Real-Time Video Analysis on an Embedded Smart Camera for Traffic Surveillance

Michael Bramberger, Josef Brunner, Bernhard Rinner
Institute for Technical Informatics
Graz University of Technology, AUSTRIA
{bramberger, brunner, rinner}@iti.tugraz.at

Helmut Schwabach
Video & Safety Technology
ARC seibersdorf research, AUSTRIA
helmut.schwabach@arcs.ac.at

Abstract

A smart camera combines video sensing, high-level video processing and communication within a single embedded device. Such cameras are key components in novel surveillance systems.

This paper reports on a prototyping development of a smart camera for traffic surveillance. We present its scalable architecture comprised of a CMOS sensor, digital signal processors (DSP), and a network processor. We further discuss the mapping of high-level video processing algorithms to embedded DSP-based platforms and identify typical pitfalls for the porting of software from desktops to embedded platforms. Our mapping strategies are demonstrated on an algorithm for automatic detection of stationary vehicles. This algorithm is migrated from a Matlab-based prototyping implementation to an embedded DSP implementation in our smart camera.

Our implemented smart camera prototype streams the video data over an IP-network to a central monitoring station and is able to detect stationary vehicles and blocking cargo on highways within the required real-time constraints of six seconds.

Keywords: *smart camera; traffic surveillance; embedded system digital signal processor; stationary vehicle detection;*

1. Introduction

Surveillance systems nowadays undergo a dramatic shift. Traditional surveillance systems of the 1st and 2nd generation employed essentially analog CCTV cameras to capture the monitored scenes. The captured data was then transferred to digital backend systems where some computation or storage took place. Current semiconductor technologies enable systems to leap forward to 3rd generation surveillance systems where the AD conversion and even highly sophisticated computation such as video compression take place in the video sensors (cp. Figure 1) [11]. *Smart cameras* [15] even extend the functionality of 3rd generation video sensor by providing on-board high-level video processing. Thus, novel surveillance systems are moving towards distributed, decentralized systems, which feature extended functionality, improved error-resistance as well as improved utilization of the provided infrastructure.

Clearly, traffic surveillance – especially highway surveillance – benefits greatly from this evolution. For example, the on-site and online computation of traffic parameters such as average speed or lane occupancy will improve the capabilities of traffic management systems; the automatic detection of dangerous situations such as accidents or wrong-way drivers will dramatically improve traffic security. 3rd generation surveillance systems, however, introduce additional (real-time) constraints to the components of the surveillance system. Especially, the smart camera must perform its high-level tasks within tight timing constraints and with limited resources for computation, memory and power.

This paper reports on the prototype development of a smart camera for traffic surveillance. The smart camera (i) captures a video stream using a sophisticated CMOS image sensor, (ii) performs high-level video analysis, e.g., average speed and stationary vehicle detection, (iii) compresses the video stream using MPEG-4, and (iv) transfers the compressed data via an IP-based network to a base station [2]. This smart camera is therefore a key component of novel surveillance systems.

The main contributions of this work include (i) the real-time implementation of customized high-level video algorithms on embedded platforms, (ii) the development of a flexible and scalable smart camera architecture, and (iii) the real-time demonstration of the automatic detection of stationary vehicles and blocking cargo on highways using our smart camera.

The migration of high-level video processing from general-purpose systems to embedded systems also asks for

Proceedings of the 10th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS'04)
1080-1812/04 $ 20.00 © 2004 IEEE

