Real-Time Imaging **6**, 313–324 (2000)
doi:10.1006/rtim.1999.0191, available online at http://www.idealibrary.com on IDEAL®

Architectural Issues on Vision-Based Automatic Vehicle Guidance: The Experience of the ARGO Project

This paper discusses the main architectural issues of a challenging application of real-time image processing: the vision-based automatic guidance of road vehicles. Two algorithms for lane detection and obstacle localization, currently implemented on the ARGO autonomous vehicle developed at the University of Parma, are used as examples to compare two different computing engines — a massively parallel special-purpose SIMD architecture and a general-purpose system — while future trends in this field are proposed, based on the experience of the ARGO project.

Alberto Broggi[1], Massimo Bertozzi[2] and Alessandra Fascioli[2]

[1]*Dipartimento di Informatica e Sistemistica, Università di Pavia, I-27100 Pavia, Italy, E-mail: broggi@dis.unipv.it*
[2]*Dipartimento di Ingegneria dell'Informazione, Università di Parma, I-43100 Parma, Italy, E-mail: {bertozzi, fascal}@CE.UniPR.IT*

Introduction

A great deal of different sensors exploiting different technologies, such as radars, lasers, sonars and bumpers, have been used on autonomous vehicles and robots in general to sense the surrounding environment, but passive sensors like cameras can offer prominent advantages such as the possibility of acquiring data in a non-invasive way, namely without altering the environment. In contrast, the use of active sensors involves the measure of the alteration of signals emitted by the sensors themselves, with the following two main advantages:

(1) first, they can measure quantities in a more direct way than vision. As an example, a Doppler radar can directly measure the relative movements between an object and the viewer, while vision can detect movement only as a result of a complex processing of image sequences;
(2) a second advantage is given by the considerably lower amount of acquired data, thus requiring less performing computing engines.

Active sensors are extremely powerful, but there are applications in which only vision can be successfully employed. In many indoor applications, such as the navigation of autonomous robots in unknown settings, vision can be greatly helped by active sensors in the task of recognition of objects, detection of the free-space, or checking for some specific objects' characteristics. Unfortunately, in case more than one robot is moving into the same environment, their active sensors may interfere with each other and cause problems. The problem gets even greater in an outdoor unstructured environment, in which a large number of vehicles could

1077-2014/00/080313+12 $35.00/0

be moving simultaneously. Hence, in the cases in which a massive and wide-spread use of autonomous sensing agents is envisaged, the use of passive sensors, such as cameras, is greatly preferred to invasive methods of perceiving the environment, which could lead to an unacceptable pollution of the environment. These are cases in which vision becomes of paramount importance.

The task of automatic vehicle guidance in outdoor environments must face other key problems intrinsic to the use of vision.

(1) Contrary to indoor settings, outdoor environments *cannot rely on structured information*. This is particularly true in the automotive field, where vehicles can move along roads with different characteristics, and the integration of specific infrastructures would be extremely expensive and prohibitive: an automatic vehicle should be able to navigate using standard road signs, without requiring further additional infrastructures.
(2) More than other applications, automatic vehicle guidance *requires fast processing*, since the maximum vehicle speed is proportional to the processing rate (for example, a processing rate of 25 frames/s allows to provide the control signals for autonomous steering every 40 ms, which is equivalent to one refinement on the steering wheel position for every meter when the vehicle drives at 100 km/h). For this purpose, the main problem, intrinsic to the processing of images, is the large amount of data, and thus of computation involved. Therefore, specific computer architectures and processing techniques must be considered in order to achieve real-time performance. Nevertheless, since the success of such automatic systems is tightly related to their cost, the computing engines cannot be based on expensive processors. Thus, either off-the-shelf components or *ad hoc* dedicated low-cost solutions must be considered.

In addition, the processing of images acquired by a camera installed on a moving vehicle in a non-structured outdoor environment suffers from some other major problems. Contrary to indoor applications, where robots move in controlled environments, the automotive field no assumptions can be made on key parameters, such as, for example, the illumination or the contrast of the scene, which are directly measured by the vision sensor. Hence, the subsequent processings must be robust enough to tolerate their changes and to adapt both to different road conditions, such as sun (high brightness and contrast due to shadows), rain (extremely high contrast due to reflections), fog (low contrast), and to their dynamic changes, such as transitions from sun to shadows or the entrance in a tunnel. Other key problems, such as the robustness to camera movements and to drifts in its calibration must be addressed as well.

This paper is organized as follows: the second section addresses the problem of automatic vehicle guidance and presents the solutions implemented on the ARGO autonomous test vehicle; the third section describes the architectural issues that were considered in the selection of the computing engine; and the final section concludes the paper with a discussion on future trends.

Automatic Vehicle Guidance

Although extremely complex and highly demanding, computer vision is a powerful mean to sense the environment and has been widely employed to address a large number of tasks for automatic vehicle guidance [7], ranging from *Road Following*, to *Platooning* (the automatic following of a manually driven vehicle by an automatic one), from *Vehicle Overtaking*, to *Automatic Parking*. To accomplish these tasks different quantities must be measured and/or patterns recognized before the closing of the control loop, such as:

(1) the relative position of the vehicle with respect to the lane, the check for obstacles on the path or for known road signs for *Road Following*;
(2) the recognition of a specific vehicle's characteristics and the computation of the time-to-impact for *Platooning*;
(3) the sensing of multiple lanes as well as obstacle detection for *Vehicle Overtaking*;
(4) the distance among already parked vehicles and the computation of the free-space for *Automatic Parking*.

Among all, the most complex and challenging task that has received the most insightful attention is indeed *Road Following* since it comprehends several basic functionalities such as Lane Detection and Obstacle Detection. A more complete survey of the different approaches developed worldwide can be found in [3].

