information contained in the selected image region.

The trend in vision-based systems is to use detection-by-classification methods [29]. These detectors use mathematical models to classify a given image patch as belonging to a class or not (e.g. 'car', 'no-car'), using the available data sources (e.g. greyscale image intensities, motion patterns, and even depth for stereoscopic setups) to extract discriminant features [30]. The models are obtained by training a classifier with a sufficiently large database with positive and negatives examples of the class. The detections can then be fed into tracking mechanisms, which track the vehicles across frames, or start online learning threads that update the models to enhance the detections of the observed vehicles [31].

For a more comprehensive review of the recent advances and trends in video-based on-road vehicle detection and tracking the reader can refer to existing surveys [32].

3 Design and development methodology

The proposed design and development methodology for computer vision based ADAS is based on two main pillars:

(i) Development cycle optimisation for low TTM.
(ii) Algorithm performance optimisation.

The first point deals with design decisions that are taken during the whole design cycle that can lead to a reduction in TTM without compromising the requirements imposed by the automotive industry. The second point deals with the internal design, that is, the construction of the SW. In that sense a list of general good

practices for computer vision algorithm development is presented that leads to enhanced algorithm performances.

3.1 Development cycle optimisation

In this paper we propose the use of a SW solution running on top of an open source OS as the best option to obtain a short development cycle. There are a large number of drivers for Linux based OS, which make the development of ADAS applications much easier. It is worth losing some computational performance to shorten substantially the development time and to gain flexibility. Eventually, real-time OS (e.g. Buildroot) can be used to remove the overhead. Furthermore, when using an OS the programmers can focus on the specific image processing algorithms without having to care about other low level details. A higher abstraction level reduces programming errors and makes the source code more portable.

First, the code is written and tested in C++ pursuing cross-platform capabilities and using a branching model as distributed version control SW (e.g. git). It is therefore tested in a variety of different OS (at least Windows and Linux) and once all the detected errors are corrected, it is ported to the final OS. For this purpose, all the code is recompiled for the target platform. This process can be straightforward if only standard C++ code is used and there is not any dependency that cannot be installed in the target OS. It is also highly recommended to use a compiler-independent method to manage the build process of SW. For this purpose, CMake [33] or any other similar application can be used.

As a key feature of our methodology, instead of devoting effort to the reimplementation of the same algorithm to create new prototypes that better exploit HW parallelisation capabilities or more efficient coding, we consider that it is better to return to the design of the algorithm itself. Previously unnoted bottlenecks in the algorithm might have been discovered after profiling in the target platform.

This procedure reduces dramatically the time and costs of creating new prototypes, since it is typically much harder to increase the performance of a given algorithm rather than finding a better algorithm. Fig. 1 illustrates this cyclic development methodology.

As shown in the figure, each algorithm design step is preceded by the previous performance report with identified bottlenecks, and the definition of HW requirements (which may have been revised after the last profiling analysis). An initial implementation might be necessary in the first iterations only to verify that the algorithm is functional (i.e. it does what it is supposed to do), and then jump to the optimisation task which aims to accelerate the execution time of the algorithm without changing its functionality. The optimisation step can be accomplished using a number of possible strategies. Next section presents these ideas, with an assessment of the expected impact on performance improvement and deployment difficulty.

Moreover, in this methodology, additional training and testing mechanisms can be added to provide additional information to the performance report, which, therefore, not only refers to processing times, but also to the quality of the results (please see Section 4 for a detailed description).

In summary, successive optimised and tested prototypes can be generated in an agile form using this methodology.
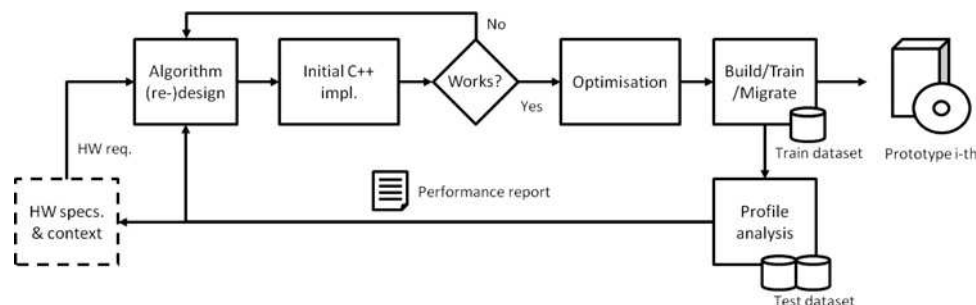
3.2 Algorithm performance optimisation

The following table illustrates a number of options for optimising computer vision algorithms in the general context of ADAS applications. They have been ordered top to bottom from highly recommended to last chance to get something working (including the worst-case migration to HW-specific languages).