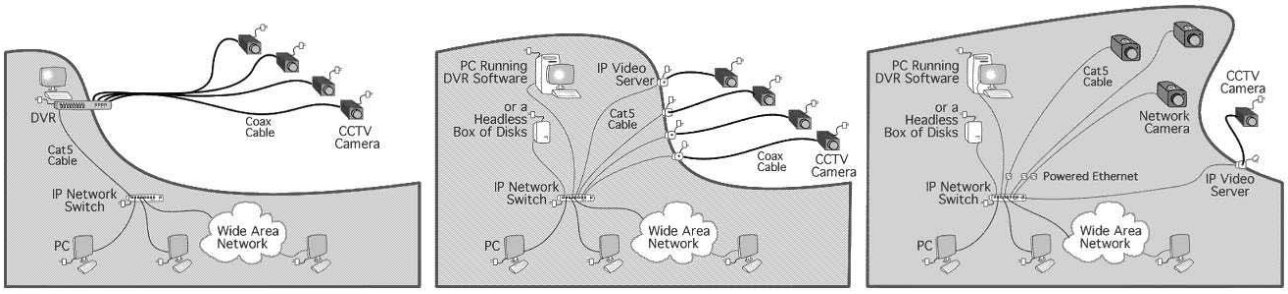

Figure 1. The evolution of surveillance systems from 1st to 3rd generation. In 3rd generation systems AD conversion and high-level computing such as video compression is performed in the video sensors.

different software development methods. Although the embedded platforms provide sufficient computing performance, efficiently developing and porting *efficient* software for these platforms is a difficult and tedious task. High-level video algorithms have been mostly prototyped and developed on general-purpose computers using languages such as Matlab or C++. However, a direct mapping of these algorithms to the embedded platform does not yield the required performance in most cases because the compiler and code generation tools are not able to exploit the hardware features of the embedded platform [8]. The use of optimized image libraries for the embedded platform considerably increases the performance. However, due to the complexity of the high-level algorithms the required performance may even not be met by using the (low-level) image libraries only. In these cases a mapping of the algorithm is required.

We identify typical pitfalls for the porting of software from desktop platforms to embedded architectures based on *digital signal processors (DSP)*, e.g., excessive memory access, inefficient data handling and data formats [7]. Our mapping strategies are demonstrated on an algorithm for automatic detection of stationary vehicles. This algorithm is migrated from a Matlab-based prototyping implementation to a DSP implementation in our smart camera.

The remainder of this paper is organized as follows. Section 2 presents the architecture of the smart camera and Section 3 briefly introduces the video analysis algorithm for the automatic detection of stationary vehicles and blocking cargo. Section 4 discusses typical pitfalls for the mapping to embedded DSP platforms. Section 5 describes the mapping and implementation of the stationary vehicle detection on our smart camera. Section 6 presents experimental results, and Section 7 discusses related and future work.

2. System Architecture

2.1. System Overview

Beside the functional requirements, traffic surveillance adds environmental requirements to smart cameras. Since smart cameras are deployed aside highways and in tunnels they are exposed to harsh environmental influences such as rapid changes in temperature and humidity as well as wind and rain. Additionally, the system's power dissipation has to be considered due to two reasons. (i) High power dissipation typically leads to excessive waste heat which requires active cooling to be dissipated. Since moving parts are subject to wear, active cooling is not desirable. (ii) Due to limited power resources at exposed locations, where the system may be fed by a solar panel, it is necessary to lower the power consumption as much as possible [1].

2.2. Hardware

As depicted in Figure 2, the smart camera is divided into three major parts: (i) the video sensor, (ii) the processing unit, and (iii) the communications unit.

2.2.1. Video Sensor The video sensor represents the first stage in the smart cameras overall data flow. The sensor captures incoming light and transforms it into electrical signals that can be transferred to the processing unit.

A CMOS sensor best fulfills the requirements for a video sensor. These sensors feature high dynamics due to their logarithmic characteristics and provide on-chip ADCs and amplifiers.

Our first prototype of the smart camera is equipped with the *LM-9618* CMOS sensor from *National Semiconductor*. Its specification is enlisted in Table 1.

2.2.2. Processing Unit The second stage in the overall data flow is the processing unit. Due to the high-performance on-board image and video processing the requirements on the computing performance are very high.