The following subsections present a brief overview of the two algorithms that are currently implemented on ARGO, a Lancia Thema passenger car (see Figure 1)

Figure 1. The experimental vehicle ARGO.

used as a test vehicle. ARGO is equipped with a pair of synchronized stereo cameras to sense the environment and an odometer to measure the vehicle speed. These data are processed in order to detect obstacles and localize the lane ahead of the vehicle. A monitor, a led-based control panel, and a pair of stereo speaker are used to warn the driver with optic and acoustic signals in case of dangerous situations. Moreover the vehicle is equipped with autonomous steering capabilities: the system is able to follow the lane automatically by issuing commands to an actuator mounted on the steering column. The driver can interact with the system through a control panel, an emergency pedal, a joystick, and a keyboard (see Figure 2).

Lane detection

In most prototype autonomous vehicles developed worldwide *Road Following* is based on Lane Detection: first the relative position of the vehicle with respect to the lane is computed and then actuators are driven to keep the vehicle in a safe position.

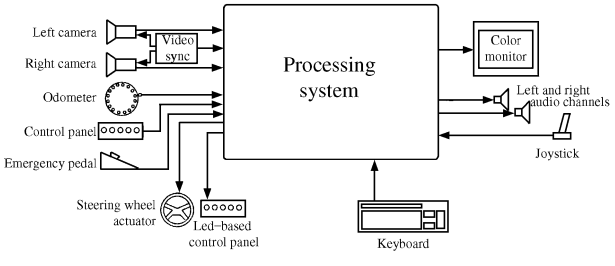

Figure 2. The computing architecture and equipment.

Thanks to the knowledge of the acquisition system setup and to a hypothesis of a flat road in front of the vehicle, from the captured image (Figure 3(a)) it is possible to generate a new image (Figure 3(b)) in which the perspective effect has been removed. In this image lane markings can be devised as almost *vertical* bright lines of *constant* width, surrounded by a darker region. In this case the pixels belonging to a road marking have a brightness value higher than their horizontal left and right neighbors at a given distance. Thus, the first phase of lane detection is based on the search for dark–bright–dark horizontal patterns with a specific size. The brightness value of every pixel is compared to that of its left and right horizontal neighbors and a new gray-level image is computed, which encodes the horizontal brightness transitions and thus the presence of lane markings.

Different illumination conditions, such as shadows or sunny blobs, cause road markings to have different brightness values; the pixels representing road markings maintain a brightness value higher than their horizontal neighbors. Thus, the image is enhanced taking advantage of its vertical correlation, and then binarized using an adaptive threshold (Figure 3(c)).

This image is horizontally scanned line by line starting from the bottom in order to build *chains* of non-zero pixels; when a non-zero pixel is found, the following actions are taken: if the distance between the pixel and the nearest extremum of a chain is less than a given threshold, the pixel is assigned to the chain, otherwise a new chain, initially formed by this pixel only, is started. Finally the chains are segmented and approximated by polylines (Figure 3(d)). When two polylines lie in similar directions and are sufficiently close to each other, they are joined, thus filling the gaps produced by either occlusions due to obstacles or an ineffective low-level processing; also dashed lane markings are converted into continuous lines (Figure 3(e)).

Every polyline is evaluated and the one which represents the center line with the highest confidence is selected. Figure 3(f) shows the output of the processing: a black line highlights the computed center line.

Finally, thanks to the knowledge of the vision system setup and to the flat road hypothesis, the spatial relationship between pixels of the acquired image and the road can be computed, thus allowing to estimate both the road geometry and the vehicle position with respect to the lane.

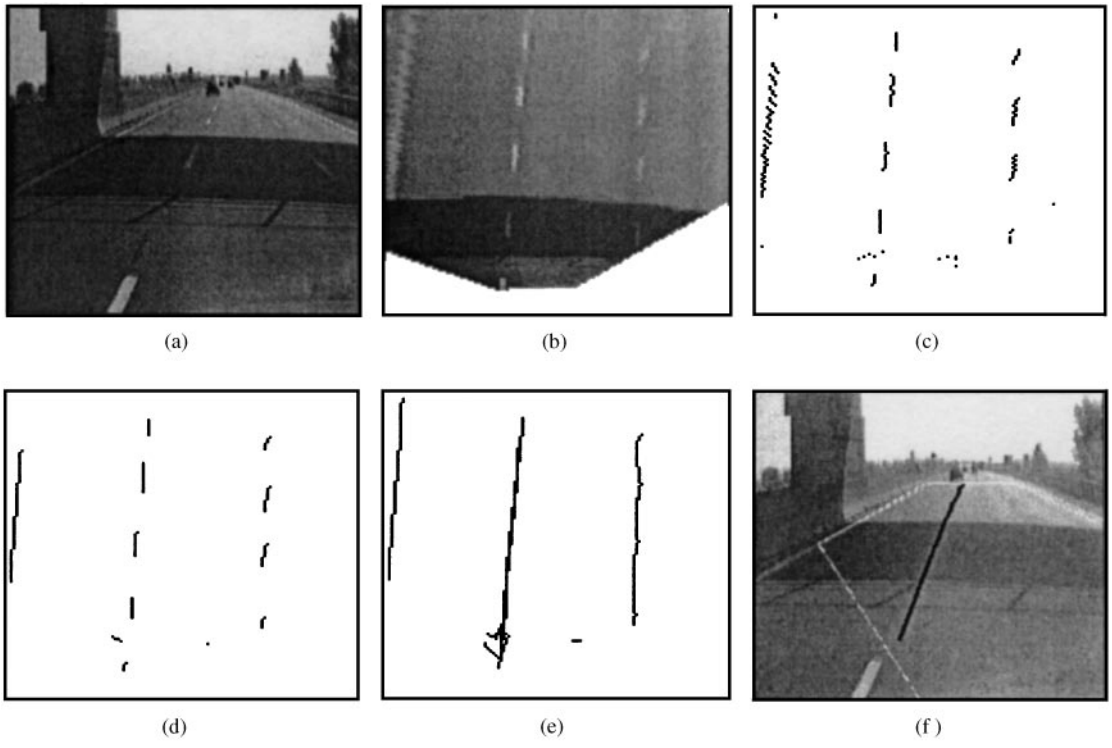

Figure 3. Lane Detection steps: (a) the captured image; (b) after the removal of the perspective effect; (c) pixels with darker horizontal neighbors; (d) chains of non-zero pixels; (e) the joined polylines; and (f) the computed center line superimposed in black onto a brighter version of the acquired image.