Normally, a good start consists in applying item #1, that is, open source libraries containing parts of already deployed, tested algorithms, and that set the basis for rapid prototyping. After completing the first cycle, obvious improvements typically are given by the application of items #2 and #3, that is, simplify the problem by using contextual information or multiple simpler algorithms to improve the performance. Further improvements are more related to fine-tune the algorithm deepening and discovering redundant operations (item #4), or to the application of code optimisation and programming skills, such defined in items #5 and #6. The remaining elements are probably useless taking into account both the expected improvements and deployment times, hence they should be avoided unless market-ready, mature prototypes have been completed and fast time to market is no longer a requirement.

4 Case study: vehicle detection

This case study shows how the methodology defined in Section 3 can be applied in a real case: a vehicle detection algorithm. First, the need for optimisation is explained. Then, the specific tasks carried out to optimise the development time and performance of the algorithm are described. Finally, the obtained results are presented.

4.1 Problem statement

Vehicle detection is typically achieved intensely scanning the images, covering the entire image with a sliding window technique, and applying the classifier to each candidate window. These schemes can be applied in image pyramids without changing the window to detect objects of variable size without the need to know their size in advance. This multiscale approach is computationally very expensive, although successful in finding all instances of the class (car) in the image, regardless their position or size.

4.2 Methods

4.2.1 Prototype #1 'Multiscale': This type of approach is the basic approach we chose at first glance, including the usage of a detector-by-classification with a trained model of images of cars in forward looking cameras. We used the combination of HOG features and a support vector machine (SVM) classifier from OpenCV and additional simpler features such a shadows, edges and symmetry, and used them to train an Adaboost model, which works as a low false negative classifier. The more reliable and computationally expensive SVM classifier is then used to verify Adaboost detections (the block diagram of the approach is
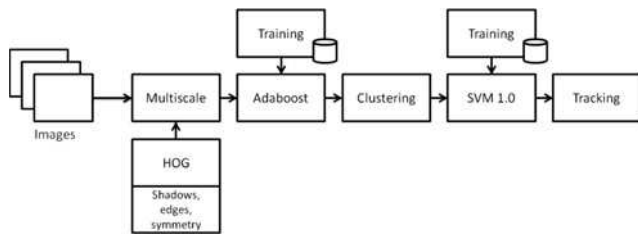

Fig. 1 *Work cycle for optimising computer vision-based ADAS SW modules*

Fig. 2 *Block diagram of Prototype #1: vehicle detection 'multiscale'*

Fig. 4 *Number of candidates proposed by the sliding window approach for different values of f, and L for an image of 600 × 400 pixels. The model size is 48 × 48, the stride is 8 × 8*

illustrated in Fig. 2). This algorithm gives good results in terms of detection rates, but at the cost of excessive processing time.

In particular, the sliding window works starting from the smallest window, that is, the smallest size of object to be detected in the image, and creating L resized copies of the image, downscaled by a factor f which is typically set from 1.05 to 1.1. This procedure generates an enormous amount of candidate regions for analysis, variable according to the values of f, L and the size of the image (see Fig. 3a).

In our experiments we had to use at least 20 levels involving about 60,000 classifier evaluations with a total processing time above 300 ms in the selected platform to be able to detect close and far distance vehicles (see Fig. 4 for an example illustration of the number of candidates thrown by a multiscale approach).

4.2.2 Prototype #2 'Perspective multiscale':
Although it may seem simple to try to re-implement the multiscale approach for massive parallelisation HW (item #9 of Table 1), it is much convenient to re-design this brute-force algorithm to focus the processing effort (CPU or GPU/FPGA) and to exploit existing prior information that is at hand (item #2 of Table 1). In this sense we created a second prototype which exploited the existence of the perspective information. When known, it is possible to pre-compute realistic candidate locations where vehicles can appear in the images and their approximate sizes. Therefore, we can replace the multiscale approach and create a custom grid of locations to search vehicles (see Fig. 3b). This procedure dramatically reduces the number of executions of the classifier, and keeps similar detection rates [34, 35]. Moreover, in this prototype we used an improved implementation of the SVM algorithm, consisting in the utilisation of the comparison between the descriptor vector with the hyperplane instead of the computation of the sum of the distances over all the support vectors of the model (update from OpenCV 2.4.6 to OpenCV 2.4.9; following item #1 of Table 1) (Fig. 5).