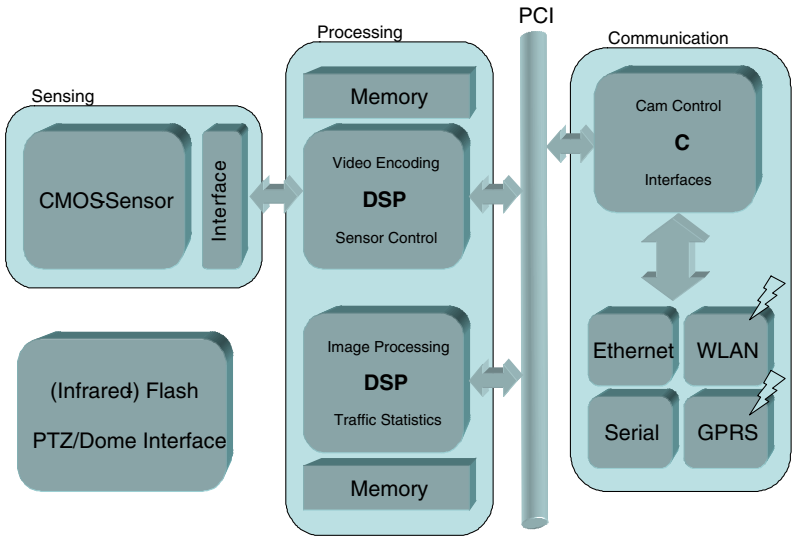

Figure 2. System architecture of the smart camera.

Dynamic range	Type	Resolution	Max. fps	ADC-Resolution	Sensor control
100 dB	Monochrome	640×480	30	12 bit	I^2C

Table 1. Video sensor specification.

A rough estimation results in 10 GIPS computing performance. These performance requirements together with the various constraints of the embedded system solution are fulfilled with *digital signal processors (DSP)*. The smart camera is equipped with two TMS320C6415 DSPs from Texas Instruments (TI) running at 600 MHz. Both DSPs are loosely coupled via the on-board PCI bus, while each processor is connected to its own local memory. The various tasks are statically mapped to the DSPs to avoid overhead induced by a global scheduler.

The video sensor is connected to one DSP via a FIFO memory to relax the timing between sensor and DSP. The image is then transferred into the DSP's external memory and via the PCI bus to the other components (DSP and network processor).

2.2.3. Communications Unit The final stage of the smart camera's overall data flow is represented by the communications unit. The unit is primarily composed of an *Intel XScale IXP425* processor, which directly manages most onboard communications like PCI, Ethernet, USB, and serial communications. Wireless LAN and GSM/GPRS are connected using a generic interface. This interface enables the connection of various peripherals or communication systems with low effort.

A second class of interfaces is also managed by the communication unit. Flashes, pan-tilt-zoom heads (PTZ), and domes are controlled using the communication unit. The moving parts (PTZ, dome) are typically controlled using serial interfaces like RS232 and RS422. Additional in/outputs are also provided, e.g., to trigger flashes or snapshots.

2.3. Software

The software architecture of the smart camera is basically divided into two parts:

DSPs are configured to basically run computation intensive tasks like video compression (MPEG-4 simple profile), image analysis, or traffic parameters calculation. Since reconfigurability and scalability are important issues, the DSPs are running on Texas Instruments' Reference Framework 5 (RF5) in combination with TI's XDAIS algorithm standard [13], which enables the exchange and reconfiguration of algorithms during runtime [12]. All reconfiguration and control actions are controlled by the system control processor as described below.

XScale processor is primarily used for system control and communication purposes. Therefore a standard operating system eases the development of internal and external communication services like web-services, proprietary control connections, or PCI-communications. Hence (Embedded-) Linux has been chosen to be used.

3. Stationary Vehicle Detection (SVD)

We demonstrate the mapping of high-level image analysis algorithms to an embedded DSP architecture on a specific video analysis algorithm, i.e., *stationary vehicle detection (SVD)* in tunnels [9]. The fundamental requirements for this algorithm are a stationary camera position and pretty static ambient light conditions – both requirements are typically fulfilled in tunnels. Thus, intensity changes are only caused by the motion of vehicles or by noise, e.g., reflections or lights of cars.

Intensity changes are exploited by the SVD algorithm. Gray-scale image frames (in 8 bit resolution) serve as input; the algorithm delivers image areas where a stationary vehicle has been detected as output. The basic idea of the algorithm is to maintain a background model of the observed scene, i.e., to identify regions where the intensity has not recently changed significantly. Thus, pixels not included in the background model represent the foreground, e.g., moving vehicles, of the observed scene. A stationary vehicle can, therefore, be detected when a sufficiently large foreground area has become a (stationary) background.

Figure 3 depicts the main steps of the SVD algorithm. These steps are repeated whenever a new frame is captured. The k least recently captured image frames I_i of size $n \times m$ are stored in a frame buffer. In the first step the statistics of the pixel's intensity is computed and stored in the *observation distribution (OD)* matrix of size $n \times m$. Each element of the OD matrix holds the mean value μ_{od} and standard deviation σ_{od} of the pixel's intensity distribution over the last k frames.