Obstacle detection

The removal of the perspective effect from a pair of stereo images allows to obtain two images (*remapped images*, see Figure 4(a) and 4(b)) whose analysis is used to localize potential obstacles: any difference between the two remapped images represents a deviation from the starting hypothesis of flat road and thus identifies a potential obstacle, namely anything rising up from the road surface.

An obstacle is detected when the image obtained as the difference between the two remapped images (Figure 4(c)) presents sufficiently large clusters of non-zero pixels that have a specific shape. Thus, the low-level portion of the processing consists of the computation of

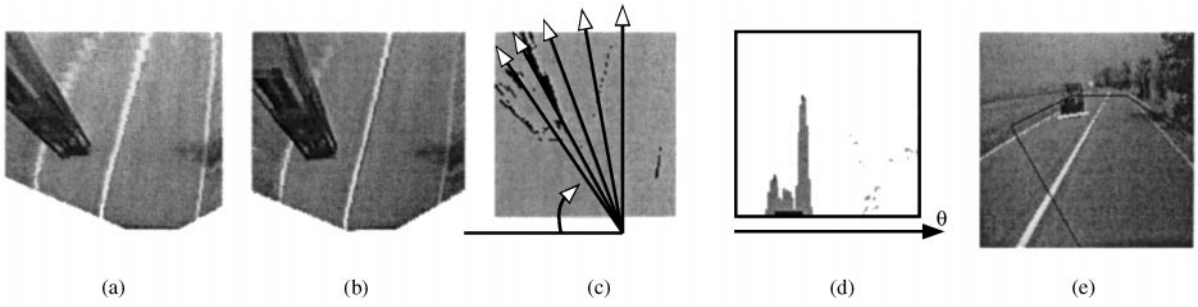

Figure 4. Obstacle Detection steps: (a) the left remapped image; (b) the right remapped image; (c) the difference image and the angles of view; (d) the polar histogram; (e) the obstacle detected shown by a white marker superimposed onto a brighter version of the left acquired image.

the difference between the remapped images, an adaptive binarization of the resulting image and a simple morphological filter aimed at the removal of small-sized details.

Because of the different angles of view of the stereo system, the vertical edges of an obstacle generate two triangles in the difference image. Unfortunately, due to their texture, irregular shape, and non-homogeneous brightness, real obstacles produce triangles that are not so clearly defined; nevertheless in the difference image some clusters of pixels with a quasi-triangular shape are recognizable (Figure 4(c)). The obstacle detection process thus consists of the localization of pairs of these *triangles*.

The medium-level processing consists of scanning the difference image in order to produce a *polar histogram* (Figure 4(d)). The polar histogram presents appreciable peaks that correspond to each triangle; the position of these peaks within the histogram determines the angle of view under which the obstacle is seen. Since in the difference image the presence of an obstacle produces two disjoint triangles that correspond to the vertical obstacles' edges, obstacle detection is reduced to the search for *pairs* of adjacent peaks.

Architectural Issues

Two main issues helped in wide-spreading the popularity of vision as a means to sensing the environment; both are tightly coupled to recent advances of the technology. First of all, technology is playing a basic role in the development of sensors. Current cameras include new important features that allow to address and solve some basic problems directly at the sensor level: image stabilization can now be performed during image acquisition, while the extension of camera dynamics allows to remove the processing required to adapt the acquisition parameters to the specific light conditions. The resolution of the sensors has been dramatically enhanced, and, in order to decrease the acquisition and transfer time, new technological solutions, such as CMOS cameras, have been considered. Their prominent advantages are that pixels can be addressed independently as in traditional memories, and that their integration on the processing chip seems to be straightforward.

In addition to this, new technological solutions, such as a higher integration and the reduction of the power supply voltage, allow to have machines that can deliver a high computational power, along with extremely fast internet-working facilities, at an affordable price. Since the early stages of vision (low-level image processing) are computationally demanding, the availability of low-cost engines helps to solve the basic bottlenecks that were originally preventing the promotion of vision as a common way to sense the environment.

For this reason a number of different *ad hoc* computer systems have been conceived and implemented, which exploit the characteristics of the various applications [11]. At the same time, current technology allows to have SIMD-like processing paradigms even in general-purpose processors, such as the new generation of Intel processors that include multimedia extensions (MMX).

In the two following subsections we analyse the two architectural solutions for the computing engine that were considered and implemented on the ARGO vehicle for automatic vehicle guidance tasks: special-purpose vs. general-purpose.

Special purpose system

According to the above considerations, a massively-parallel SIMD architecture, PAPRICA-3 [10] (PArallel PRocessor for Image Checking and Analysis, Version 3), has been considered as a hardware support to low-level image processing.