4.2.3 Prototype #3 'Two-stages perspective multiscale':
In the third iteration we found that the joint use of HOG and additional features (such as shadows, edges and symmetry) can be decoupled and used at different stages. These additional features are compu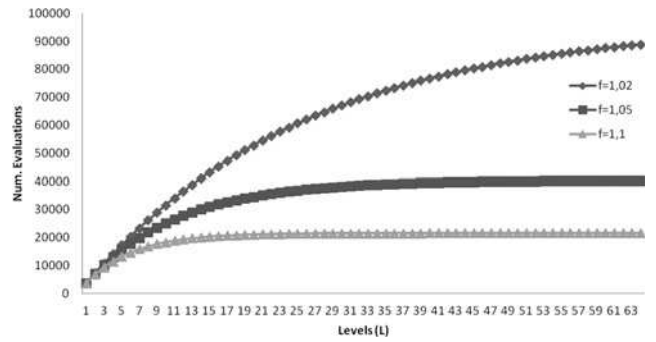ted faster and work well as a first filter, while the HOG features are applied only to the strongest candidates (items #4 and #6 of Table 1). It is then possible to reduce the complexity of the hypothesis generation stage by replacing the application of classification procedures for all candidates of the grid by a pre-detection algorithm. Thus most negatives of the grid are discarded, and the effort of the classifier is exploited efficiently. For additional speed up we also used integral images to achieve fast detection rates [36].

We also replaced the clustering submodule which grouped overlapping detections by a perspective-based clustering approach that takes into account possible occlusions that may arise due to the effect of perspective (item #2 in Table 1) (Fig. 6).

4.2.4 Prototype #4 'Day/night switch':
The latest addition to our prototype is an image brightness analysis that can switch between day and night detection modes (item #3 of Table 1). The appearance of vehicles at night is much different than during the day and so are the required detection models. In particular, the detection is roughly based on the presence of bright spots, coherent with the shape and position of vehicle tail lights [37]. Such a detector is lighter than SVM-based detectors and therefore the identification of night-time images (also in tunnels) can lead to less processing time and better detection performance (Fig. 7).

4.3 Results

Tables 2 and 3 show the average values of processing time for each sub-module of the different prototypes described above (using a sample test sequence of 2500 frames). The implementation takes into account only the use of CPU, so that neither code optimisation nor utilisation of GPU/FPGA are reported in this figure, which aims to show the speed-up factor that can be achieved with a better organisation of the data process flow. The
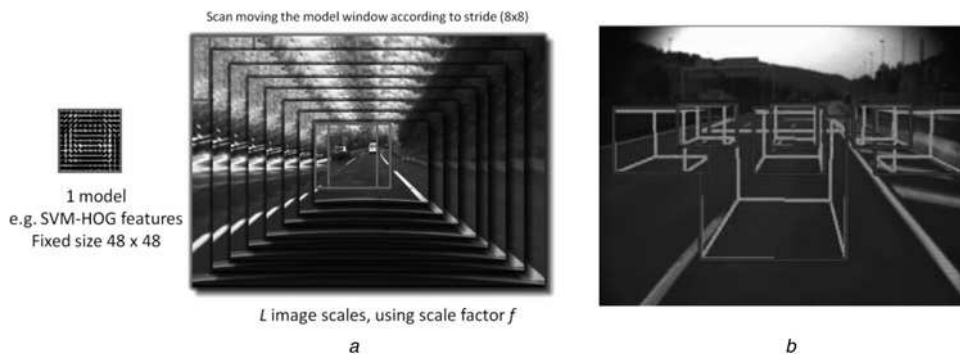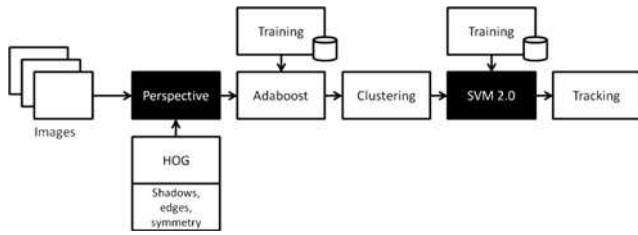

Fig. 3 *Multiscale detection based on sliding window and perspective contextual information*
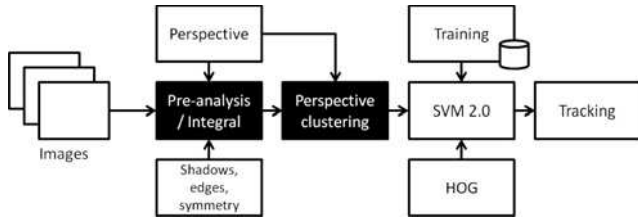*a* Multiscale detection
b Grid of candidate regions according to the perspective

Table 1 List of optimisation-related good practices applicable in the context of computer vision applications for ADAS