In the second step, the OD values are used to adapt the values of the *background model (BG)* which are stored in the BG matrix. The BG represents the long-term intensity distribution of each pixel. Each element of the BG matrix also holds values for the mean μ_{bg} and the standard deviation σ_{bg}. Note that the smaller the value of σ_{bg} the more confidence we have that the corresponding pixel belongs to the background of the image scene. Thus, the standard deviation of the OD and the BG are used to control the rate of the BG adaptation. The parameters of the background model are adapted according to the following equations:

$$\begin{aligned} \mu_{bg} &= (1-f)\cdot\mu_{bg} + f\cdot\mu_{od} \\ \sigma_{bg} &= (1-f)\cdot\sigma_{bg} + f\cdot\sigma_{od}. \end{aligned} \quad (1)$$

The update factor f is computed as

$$f = \frac{\alpha}{1 + e^{a\cdot(\sigma_{od}-\sigma_{bg})}} \quad (2)$$

where α defines an upper limit on f (typically 0.1) and a (typically 1.0) is a scaling factor. The BG matrix is initialized with large values for σ_{bg}. Thus, no pixel is associated with the background during initialization.

In the third step the algorithm identifies long-term intensity changes between the BG and the OD distribution. If the distributions represented by the corresponding elements in the BG and OD matrixes are significantly different, new background areas (pixels) have been identified. The result of this statistical test is stored in a binary image of size $n \times m$.

In the final step the algorithm searches for connected components in the binary image. If a connected component exceeds a predefined area a stationary vehicle has been identified.

Note that all except the final steps can be performed for each pixel independently. This data-independence is exploited in the mapping on our DSP architecture.

4. Algorithm Mapping

High-level image analysis algorithms are typically prototyped and tested using high-level languages such as Matlab or C++ on a desktop computer. However, such prototyping code has to be carefully mapped to a DSP architecture in order not to defeat the performance capabilities of such architectures. In this section we identify the main pitfalls in mapping a high-level algorithm to a DSP architecture and discuss strategies for a successful mapping.

4.1. External memory access

Excessive (and unnecessary) access to external memory is a major source for poor performance on embedded DSP architectures. In many high-level languages memory management is (by intention) hidden from the programmer, and complex and large data structures are used. While these features ease the algorithm development, they often result in many references to external memory. However, by a reorganization of the program and by exploiting data independence, access to external memory can be strongly reduced.

Consider for example the computation of the mean and standard deviation in the SVD algorithm. In Matlab, this is typically performed on matrices representing entire image frames. The computation of the statistics basically requires the following operations on the matrices `A` and `B` both of size $n \times m$.

	Loads	Stores
	$n \times m$ matrices	
`sum=A+B`	2	1
`avg=sum/count`	1	1
`sum2=A*A + B*B`	4	3
`std_dev=sqrt(sum2-sum*sum)`	4	3

When the entire matrix does not fit into the available internal memory (which is the case for many embedded DSP architectures), this code fragment requires 11 load and 8 store operations of $(n \times m)$-sized matrices. For image data