As shown in Figure 5, the core of the system is a dedicated SIMD cellular architecture based on a linear array [18] of Q identical 1-bit Processing Elements

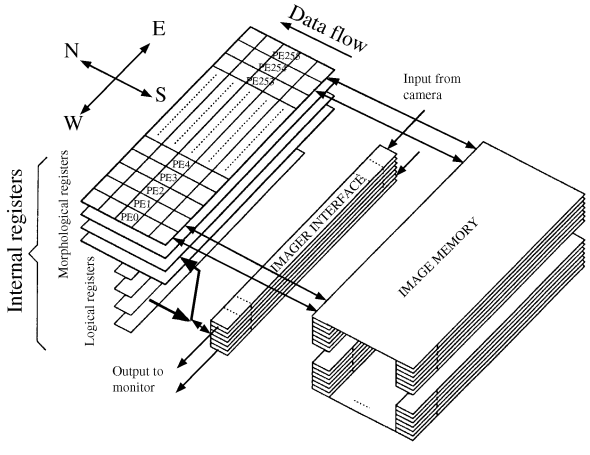

Figure 5. Structure of a 256 PEs Processor Array.

(PEs). The array as a whole is connected to an external image memory via a bi-directional Q-bit data bus; therefore each memory read or write operation transfers a complete vector of Q pixels at a time, 1 bit per pixel. This specific organization of the data-bus (Q bits wide and 1 bit deep) allows to reach a hardware efficiency higher than in the first prototype architecture, since the whole set of data transferred within a single cycle is generally completely significant.

The rationale behind this system is that the size of the PA matches exactly the width of the input image. This solution reduces the PE virtualization mechanism problem, which has been proven to be a critical design issue. the PA processes one full image line per machine cycle, whose duration ranges from 10 to 40 ns, depending on the specific instruction flow. Data are transferred into the PEs' internal registers, processed, and explicitly stored back again into the external memory according to a RISC-oriented processing paradigm. Since the instruction set is based on morphological operators, the result of an operation depends, for each PE, on the values of pixels in a given neighborhood (5×5). Data from **EAST** and **WEST** directions may be obtained by direct connection with neighboring PEs, while all other directions correspond to data of previous (**N, NE, NW**) or future (**S, SE, SW**) data lines. For this reason a number of processor registers (*Morphological Registers*, *MOR*) have a structure which is more complex than that of a simple memory cell and are actually composed of 51-bit cells with a **S→N** shift register connection. Besides the MOR, each PE owns also a number of *Logical Registers* (*LOR*) which may be used for intermediate storage and for all operations which do not require the use of neighboring values.

The system comprehends also a serial-to-parallel I/O device, called *Imager Interface*, connected to a conventional camera. While a line is processed by the PA, the Imager Interface automatically loads the following image line from the camera. At the end of the processing, the PA stores in parallel the results back into the Imager Interface (on different bit-planes) and loads in parallel the following image line. During the data acquisition process, the Imager Interface behaves like a shift-register, serially loading the data from the camera, and serially outputting the processed data to a monitor.

An interprocessor communication mechanism has been included to exchange information among PEs which are not directly connected: the Interprocessor Communication Network (ICN), shown in Figure 6. It allows global and multiple communications among components of different subsets of the PA (clusters of adjacent PEs) and its interconnection topology is fully and dynamically programmable: each PE drives a switch that enables or disables the communication (in wired-or) between itself and its left neighbor; the PEs can thus be dynamically grouped into clusters in which each PE can broadcast its value to the whole cluster within a single instruction.

In general, dedicated hardware systems require the development of specific programming languages and environments. In the case of the PAPRICA-3 architecture, a complex environment has been built to ease the prototyping of real-time applications: algorithms are implemented in C++ using high-level data types and the corresponding assembly code is automatically created by a code generator; in addition, a stochastic code optimizer takes the generated assembly code and improves it according to a genetic approach [4].

General purpose system

An alternative architectural solution which is currently under evaluation on the ARGO vehicle is based on a standard 200 MHz MMX Pentium processor.

MMX technology represents an enhancement of the Intel processor family, adding instructions, registers, and data types specifically designed for multimedia data processing. The MMX technology provides new features to the Intel processors family but at the same time maintains backward compatibility with all non-MMX operating systems and applications developed for the Intel platform, namely all existing software and operating systems developed for the Intel architecture do not need modifications to run on an MMX processor even in the presence of updated software that exploits MMX features.

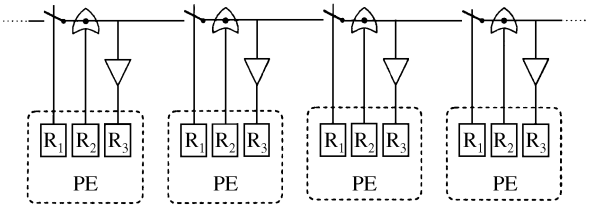

Figure 6. Interprocessor Communication Network (ICN): for each PE, Register R_1 drives its ICN switch; Register R_2 sends its value over the network; and Register R_3 collects the resulting value from the network.

On the other hand, software performance can be boosted by exploiting a SIMD technique: multiple data elements can be processed in parallel using a single instruction. The new general-purpose instructions supported by MMX technology perform arithmetic and logical operations on multiple data elements packed into 64-bit quantities. These instructions accelerate the performance of applications based on compute-intensive algorithms that perform localized recurring operations on small native data. More specifically in the processing of gray-level images, data is represented in 8-bit quantities, hence an MMX instruction can operate on 8 pixels simultaneously.

Basically the MMX extensions provide the programmers with the following new features:

MMX registers
The MMX technology provides 8 general-purpose 64-bit new registers. The main problem the Intel architects had to face was the backward compatibility with the existing software and specifically with the multi-tasking operating systems. In fact in a multi-tasking environment a context switch operation requires to save or restore the CPU status and register the contents. Nevertheless existing operating systems do not know the presence of MMX registers. In addition, different operating systems take different approaches for the state saving and restoring [20, 6]. To cope with these problems MMX registers have been overlapped to the floating point registers. Hence, when floating point registers are saved or restored also MMX registers are saved or restored, thus allowing existing operating systems to run without modifications.
On the other hand this solution has two drawbacks:

(1) the programmer is expected not to mix MMX instructions and floating point code in any way, but is forced to use a specific instruction (EMMS) at the end of every MMX enhanced routine. The EMMS instruction empties the floating point tag word, thus allowing the correct execution of floating point operations;
(2) frequent transitions between the MMX and floating-point instructions may cause significant performance degradation.