Expected impact	Deployment difficulty/cost	Description
very high	very low	**(#1) Effective exploitation of multiplatform libraries**: the design cycle for electronics are rapidly decreasing [1], partly due to the existence of libraries that prevent reprogramming everything. In the field of computer vision, libraries such as OpenCV, OpenCL, OpenGL, or ROS provide a great help for rapid prototyping. The use of such libraries involves finding a the right balance: on the one hand, they help to speed development times in a way that an experienced designer can create new prototypes in short time and without the need to hire skilled programmers. On the other hand, they provide atomic modules that can tempt the designer to use them without clear knowledge of their behaviour or even when they do not correspond with the best-theoretical solution. Adopting existing libraries help bringing proven, functional modules to the system, such as robust visual features such as histograms of oriented gradients (HOG) or SVM classifiers, which have been shown to give excellent results to visually describe objects with recognisable shapes such as persons or cars [34, 35] and has since its publication become a de-facto standard for robust non-deformable object detection in images.
very high (can focus analysis to pre-defined or dynamically designated pieces of information)	mid (algorithms need to be reshaped to interpret contextual information)	**(#2) Use contextual information as much as possible**: it is common to have some information related to the scene, the kind of images, etc. that can be used as prior or contextual information by the analytics algorithm. Typically, such use may result in a large reduction in computational complexity, because fewer number of detectors or sub-processes need to be applied, or the range of parameters can be narrowed (e.g. using perspective information, as we show in the case study in Section 4, helps reduce the number of candidate windows to evaluate to detect vehicles in images).
high (higher for finer identification of situations)	mid (several algorithms need to be designed, and an additional selection algorithm)	**(#3) Find operating modes and create multiple simpler algorithms**: it is difficult to design an algorithm that works well for all situations the application may need (e.g. day/night function, as explained in the use case of Section 4). The inertia of the designer is to increase the complexity of the algorithms so that they can handle a greater variety of conditions. However, it is sometimes more effective to have simpler algorithms that work well for specific conditions and a switch function that determines the operation mode.
mid	mid (requires profiling and full understanding of algorithm)	**(#4) Avoid redundant operations**: some algorithms internally use basic information such as gradient, colours, etc. Using several of such high-level algorithms could involve that several identical image filters are applied (e.g. HOG feature extractor internally uses the Sobel detector for gradient approximation). This knowledge may be overlooked by inexperienced developers without the supervision of a design engineer.
mid (depends on HW capabilities)	high (the entire code need revision)	**(#5) Balance between operations and memory**: some operations are slow, especially those related to trigonometric or probability functions. Quantised arrays can be pre-computed and stored in memory so that arithmetic operations are substituted by memory accesses (each HW platform has its own specifications of time consumption comparing memory access and CPU operations).
mid (depends on HW capabilities)	mid (might imply reshaping parts of the code, such as loops and callbacks)	**(#6) Pre-compute repetitive computations**: some operations must be done in execution time and cannot be pre-computed. However, they can be computed once and kept in memory for the rest of the process. Therefore, an analysis of memory resources must be done to consider fixed memory used by the program (in C++, the memory allocated for the creation of objects), dynamic memory (the memory temporarily allocated and deallocated, such as when running specific functions that require copies of the images to be done and deallocate the memory when leaving the scope), and the run-time fixed memory, which is the memory allocated at some point of the process and kept in memory until the end.
low (not solvable with SW)	low (profile analysis required)	**(#7) Analyse memory transfer times**: the use of various CPU or auxiliary GPU, DSP or FPGA can dramatically speed up operations through parallelisation. However, HW imposes its physical restrictions, and such use involves moving large blocks of memory from one processors to another, which might result itself in large transfer times which make the whole migration useless.
low (depends on programmers ability)	low (requires profiling)	**(#8) Avoid unnecessary operations**: Avoid operations that produce useless results: for example, filtering an entire image with a gradient filter mask when only bright pixels are interesting. Instead, it is better to identify such pixels in advance and filter only them (note that this knowledge might arrive later on the design process and thus it is impossible to correct unless a flexible approach is adopted). In general, careful design should not propagate, or at least minimise, the computation of intermediate information, and focus the computational resources effectively.
very high	very high	**(#9) Multicore processing capabilities to distribute asynchronous tasks**: the central processing unit (CPU), (e.g. an ARM processor) will lead the creation and coordination of threads communicating with other devices such as GPU and FPGA, which might be operating on asynchronous information (e.g. 3D reconstruction at key frames, recognition of traffic signs when detected, etc.)

Fig. 5 *Block diagram of Prototype #2, vehicle detection 'perspective multiscale': in black, the blocks modified, reimplemented or added*

Fig. 6 *Block diagram of Prototype #3, vehicle detection 'pre-detection perspective multiscale'*
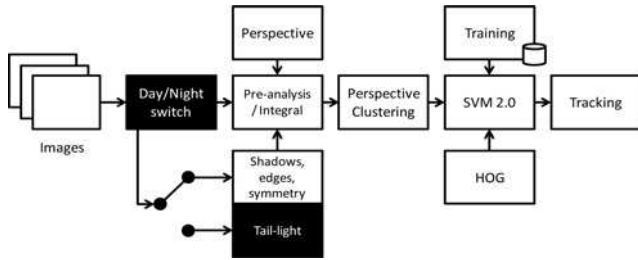
Fig. 7 *Block diagram of Prototype #4, vehicle detection 'day/night switch'*

gain introduced by the reimplementation of each submodule in GPU/FPGA falls outside the scope of this paper.