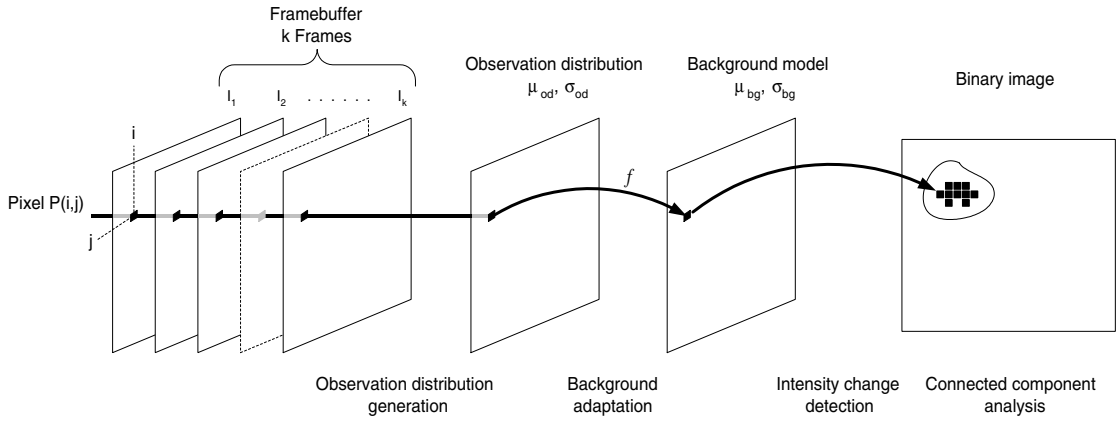

Figure 3. Main steps of the stationary vehicle detection.

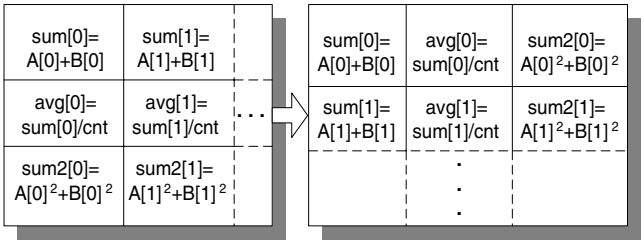

Figure 4. Image-based vs. pixel-based computation.

at full PAL resolution (720×576, 8-bit pixels), this results in a total of 7.12 MB of transferred data. The large number of matrix store and load operations is caused by storing and loading the intermediate results.

However, by migrating from an image-based to a pixel-based computation the number of external memory accesses can be largely reduced. In many image analysis algorithms the data dependency between individual pixels is small. In this case the successive steps of the algorithm can be applied on individual (blocks of) pixels, and the intermediate results can be stored in registers or internal memory. This migration from image-based to pixel-based computation can be viewed as a "transposition" (Figure 4). The columns in the left matrix correspond to the pixels; the rows correspond to the individual steps of the image algorithm. The matrix is processed in row major order. In pixel-based computation (Figure 4 right) the matrix is transposed but still processed in row major order.

	Loads	Stores
`sum[l]=A[l]+B[l]`	2	1
`avg[l]=sum[l]/count`	0	1
`sum2[l]=A[l]*A[l] + B[l]*B[l]`	0	1
`std_dev[l]=sqrt(sum2[l]- sum[l]*sum[l])`	0	1

The number of load and store operations has been decreased to two loads and four stores, whereas the stores are only necessary, if the intermediate results (sum, $sum2$) are needed for later computations. Since the code fragment is iterated $(n \times m)$-times, the transferred data is reduced to 2.47 MB (including the intermediate results). If the intermediate results are not used, the total memory transfer is 1.6 MB.

4.2. Data transfer

Most embedded architectures provide caches and direct memory access (DMA) to improve the memory transfer between the external memory and processor. While caching is transparent to the programmer DMA typically achieves an even higher performance than caching. A well-known and regular data access pattern is an important precondition for effective DMA. Many image processing algorithms fulfill this requirement. For these applications DMA transfer is often realized by double buffering using "ping-pong" buffers.

4.3. Number format issues

In high-level languages and on desktop computers the used number formats are typically no big issue. The situation is completely different on embedded platforms which are mostly comprised of fixed-point processors. On fixed-point processors, floating-point operations have to be realized in software and, therefore, dramatically degrade the

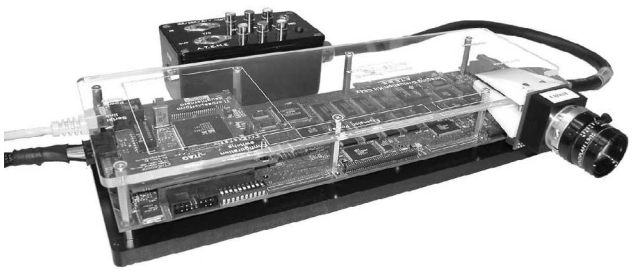

Figure 5. Prototype of the smart camera.

performance compared to fixed-point operations. It is desirable to implement integer versions of the algorithm on the embedded platform. However, this is not always possible.