MMX data types
The MMX instructions can handle four different 64-bit data types (see Figure 7):

(1) 8 bytes packed into one 64-bit quantity;
(2) 4 words packed into one 64-bit quantity;

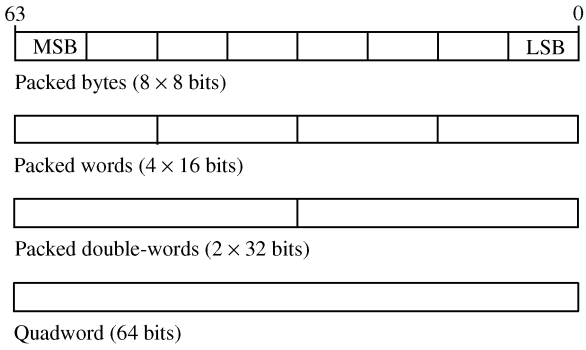

Figure 7. The MMX data types, their ordering, the most significant byte (MSB), and the least significant byte (LSB).

(3) 2 double-words packed into one 64-bit quantity; or
(4) 1 quadword (64-bit).

This allows to process multiple data using a single instruction or to directly manage 64-bit data.

MMX arithmetics
The main innovation of the MMX technology consists of two different methods used to process the data:

(1) saturation arithmetic; and
(2) wraparound mode.

Their difference depends on how the overflow or underflow caused by mathematical operations is managed. In both cases MMX instructions do not generate exceptions nor set flags, but in wraparound mode, it results that overflow or underflow are truncated and only the least significant part of the result is returned; conversely, the saturation approach consists in setting the result of an operation that overflows to the maximum value of the range, as well as the result of an operation that underflows is set to the minimum value. For example, packed unsigned bytes for results that overflow or underflow are saturated to $0 \times FF$ or to 0×00, respectively.

The latter approach is very useful for gray-level image processing, in fact, saturation brings the gray value to pure black or pure white, without allowing for an inversion as in the former approach.

MMX instructions
MMX processors are featured by 57 new instructions, which may be grouped into the following functional categories: arithmetic instructions, comparison instructions, conversion instructions, logical instructions, shift

instructions, data transfer instructions, and the EMMS instruction. All MMX instructions, except the EMMS one, have two operands: source (the left one) and destination (the right one). The source operand for all the MMX instructions (except for the data transfer instructions), can reside either in memory or in an MMX register, while the destination operand must reside in an MMX register.

Algorithms implementation

The low-level portion of the algorithms described in obstacle and lane detection have been implemented and tested on both the PAPRICA-3 and the Pentium MMX architectures. the PAPRICA-3 assembly code has been generated using the code generator starting from a C++ source, while the assembly code for the MMX implementation has been written directly by hand: since upto now there is no standard C compiler that handles MMX extensions, whereas, in case of a future availability, probably an *automatic parallelizing* compiler would not generate a fully optimized code unless some *ad hoc* C statements are introduced to help the parallelization.

According to the analysis in [1] the two following categories of operations have been considered:

(1) *Pointwise operations*: the new status of each image pixel is computed as a function of the previous values of the same pixel; they are used for simple operations, such as thresholding or image difference.
(2) *Cellular Automata operations*: the new status of each pixel is computed as a function of the values of the pixel's neighbors; they are used for operations such as morphological filters or adaptive thresholding.

For the sake of simplicity and for comparison purposes, the following two examples, one for each above category, are considered: absolute difference between two images (pointwise operation used for obstacle detection), and brightness comparison in a horizontal neighborhood (cellular automation operation used for lane detection).

Since these operations are performed on 8-bit gray-level images; the MMX processor handles 8 pixels simultaneously, while PAPRICA-3 operates on a single bit of every pixel of a full image line at the same time.

Absolute difference between two images
A pixelwise comparison of two images is performed. Every pixel of the resulting image is obtained computing the absolute difference between the corresponding pixels of the two images. In the case of the MMX-based processor, naming a and b the two images, the following two unsigned differences $ud_1 = a - b$ and $ud_2 = b - a$ are computed. Thanks to the saturation arithmetic ud_1 is null where $a < b$ as well as $ud_2 = 0$ where $a > b$, thus the absolute value $ud = |a - b|$ can be computed as $ud = ud_1 \vee ud_2$. Conversely, the PAPRICA-3 implementation is based on a single signed difference ($sd = a - b$) that gives a two's complement 9-bit result, whose ninth bit (s) is the sign. The final result $ud = |a - b|$ is computed as the eight less significant bits of $\overline{sd} + 1$ in case s is set, and as the eight less significant bits of sd otherwise.

Brightness comparison in a horizontal neighborhood
The value of every pixel is computed as a function of its value and the values of its left and right neighbors at a distance of 2. In the MMX implementation, multiple accesses to the memory are required; there are two different approaches to this problem:

(1) using a single access to the memory, the pixel value as well as the values of its neighbors are loaded into a single MMX register; in order to perform the comparison several shift operations must be used to align the pixels with its second neighbors. Unfortunately these shift operations reduce the number of pixel that can be processed simultaneously for each memory access from 8 to 4 (Figure 8(a)). Thus, two read accesses to the memory are required for the processing of 8 pixels;
(2) conversely, using three accesses to the memory, the pixel value and the values of its neighbors can be loaded in three different MMX registers in such a way that they have the correct alignment. In this case, 8 pixels can be processed simultaneously and there is no need for shift operations (Figure 8(b)).