The tests were carried out in two different phases. During the first one we compared the first three prototypes implemented in an embedded platform (Xilinx's Zynq 7000), to find an approach that provides real-time performance in a platform suitable for use in real applications (results are reported in Table 2).

The second round of tests compared the third and fourth prototype in two different platforms: a general purpose PC and a fanless in-vehicle PC equipped with Tizen IVI OS (results reported in Table 3), in an attempt to verify the feasibility of our final approach in a standard in-vehicle platform.

In Table 2 we can see that the extraction of features and the computation of the HOG descriptors is the most time consuming part of Prototype #1. The introduction of the perspective-based processing in Prototype #2 reduced the number of levels and then the HOG time is reduced to half. In Prototype #3, the pre-analysis and use of integral images boosted the speed of the computation of features, and most importantly, the computation of HOG features is moved to the verification stage, and therefore executed only for very strong candidates. The average HOG time is then approximately 1 ms. For the SVM algorithm, we found a better implementation for Prototype #2, in which each descriptor vector was compared by scalar product with the hyperplane of the SVM model rather than the sum of the distances of the support vectors as it was done by the SVM implementation we used in Prototype #1.

In all three cases the detection results are similar, while the speed up achieved by Prototype #3 is about ×15 compared with Prototype #1 and ×6 with respect to Prototype #2. As a result, Prototype #3 is ready to be functional in real scenarios, and further implementation improvements (such as migration of submodules to GPU/FPGA, utilisation of SIMD/MIMD architectures, or reprogramming more efficiently algorithms) would only increase the performance and give room for more applications to be launched in the same processor, such as pedestrian detection, traffic sign recognition, lane departure warning, etc.

Once this goal was achieved, we included a small change that allowed the system to switch between day and night modes, giving Prototype #4 as a result. This minor change allows the selection of features to be applied to the images at the cost of having an initial day-night detection module.

The performance of this latest Prototype #4 was tested against ground truth sequences, to evaluate how accurately vehicles can be detected. We collected a set of 59 videos with a total approximate duration of 363 min, in a variety of settings, including varying weather conditions (e.g. rain), day and night sequences and different types of vehicles (i.e. cars and buses, basically). For practical reasons, we only manually annotated a subset of representative videos, grouped into three categories: day, rainy and night. The total number of annotated frames is 15,450, and the number of vehicles in the images is 44, 105 being the average number of frames in which each vehicle can be seen in the images. The annotations were stored in a frame-wise and object-wise manner. For each frame, all vehicles in the region of interest were given a bounding box representing its position, and a number representing its identity.

Therefore, the evaluation can be done on two levels. At object level, where we get true positives (TP), false positive (FP) and false negatives (FN) at vehicle level, that is, a single vehicle appearing in a sequence of 1000 frames will result in 1 TP if correctly detected, or 1 FN if not, and as many FP as unmatched detections. The association between ground truth and detections is done in a spatiotemporal manner, in a way that a ground truth object is associated (i.e. TP) with a detection if there is sufficient temporal and spatial similarity. For this type of analysis we adopt

Table 2 Comparison of the performance (in ms per frame) of the first three prototypes on the embedded platform

	Features	Adaboost	Perspective clustering	HOG	SVM	Tracking	Total (average)
prototype #1	135.88	18.45	–	252.23	13.67	0.29	420.52
prototype #2	56.43	5.88	–	112.56	4.12	0.24	179.23
prototype #3	11.23	–	10.22	1.19	4.25	0.27	27.16

Table 3 Performance of Prototype #4 for PC (Ubuntu 12.04) and Nexcomm (Tizen IVI), measured in ms per frame