A compromise is to transform floating-point numbers to a self-defined fixed-point format consisting of a non-standardized number of integer and fraction bits. The number of fraction bits can be typically limited for a specific application. The *Qm.n* notation is an example of a signed fixed-point number format with m integer bits, n fraction bits and a single sign bit. Thus, a *Qm.n* number requires $m + n + 1$ bits.

The number of fraction bits defines the precision of the number format. However, it is desirable to keep the total bit-count of the fixed-point numbers low due to the following reasons.

Memory is a crucial resource in embedded systems. Especially, internal memory has to be handled very carefully. For example, a 16-bit fixed-point number preserves expensive internal memory compared to a 32-bit number. Additionally, the amount of transferred data is reduced.

Parallelism can be improved by exploiting packed-data processing capabilities of current DSPs. In some 32-bit DSPs, such as the TMS320C64x from Texas Instruments, two 16-bit or four 8-bit results can be concurrently computed by a single 32-bit unit.

5. Implementation

The SVD algorithm has been implemented on a prototype of our smart camera (cp. Figure 5). The heart of our prototype is the *Network Video Development Kit - NVDK* from ATEME. The NVDK includes a Texas Instruments TMS320C6416 DSP with 264 MB of on-board memory and an Ethernet network extension card. We have extended the prototype platform by an additional extension board that connects a digital CMOS image sensor to the NVDK.

Starting point for the implementation on the smart camera was a SVD prototype in Matlab. We applied all mapping strategies discussed in Section 4 whereas all code transformations have been implemented in Matlab first to ease the test of our modifications. After functional testing in Matlab, the code was ported to C. No optimization in assembler has been applied.

Case	Approach	Cycles	Time	fps
1	Matlab	—	2900 ms	0.34
2	C++	4200 M	7000 ms	0.14
3a	C, no DMA	380 M	633 ms	1.57
3b	C, DMA	304 M	508 ms	1.9
3c	C, DMA, packed-data	250 M	416 ms	2.4

Table 2. Runtime and achieved frame rates of different mappings of the SVD algorithm.

The first three steps of the SVD algorithm (OD generation, BG adaption and intensity change detection) have been transposed to pixel-based computation. The connected component analysis is still applied on the entire image. The Matlab prototype has been realized in floating point arithmetic. For the implementation on the smart camera prototype the fixed-point format *Q9.6* has been chosen for the computation of the OD and BG statistics. Whether the range and precision of this number format is sufficient for the SVD has been previously checked in Matlab. As a consequence arithmetic operations such as multiplication, division, square root, and exponential function, have been implemented on the basis of the *Q9.6* number function. The image data transfer between external and internal memory has been realized by DMA using double buffering ("ping-pong" buffers). The size of the image blocks to be transferred can be configured.

6. Experimental Results

6.1. Computing Performance

Table 2 summarizes the achieved performance results for different implementations of the SVD algorithm. This table shows the cycles and required run time for processing a single full-resolution PAL image as well as the achieved frame rates.

Starting point for our experiments was a SVD algorithm prototyped in Matlab 6.1R12. This prototype required 2.9 s to process a single full-PAL resolution image on a standard 2.4 GHz Pentium 4 desktop computer (case 1).

On our smart camera prototype, we implemented two different versions of the SVD algorithm to evaluate the mentioned mapping and optimization techniques. This prototype is equipped with a 600 MHz TMS320C6416 DSP.

The first version (case 2) was a class-based C++ implementation directly derived from the SVD Matlab code. The matrix-oriented structure and the use of floating-point operations shortened the porting time to a few days. However,

Internal			External
code	stack	data	data
5 kB	1 kB	330 kB	17 MB

Table 3. Memory requirements of the SVD algorithm.

the achieved performance was very poor. The run time for a single frame has been more than doubled compared to the Matlab prototype. Excessive memory trashing and heavy use of the dynamic memory management due to the class-based structure are the main reasons for this poor performance. Even the introduction of a fixed-point number format did not improve the performance substantially.

The second approach boosted the performance by more than an order of magnitude. The mapping strategies in combination with caching (case 3a) results in a frame rate of 1.57 fps. DMA transfers (case 3b) further reduces the run time by 125 ms per frame. Thus, 18% of the computation time of case 3a are used for memory transfers. Since we use a 16-bit fixed-point format, almost half of the computation can be done in parallel by utilizing the packed-data capabilities of the TMS320C6416. This parallelization (case 3c) leads to a further performance gain of approximately 17%.