On the other hand, in the PAPRICA-3 architecture, the PEs can directly access data held in their neighborhood, thus it is possible to process one full line after a single load operation. Moreover, in case the cellular automation is based on a wider neighborhood, PAPRICA-3 has no additional overhead, while the MMX implementation requires a sufficiently large number of registers and memory accesses.

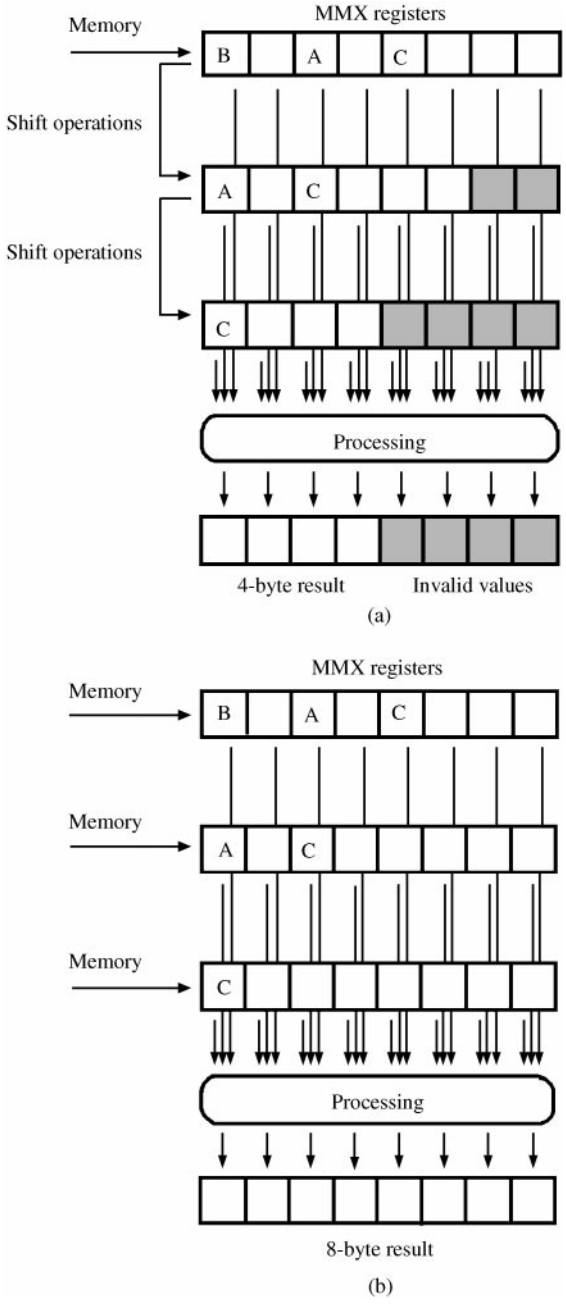

Figure 8. Two different approaches to a cellular automation operation in the MMX technology involving 8-bit pixels A, B, and C: (a) a single access to the memory followed by shift operations; (b) three partially overlapped accesses to memory.

More generally, when considering pointwise operations on 8-bit deep images, the MMX-based processor and PAPRICA-3 architecture have similar behavior, the only differences being the instruction sets and the number of pixels that can be processed simultaneously. Nevertheless, since PAPRICA-3 is a single bit processor, it can also efficiently handle images with a generic depth, while the MMX instruction set is able to handle fixed-depth images (8, 16, 32, or 64 bit/pixel) only.

Finally, the PEs of PAPRICA-3 are able to access directly the data of the neighboring PEs, thus allowing a highly efficient implementation of cellular automata operations. Conversely, in the MMX technology multiple accesses to the memory are required and this reduces the overall performance, even if this effect is lessened by the presence of multiple levels of cache memory.

Performance evaluation

For comparison purposes the algorithms described in obstacle and lane detection have been evaluated on the PAPRICA-3 architecture, featuring 256 processors, a 100 MHz clock and a 20 ns memory access time, and on two different MMX Intel Pentium architectures, featuring a 66 MHz bus speed and a 166 MHz or 200 MHz processor clock.

Table 1 shows the results for the low-level portion of lane and obstacle detection algorithms [3] which include both pointwise operations and cellular automata filters. The size of the processed images in 256×256 pixels. As a first consideration, it is important to note that the processor clock variation between the two MMX based processors does not seem to equally affect the execution time: infact, despite a frequency increase of 20% only 11–15% speedups have been measured. The rationale behind this result is that both systems use the same bus (66 MHz) for memory access which, for this implementation, represents the actual bottleneck.

These results show that PAPRICA-3 performs two to four times better than MMX processors depending on the algorithm. It should be considered, however, that these results have been obtained in a configuration where both architectures offer the maximum efficiency: namely when the number of PEs matches the image width in case of PAPRICA-3, and when processing 8-bit-deep images for the MMX based processor. While the first condition is assumed to be valid, the processing of images with a reduced depth is generally envisaged, particularly for real-time constrained applications; in this case PAPRICA-3 reaches higher efficiency levels than MMX based processors.

Table 1. Performance evaluation on PAPRICA-3 and MMX based architectures; the values referring to the PAPRICA-3 system have been obtained thanks to a simulator

	Low-level obstacle detection (ms)	Low-level lane detection (ms)
PAPRICA-3 (100 MHz, 256 PEs)	1.04	1.67
Pentium MMX (166 Mz)	3.2	7.0
Pentium MMX (200 MHz)	2.7	6.2

Discussion

In this work both the architectural and algorithmic aspects of the challenging problem of automatic vehicle guidance have been discussed. In particular, two different solutions have been pointed out and analysed: in general, the former, based on a special-purpose processing system, requires the complete design of the computer architecture as well as a programming language and a debugging environment. Conversely, the latter takes advantage of standard development tools and environments but suffers from a less specific instruction set and a less oriented system architecture. Moreover, since the use of MMX extensions relies on a SIMD computational paradigm which cannot be fully exploited by a standard C compiler, specific approaches are needed: in fact, in order to obtain a highly efficient code, either assembly code-specific parallel programming extensions to the C language are required.