Platform	Sequence	Day/night switch	Features	Perspective clustering	SVM-HOG	Tracking	Total (average)
PC	day	0.12	2.96	0.25	0.47	0.37	4.40
in-vehicle	day	0.16	9.88	0.92	0.57	0.21	11.85
PC	night	0.12	0.43	0.11	–	0.05	0.77
in-vehicle	night	0.16	0.71	0.51	–	0.14	1.62
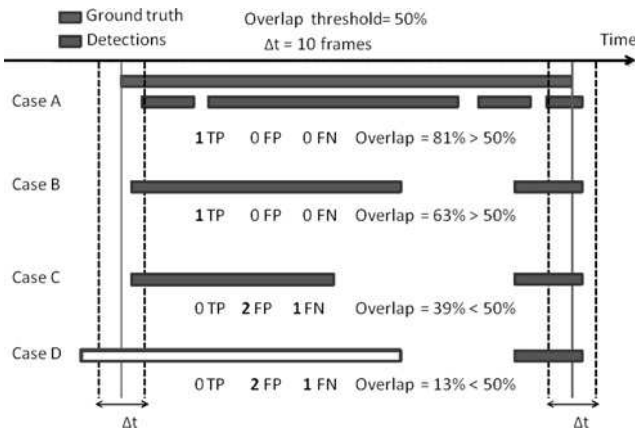
Fig. 8 *Sparse one-to-many temporal data association between ground truth and detections. Case A represents a typical situation in which a vehicle is detected sparsely as different tracks, with short miss-detection periods. The overlap threshold is set up to define how permissive is the evaluation with these holes. An extreme but still acceptable case of correct detection is Case B, where the overlap is above the set threshold. Finally, not valid detections are exemplified in cases C and D, where lower event overlap has been detected. In case D, in addition, the first detected event has been considered as unmatched, due to its extension beyond the time margin*

**Table 4** Object-level (i.e. spatio-temporal similarity) results

	R	P	F	# TP	# FP	# FN
day	0.667	0.286	0.400	6	15	3
rainy	0.857	0.343	0.490	12	23	2
night	0.765	0.464	0.578	13	15	4

Table 5 Level (i.e. spatial similarity) results

	R	P	F	# TP	# FP	# FN
day	0.717	0.740	0.728	2013	707	795
rainy	0.556	0.690	0.615	1006	453	804
night	0.269	0.302	0.284	159	368	433

a one-to-many data association procedure (illustrated in Fig. 8) which allow associating a single, for example, long-term, ground truth object with a number of sparse, brief, detected objects.

We also provide traditional frame-level analysis, where the time information is ignored and only the spatial correlation (i.e. overlapping) between ground truth and detections are taken into account. As in the previous example, a vehicle appearing in 1000 frames can generate 1000 TP if correctly detected, and 1000 FN if not, and as many FP as unmatched detections. The obtained results are shown in Tables 4 and 5, in the form of recall ($R = $ TP/(TP+FN)), precision ($P = $ TP/(TP+FN)) and f-measure ($F = 2(PR/(P+R))$), ranging from 0 to 1:
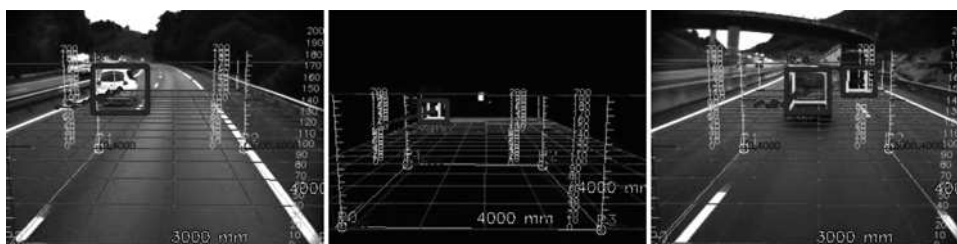
The object-level analysis reveals a significant number of FPs, which relates mainly to short detections occurring during a few frames and then disappear. Actually, most FPs correspond to extreme lateral positions where curbs and guardrails may appear and tend to generate confusing visual patterns for the classifiers. That said, we trained our vehicle detection model with a limited-size dataset (available at the UPM website: http://www.gti.ssr.upm.es/~jal/) and better performance shall be obtained when training the models with larger datasets. In any case, such detections can be easily removed with additional restrictions applied to the detections, if desired (such as time consistency, perspective, etc.). The frame-level analysis shows a better overall performance because the temporal correlation between objects is not taken into consideration. As we can see, the best performance is achieved at day time, where the system reaches recalls of about 71% (Fig. 9 shows some example images of the detected vehicles). For the two other types of sequences the performance is worst mainly due to the presence of wipers, reflections on the wet pavement, or bright spots in the night sequences.

5 Discussion

This paper proposes a design and development methodology in the form of an iterative cycle that creates optimised prototypes in short time improving certain aspects of the algorithms attending to a set of identified best practices. This methodology is adequate to address the two main challenges of ADAS development: (i) algorithm performance optimisation, and (ii) development cycle optimisation. There are many works in the literature that present vision-based ADAS algorithms. However, they focus mainly on analysing the effectiveness of their algorithms, and they do not pay special attention to how these algorithms can reach market. To the best of our knowledge, our work is the first in the literature that presents an effective and practical methodology designed for the general case of developing vision-based ADAS targeting both short TTM and algorithm performance optimisation.