6.2. Memory Requirements

Table 3 summarizes the memory requirements of our SVD algorithm separated in internal and external memory. The code's footprint and the stack have a small size. Most part of the internal data memory is required for the "ping-pong" buffers for the DMA transfer. In our implementation, the block size for the DMA transfer can be configured. If 8 complete rows of the image are chosen as buffer size, the "ping-pong" buffers require 330 kB.

A total of 17 MB of the external memory is used to store the frame buffer and the various temporary matrices used for computation. The frame buffer utilizes 8 MB for storing 20 frames.

6.3. System Performance

The mentioned results only reflect the performance of the migrated algorithm alone. The prototype has been fitted with Ethernet access, networking stack, video acquisition and image conversion software to be able to run in a stand-alone mode.

These additional tasks result in an overall performance of 1.5 fps at full PAL resolution. However, the prototype is not fitted with a network processor, so the final system will achieve higher frame rates since the network processor will manage all communication issues. Again, the reduced memory bus utilization will yield in a higher performance gain than the reduced need for processing power.

6.4. Real-World Demonstration

A sample output of the stationary vehicle detection is depicted in Figure 6. The sequence shows the detection of a stationary vehicle after a car crash in a tunnel. The left image shows the situation immediately after the accident. Approximately 3 seconds after the crashed car stopped its movement, the the first parts of the crashed car are recognized as a potential stationary area (middle image). After additional 3 seconds, the system has detected the crashed car as a stationary item and an alarm is generated (right image). Note that the truck has not yet been detected as stationary since it has stopped its movement a few seconds later than the crashed car. According to the real-time constraints, the stationary vehicle has been detected within six seconds.

7. Discussion

In this paper we have presented a smart camera designed for use in an 3^{rd} generation traffic surveillance system. To be able to meet the real-time requirements of a traffic surveillance system while running high-sophisticated image analysis algorithms, some strategies for mapping high-level image analysis algorithms to DSP-based platforms have been presented. These strategies have been applied in the implementation of an algorithm for stationary vehicle detection on an embedded DSP-platform. Although the implementation has been limited to C-level optimizations, the performance has been improved by an order of magnitude. This improvement has enabled on-board and online detection of stationary vehicles at full PAL-resolution in our smart camera. It is important to recognize that dramatic performance improvements can be achieved by applying high-level strategies. Tedious hand-optimization of assembly code can be avoided.

In [14] Wolf and Kandemir identified the memory system as a crucial resource in embedded systems. Principles and an implementation example of assembly-level code optimizations for DSPs were proposed by Karadayi et al. [6], Hwang and Sung [4] and Pham-Ngoc et al. [10]. Smart cameras are also an emerging field of research. Wolf et al. [15] presented a PC-based smart camera system with high-level image processing. Regazzoni et al. [11] [3] identified smart cameras as building blocks for third generation surveillance systems.

Future research work includes (i) a further optimization of the SVD implementation (including the exploitation of instruction level parallelism), (ii) communication structures and principles for knowledge-transfers between smart cam-