Nevertheless, the advantages offered by the first solution, such as an *ad hoc* design of both the processing paradigm and the overall system architecture, are diminished by the necessity of managing the complete project, starting from the hardware level (design of the ASICs) upto the design of the architecture, to the programming language along with an optimizing compiler, and finally to the development of applications using the specific computational paradigm.

Anyway, in the comparison between these two solutions not only the details about the architecture must be taken into account, but the technological aspects need to be considered as well, the main issues being:

(1) the fast technological improvements, which tend to reduce the life-time of the system;
(2) the costs of the system design and engineering, which are justified for productions based on large volumes only.

These additional considerations point out that as soon as a new technological process becomes available, the design must be reconsidered from scratch, since new computational paradigms could benefit from the increment in the computational power derived by a higher integration, as happened for SIMD processors and for MMX extensions.

The development of dedicated architectures was one of the main research directions in the mid 1980s, when the technology was pushed at its limits and general-purpose processors could not provide sufficient computational power to support fast processing of large data structures, like bit-mapped images. The "general-purpose" solution was definitely unsuitable for some specific problems for which even expensive *ad hoc* solutions based on long-design, development, and testing times were justified.

Then, when SIMD machines became possible, a large number of simple processing elements were integrated on the same chip. There was a large explosion of custom architectures, mostly based on processor arrays with a bi-dimensional grid interconnection scheme since it was thought to be the best solution for image processing. Starting from the original ideas of Unger [31], Fountain [13] classified in chronological order the different research projects and implementations according to a three generation taxonomy.

The early ILLIAC [5] and CLIP [12] machines belong to the first generation, mainly devoted to low-level image processing tasks. The second generation comprises systems such as the ICL DAP [26], the Goodyear MPP [2] and the CLIP4 system [14] which have evolved to complete processing systems with dedicated operating

systems and languages and extended the application spectrum to high-speed scientific computation.

The triggering factors of the third generation were the availability and widespread use of VLSI technologies of the 1980s and the natural mapping of a bi-dimensional interconnection scheme over the planar structure of a silicon chip. This led to an explosion of different proposals and implementations [30, 25, 19, 29, 27, 10] and in some cases to the redesign of previous architectures. Most designs share original characteristics such as the bidimensional mesh interconnection scheme and bit-serial computation, while other ones, such as the Connection Machine [29], have increased the complexity of the interconnection network or widened, as in the CLIP7 [15] or MasPar [24] systems, the data path of the elementary PE. Other machines, such as the AIS [27] and the PAPRICA-3 [16, 9], have a 1-D interconnection scheme emulating a 2-D mesh organization.

These architectures demonstrated to be extremely important since they allowed to design and efficiently test solutions based on new approaches and new computational paradigms, such as Cellular Automata [23] or Mathematical Morphology [28, 17].

Moreover, the following availability of fast internet working facilities gave impulse to the MIMD paradigm, for which processors operating at very high clock frequencies were needed. Thanks to the widespreading of the Internet, in the mid-1990s, multimedia applications gained more importance, and image processing became a basic tool for the solution of problems which were becoming more and more common. Therefore, after a first stage in which *ad hoc* hardware boards were build to support fast image compression, and image processing, general-purpose processors were built considering that much of their computational power had to be devoted to the processing of images. Although in a period of technological pressure, Intel introduced the Multimedia Extension (MMX) technology, which, thanks to an overlapping with hardware used in floating point operations and to a small increment in the chip size, included specific instructions tailored to the processing of new data structures, such as pixels.

Under this framework, the advantages offered by *ad hoc* solutions are more and more confined in a small and specific application field, where hard constraints impose the design of a specific system architecture, for example in the development of embedded systems such as in the automotive field.

Trends in the Automotive Field

Similar issues have been addressed by other research institutes working in the automotive field (a thorough survey of state-of-the-art research can be found in [8]). The great majority of the projects involving the use of machine vision as a mean to sense the environment started 5 to 10 years ago with the development of *ad hoc* computer architectures that could provide sufficient computational power to support low-level image processing. Among them, the most important and significant examples are the cases of Carnegie Mellon University (Pittsburgh, PA) and Universität der Bundeswehr (München, Germany), where both research groups developed their own system architectures: in the first case a 16 k MasPar MP-2 was installed on the experimental vehicle NAVLAB I [21, 22], and in the second case special-purpose boards were included in the Transputer-based architecture of VITA.

On the other hand, both groups are currently heading their research toward the use of general-purpose processors (such as the Intel family) in a commercial architecture with off-the-shelf components, such as frame-grabbers and I/O boards. These systems can be efficiently used to test the algorithms and any new approach without worrying about the technological advances during the course of the project. Once the algorithms have been defined, tested, and tuned according to the specific application and environment, the engineering phase may take place in order to produce the final embedded system taking advantage of up-to-date technology.

As a final consideration, current technology allows to have sufficiently powerful processing engines to provide an efficient development infrastructure (considering both hardware devices and programming tools). Obviously, since the final product generally needs specific characteristics such as low cost, low power consumption and low physical size, the final phase will consist of the re-engineering of the whole system leading to a special-purpose *ad hoc* product.

Therefore, although some years ago the extremely expensive development of special-purpose hardware was required because of the low performance of general-purpose systems, currently the special-purpose solution is the basis for the final marketing product in specific hard-constrained applications.