The paper has also demonstrated the applicability of the proposed methodology in a case study centred on vehicle detection. The case study has shown how a vehicle detection algorithm can be optimised iteratively without the need to migrate code parts to special-purpose HW. Instead of devoting effort to trying to exploit parallelisation capabilities or complex code optimisation techniques, we consider it better to go back to the design of the algorithm itself to address bottlenecks that have been discovered after profiling in the target platform. Once the algorithm has been iteratively optimised following this method, other optimisation options, such as porting code to GPU, can be considered.

The proposed ADAS application has been successfully implemented in a standard automotive platform formed by an in-vehicle computer with Tizen IVI OS. The finally implemented algorithm was able to achieve real-time performance. In our implementation methodology, we first developed and tested the applications on a desktop PC. Then, we recompiled all the code in the target platform. This process can be straightforward if some considerations are followed such as using only standard C++ or not using any dependency that cannot be installed in the target platform. In our case, we have used some cross-platform libraries

Fig. 9 *Example output images of detected and tracked vehicles in different situations*

that can be downloaded directly from Tizen IVI repository. This repository is publicly available and contains all the essential packages for developing ADAS applications.

The use of an open source OS has important advantages compared with proprietary solutions. First, the development cost is lower, as there is no need to pay licensing fees. Furthermore, the platform is publicly assessed and rated by different partners and potentially also by academic institutions. Consequently, there is greater confidence in 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.

6 Conclusions and future work

The introduction of computer vision into the ADAS market is subject to a number of challenges, associated to the adaptation of this powerful technology to the restrictive conditions imposed by the industry. Optimisation becomes a necessity, and takes two main forms: algorithm optimisation to achieve real-time performance, and development cycle optimisation to allow rapid TTM.

In this paper we have exemplified our methodology to achieve these objectives in a use case centred in the well-known vehicle detection problem in monocular systems. We have shown that very rapid development can be achieved using open standards platforms and OS such as Tizen IVI. In addition, we focus on improving the algorithm rather than applying code optimisation, achieving speed up factors of up to ×15 with four subsequent prototypes where the algorithm itself was redesigned without any costly code optimisation or migration to parallelisation languages. Eventually, that migration can be carried out for a further performance improvement using open initiatives such as OpenCL that ensure interoperability in the medium and long-term.