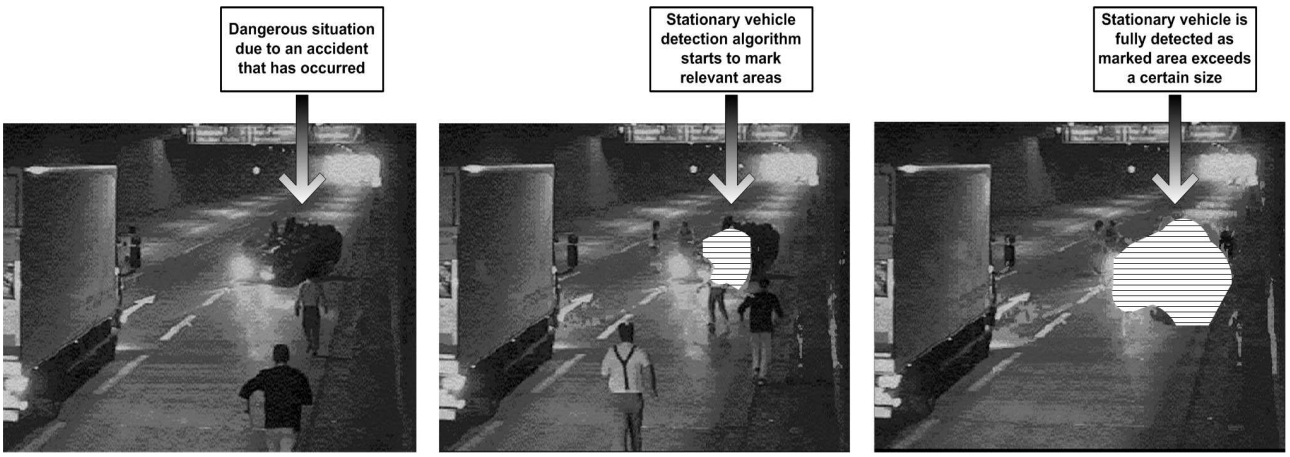

Figure 6. Example output from the stationary vehicle detection.

eras, based on multi-agent systems, to enable (iii) the mapping of different high-level video algorithms on our DSP-based smart camera including tracking and multi-camera analysis. Additionally, we are interested in (iv) an automation of the high-level algorithm mapping to embedded DSP-platforms [5], and the comparison of manual and automated mapping as well as the evaluation of recently announced tools for automated coding for embedded platforms, i.e., Mathworks' *Embedded Coder for Matlab/Simulink*.

References

[1] M. Bhardwaj, M. Rex, and A. Chandrakasan. Power-aware systems. In *Proceedings of the Asilomar Conference on Signals, Systems and Computers*, volume 2, pages 1695–1701, 2000.

[2] M. Bramberger, R. P. Pflugfelder, A. Maier, B. Rinner, B. Strobl, and H. Schwabach. A smart camera for traffic surveillance. In *Proceedings of the First Workshop on Intelligent Solutions in Embedded Systems*, pages 153–164, 2003.

[3] G. L. Foresti, P. Mähönen, and C. S. Regazzoni, editors. *Multimedia video-based surveillance systems*. Kluwer Academic Publishers, Boston, 2000.

[4] T. Hwang and W. Sung. Implementation of a digital copier using TMS320C6414 VLIW DSP processor. In *Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing*, 2003.

[5] M. Jungmann and M. Beine. Automatic code generation for safety-critical system development. *Embedded Control Europe*, pages 30–32, November 2003.

[6] K. Karadayi, V. Markandey, J. Golston, R. J. Gove, and Y. Kim. Strategies for mapping algorithms to mediaprocessors for high performance. *IEEE Micro*, 23(4), July-Aug 2003.

[7] D. Menard, D. Chillet, F. Charot, and O. Sentieys. Automatic floating-point to fixed-point conversion for DSP code generation. In *Proceedings of the International conference on compilers, architecture, and synthesis for embedded systems*, pages 270–276, 2002.

[8] K. Patel. Porting PC based algorithms to DSPs. *Embedded Edge*, pages 15–19, Fall 2003.

[9] R. P. Pflugfelder and H. Bischof. Learning spatiotemporal traffic behaviour and traffic patterns for unusual event detection. In *26th Workshop of the Austrian Association for Pattern Recognition*, pages 125–133, 2002.

[10] N. Pham-Ngoc, G. Lafruit, J.-Y. Mignolet, S. Vernalde, G. Deconinck, and R. Lauwereins. A framework for mapping scaleable networked multimedia applications on runtime reconfigurable platforms. In *Proceedings of the International Conference on Multimedia and Expo*, pages 469–472, 2003.

[11] C. S. Regazzoni, V. Ramesh, and G. L. Foresti. Introduction of the special issue. *Proceedings of the IEEE*, 89(10), Oct 2001.

[12] L. Sha. Upgrading real-time control software in the field. *Proceedings of the IEEE*, 91(7), July 2003.

[13] Texas Instruments. *TMS320 DSP Algorithm Standard Rules and Guidelines*, October 2002.

[14] W. Wolf and M. Kandemir. Memory system optimization of embedded systems. *Proceedings of the IEEE*, 91(1), Jan 2003.

[15] W. Wolf, B. Ozer, and T. Lv. Smart cameras as embedded systems. *IEEE Computer*, 35(9):48–53, September 2002.