Acknowledgement

This work has been partially supported by the Italian National Research Council (CNR) in the framework of the MADESS2 Project; up to date information can be found at http://millemiglia.ce.unipr.it

References

1. Ballard, D. H. & Brown, C. M. (1982) *Computer Vision*. Prentice Hall.
2. Batcher, K. (1980) Design of a Massively parallel processor. *IEEE Transactions on Computers* **C-29**: 836–840.
3. M. Bertozzi & Broggi, A. (1988) GOLD: a Parallel Real-Time Stereo Vision System for Generic Obstacle and Lane Detection. *IEEE Transactions on Image Processing*, **7**(1): 62–81.
4. M. Bertozzi & Broggi, A. (1998) Tools for code Optimization and System Evaluation of the Image Processing System PAPRICA-3. *Journal of Systems Architecture*, **45**: 519–542.
5. Bouknight, W. L. *et al*. The ILLIAC IV system. *IEEE Proceedings*, **60**: 369–388, 1972.
6. Bray, B. B. (1996) *Programming the 80286*, *80386*, *80486 and Pentium-based Personal Computer*. Englewood Cliffs, New Jersey, USA: Prentice Hall, Inc.
7. Broggi, A. (1998) Special-Issue on ''Machine Vision for Intelligent Vehicles and Autonomous Robots,'' A. Broggi guest-editor. *International Journal on Engineering Applications of Artificial Intelligence*, **11**(2).
8. Broggi, A., Bertozzi, M., Fascioli, A. & Conte, G. (1999). *Automatic Vehicle Guidance: the Experience of the ARGO Autonomous Vehicle*. World Scientific Co. Publisher, Singapore. ISBN 981-02-3720-0
9. Broggi, A., Conte, G., Gregoretti, F., Sansoè, C., Passerone, R., & Reyneri, L. M. (1998) Design and Implementation of the PAPRICA Parallel Architecture. *The Journal of VLSI Signal Processing*, **19**(1): 5–18.
10. Broggi, A., Conte, G., Gregoretti, F., Sansoè, C., & Reyneri, L. M. (1997) The Evolution of the PAPRICA System. *Integrated Computer-Aided Engineering Journal - Special Issue on Massively Parallel Computing*, **4**(2): 114–136.
11. Broggi, A. & Gregoretti, F. (1996) Special-Issue on ''Special-Purpose Architectures for Real-Time Imaging,'' A. Broggi and F. Gregoretti guest-editors. *Real-Time Imaging Journal*, **2**(6): 329–330.
12. Duff, M., Watson, D. M., Fountain, T. & Shaw, G. (1973) A Cellular Logic array for Image Processing. *Pattern Recognition*, **15**: 229–247.
13. Fountain, T. (1987) *Processor Arrays: Architectures and applications*. Academic-Press, London.
14. Fountain, T. & Goetcherian, V. (1980) CLIP4 Parallel Processing System. *IEEE Proceedings*, **127E**: 219–224.
15. Fountain, T. & Matthews, K. (1988) The CLIP 7A Image Processor. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, **10**(3): 310–319.
16. Gregoretti, F., Passerone, R., Sansoè, C. & Broggi, A. (1996) The PAPRICA-3 Parallel Processor. In *Proceedings MIPRO'96 Symposium* , Opatija, Croatia.
17. Haralick, R. M., Sternberg, S. R. & Zhuang, X. (1987) Image Analysis Using Mathematical Morphology. *IEEE Transaction on Pattern Analysis and Machine Intelligence*, **9**(4): 532–550.
18. Helman, D. & Jájá, J. (1995) Efficient Image Processing Algorithms on the Scan Line Array Processor. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, **17**(1): 47–56.
19. Robinson, I. N. & Moore, W. R. (1982) A parallel processor array architecture and its implementation in silicon. In *Proceedings of IEEE Custom Integrated Circuits Conference*, 41–45, New York, Rochester.
20. Intel Corporation. *MMX Technology Programmers Reference Manual*. Intel Corporation, 1997. Available at http://www.intel.com.
21. Jochem, T. M. & Baluja, S. A. (1993) Massively Parallel Road Follower. In M. A. Bayoumi, L. S. Davis, and K. P. Valavanis, (eds.), *Proceedings CAMP'93 - Computer Architectures for Machine Perception*, pages 2–12, New Orleans, IEEE Computer Society.
22. Jochem, T. M. & Baluja, S. (1993) Massively Parallel, Adaptive, Color Image Processing for Autonomous Road Following. In: Kitano, H. (ed.), *Massively Parallel Artificial Intelligence*. AIII Publishers in cooperation with MIT Press.
23. Margolus, N. & Toffoli, T. (1990) Cellular Automata Machines. In: Doolen, G. D. *et al*., (ed.), *Lattice Gas Methods for Partial Differential Equations*. Redwood City, California, Addison Wesley, pp. 219–249.
24. MasPar Computer Corporation, Sunnyvale, California. *MP-1 Family Data-Parallel Computers*, 1990.
25. NCR Corporation, Dayton, Ohio. *Geometric Arithmetic Parallel Processor*, 1984.
26. Reddaway, S. (1973) DAP–A Distributed Array Processor. In *1st Annual Symposium on Computer Architectures*, pages 61–65, Florida.
27. Schmitt, L. A. & Wilson, S. S. (1988) The AIS-5000 Parallel Processor. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, **10**(3): 320–330.
28. Serra, J. (1982) *Image Analysis and Mathematical Morphology*. Academic Press, London.
29. Thinking Machines Corporation, Cambridge, Ma. (1991) *Connection Machine CM-200 Series - Technical Summary*.
30. Sudo, T., Nakashima, T., Aoki, M. & Kondo, T. (1982) An LSI adaptive array processor. In *Proceedings IEEE International Solid-State Circuits Conference*, pages 122, 123,307.
31. Unger, S. (1958) As computer oriented toward spatial problems. *Proceedings IRE*, **46**: 1744–1750.