7 Acknowledgments

8 References

1 Schneiderman, R.: 'Car makers see opportunities in infotainment, driver-assistance systems', *IEEE Signal Process. Mag.*, 2013, **30**, (1), pp. 11–15
2 Nieto, M., Ortega, J.D., Otaegui, O., *et al.*: 'Optimization of computer vision algorithms in codesign methodologies'. ITS World Congress 2014, Detroit, US, 7–11 September 2014
3 Velez, G., Nieto, M., Otaegui, O., *et al.*: 'Implementation of a computer vision based Advanced Driver Assistance System in Tizen IVI'. ITS World Congress 2014, Detroit, US, 7–11 September 2014
4 Lin, H.-Y., Chen, L.-Q., Lin, Y.-H., *et al.*: 'Lane departure and front collision warning using a single camera'. Int. Symp. on Intelligent Signal Processing and Communications Systems 2012 (ISPACS), pp. 64–69
5 Wu, B.-F., Huang, H.-Y., Chen, C.-J., *et al.*: 'A vision-based blind spot warning system for daytime and nighttime driver assistance', *Comput. Electr. Eng.*, 2013, **39**, (3), pp. 846–862
6 Turturici, M., Saponara, S., Fanucci, L., *et al.*: 'Low-power DSP system for real-time correction of fish-eye cameras in automotive driver assistance applications', *J. Real-Time Image Process.*, 2013, **9**, pp. 463–478
7 Malinowski, A., Yu, H.: 'Comparison of embedded system design for industrial applications', *IEEE Trans. Ind. Inf.*, 2011, **7**, (2), pp. 244–254
8 Darouich, M., Guyetant, S., Lavenier, D.: 'A reconfigurable disparity engine for stereovision in advanced driver assistance systems', *Lect. Notes Comput. Sci.*, 2010, **5992**, pp. 306–317
9 Mielke, M., Schafer, A., Bruck, R.: 'ASIC implementation of a gaussian pyramid for use in autonomous mobile robotics'. 2011 IEEE 54th Int. Midwest Symp. on Circuits and Systems (MWSCAS), 2011, pp. 1–4
10 Samarawickrama, M., Pasqual, A., Rodrigo, R.: 'FPGA-based compact and flexible architecture for real-time embedded vision systems'. Int. Conf. on Industrial and Information Systems (ICIIS), 2009, pp. 337–342
11 Wojcikowski, M., Zaglewski, R., Pankiewicz, B.: 'FPGA-based real-time implementation of detection algorithm for automatic traffic surveillance sensor network', *J. Signal Process. Syst.*, 2012, **68**, (1), pp. 1–18
12 Hiraiwa, J., Amano, H.: 'An FPGA implementation of reconfigurable real-time vision architecture'. 27th Int. Conf. on Advanced Information Networking and Applications Workshops (WAINA), 2013, pp. 150–155
13 Lee, S., Son, H., Choi, J.-C., *et al.*: 'High-performance hog feature extractor circuit for driver assistance system'. IEEE Int. Conf. on Consumer Electronics (ICCE), 2013, pp. 338–339
14 Souani, C., Faiedh, H., Besbes, K.: 'Efficient algorithm for automatic road sign recognition and its hardware implementation', *J. Real-Time Image Process.*, 2013, **9**, pp. 79–93
15 Stein, G., Rushinek, E., Hayun, G., *et al.*: 'A computer vision system on a chip: a case study from the automotive domain'. IEEE Computer Society Conf. on Computer Vision and Pattern Recognition – Workshops, 2005, pp. 130–130
16 Hsiao, P.-Y., Yeh, C.-W.: 'A portable real-time lane departure warning system based on embedded calculating technique'. IEEE 63rd Vehicular Technology Conf., 2006, vol. 6, pp. 2982–2986
17 Jeng, M.-J., Guo, C.-Y., Shiau, B.-C., *et al.*: 'Lane detection system based on software and hardware codesign'. 4th Int. Conf. on Autonomous Robots and Agents, 2009, pp. 319–323
18 Velez, G., Cortés, A., Nieto, M., *et al.*: 'A reconfigurable embedded vision system for advanced driver assistance', *J. Real-Time Image Process.*, 2014, doi: 10.1007/s11554-014-0412-3
19 'Automotive Grade Linux', available at http://automotive.linuxfoundation.org/, accessed December 2014
20 'Tizen IVI', available at https://wiki.tizen.org/wiki/IVI, accessed December 2014
21 'Android Auto', available at https://www.android.com/auto/, accessed December 2014
22 Dorst, K.: 'The core of 'design thinking' and its application', *Des. Stud.*, 2011, **32**, (6), pp. 521–532
23 Norman, D.A., Draper, S.W.: 'User centered system design. New Perspectives on Human-Computer Interaction' (L. Erlbaum Associates Inc., Hillsdale, NJ, 1986)
24 Löwgren, J.: 'Applying design methodology to software development'. 1st Conf. on Designing Interactive Systems: Processes, Practices, Methods and Techniques, 1995, pp. 87–95
25 Royce, W.W.: 'Managing the development of large software systems', *IEEE WESCON*, 1970, **26**, (8), pp. 328–338
26 'Manifesto for Agile Software Development', available at http://agilemanifesto.org/, accessed December 2014
27 Teich, J.: 'Hardware/software codesign: The past, the present, and predicting the future'. IEEE 100(Special Centennial Issue), 2012, pp. 1411–1430
28 Sun, Z., Bebis, G., Miller, R.: 'On-road vehicle detection using optical sensors: A review'. IEEE Proc. Int. Conf. Intelligent Transportation Systems, 2004, pp. 585–590
29 Ortega, J.D., Nieto, M., Cortés, A.: 'Perspective multiscale detection of vehicles for real-time forward collision avoidance systems'. Advanced Concepts for Intelligent Vision Systems, 2013 (*LNCS*, 8192), pp. 645–656
30 Arróspide, J., Salgado, L.: 'Log-Gabor filters for image-based vehicle verification', *IEEE Trans. Image Process.*, 2013, **22**, (6), pp. 2286–2295
31 Chang, W.-C., Cho, C.-W.: 'Online boosting for vehicle detection', *IEEE Trans. Syst. Man Cybern. B: Cybern.*, 2010, **40**, (3), pp. 892–902
32 Sivaraman, S., Trivedi, M.M.: 'Looking at vehicles on the road: a survey of vision-based vehicle detection, tracking, and behaviour analysis', *IEEE Trans. Intell. Transp. Syst.*, 2013, **14**, (4), pp. 1773–1795
33 'CMake', available at http://www.cmake.org/, accessed December 2014
34 Dalal, N.: 'Histograms of oriented gradients for human detection'. Proc. IEEE Computer Vision and Pattern Recognition, 2005, pp. 886–893
35 Nieto, M., Ortega, J.D., Cortes, A., *et al.*: 'Perspective multiscale detection and tracking of persons'. MMM 2014, Part II, 2014 (*LNCS*, 8326), pp. 92–103
36 Viola, P., Jones, M.: 'Robust real-time face detection', *Int. J. Comput. Vis.*, 2004, **57**, (2), pp. 137–154
37 Chan, Y.-M., Huang, S.-S., Fu, L.-C., *et al.*: 'Vehicle detection and tracking under various lighting conditions using a particle filter', *IET Intell. Transp. Syst.*, 2012, **6**, (1), pp. 1–8