Available online at www.sciencedirect.com
SCIENCE DIRECT®
Real-Time Imaging 9 (2003) 99–112
www.elsevier.com/locate/rti

A modular and scalable architecture for PC-based real-time vision systems

Judit Martínez*, Eva Costa, Paco Herreros, Xavi Sánchez, Ramon Baldrich

Centre de Visió per Computador, Edifici O, Campus UAB, 08193 Bellaterra, Spain

Received 26 July 2002; received in revised form 11 December 2002; accepted 6 January 2003

Abstract

PC-based real-time vision systems are becoming a de facto standard in industrial applications. They are composed of an illumination system, an image acquisition system and a processing system. In this article, a modular and scalable architecture for real-time vision systems is proposed. On the one hand, we define an acquisition module that allows simultaneous acquisition of up to three monochrome cameras. The acquisition system can be scaled by adding more acquisition modules. The architecture allows simultaneous acquisition of all the modules. On the other hand, we define a processing system composed of different modules and sub-modules which specialize in a particular task: the master module interacts with the external systems; the slave module interacts with the acquisition system and manages the results of its processing sub-modules; each processing sub-module processes the information provided by one single camera. Scalability is provided by increasing the number of slave modules and processing sub-modules that compose the complete vision system. Fast real-time applications can be achieved by assigning one processor per processing sub-module. In this case, multiple PC can be used. Inter-computer communication among modules is carried out by means of sockets (following a master–slave design pattern); intra-computer communication is carried out by means of pipes, shared memory and events. We emphasize herein some real-time considerations related to the multi-processor architecture and the multitasking operating system that allow the implementation of the proposed architecture for real-time vision systems applications. An implementation of this architecture is exemplified with an application successfully installed in a manufacturing company.

1. Introduction

A computer vision system is composed of an illumination system, an acquisition system and a processing system. The illumination system allows setting the best lighting conditions over the objects to be acquired. The acquisition system obtains images from the objects and transfers them to the processing system. The processing system analyzes these images and transfers the results to some external systems; it can also collect information from external systems such as sensors or databases. The particular implementation of each one of these systems depends on the application, so that, a generalization is not possible. However, there exist common characteristics of the real-time computer vision applications that allow defining a general modular architecture for the acquisition and processing systems. A modular scalable architecture simplifies the design and the implementation of the whole system. Modularity allows splitting a complex problem into simpler ones and using only those modules that are required for a particular application. Scalability of the architecture involves that, if the system becomes more demanding (in speed, image resolution or processing), it is possible to upgrade its performance by adding the proper modules. The design of the architecture cannot be independent of the platform. We center herein in an Intel platform that uses commonly available PC components. PC-based computer vision applications running under Windows operating systems are becoming de facto standard in industrial applications. This is the reason why we deep into the design of a PC-based modular scalable architecture for computer vision applications. A different approach is taken by some other authors that center on the design of specific boards for image processing [1–3].

Real-time computer vision applications are, for example, those that provide a classification of an object

*Corresponding author.
E-mail addresses: judit@cvc.uab.es (J. Martínez), evac@cvc.uab.es (E. Costa), paco@cvc.uab.es (P. Herreros), javier@cvc.uab.es (X. Sánchez), ramon@cvc.uab.es (R. Baldrich).

1077-2014/03/$ - see front matter
doi:10.1016/S1077-2014(03)00002-0

moving on a conveyor belt before the next object on the conveyor belt is ready for acquisition. Another real-time vision application is the one that provides the 3D displacement coordinates of a vision-guided robot system when the robot requires them. The particular time involved in these systems depends on the particular application (from milliseconds to many seconds or minutes). The relevant characteristic of real-time systems is not the whole time required but the variation of this time due to uncertainties. These uncertainties can cause the system to fail if not taken into account. Therefore, real-time applications add special requirements that need to be taken into account when defining a PC-based architecture under a Windows operating system. There exist other operating systems or even extensions of the Windows operating system that are specifically designed for real-time operations [4,5]. However, we center on commonly used operating systems, such as, Windows NT 4.0 (or its successors Windows 2000 and Windows XP, which basically preserve the same kernel) because of the availability of code, libraries and drivers that facilitate the integration of common off-the-shelf hardware and software components.

We describe in this article a modular and scalable architecture for general PC-based real-time vision applications. Sections 2 and 3 are devoted to describe it and to analyze some real-time considerations, respectively. The implementation of this architecture is exemplified in Section 4, where the details of a real-time application that has already been installed in a manufacturing company are shown. Finally, Section 5 summarizes the main conclusions.

2. Modular architecture

Computer vision systems can be divided into three main systems: an illumination system, an image acquisition system and a processing system. As for the acquisition and processing system, we define a modular architecture that facilitates scalability and interaction between them. The architecture is shown in Fig. 1.

2.1. Image acquisition system

The image acquisition system provides the information that will be analyzed in the processing system. This information is collected from the scene by means of *cameras* and transferred to the PC-based processing system by means of *frame-grabbers* (FG). There exist different types of frame-grabbers that differ on their capability to interface particular types of cameras, their storage and transfer features, their synchronization signals and the availability of input/output signals. See [6,7] for a general background on image acquisition.

Given a particular computer vision application, the set-up of a camera configuration depends on a priori knowledge of the requirements of the application. One of the features that influences this configuration is the *image resolution*, that is, the pixels per millimeter of the

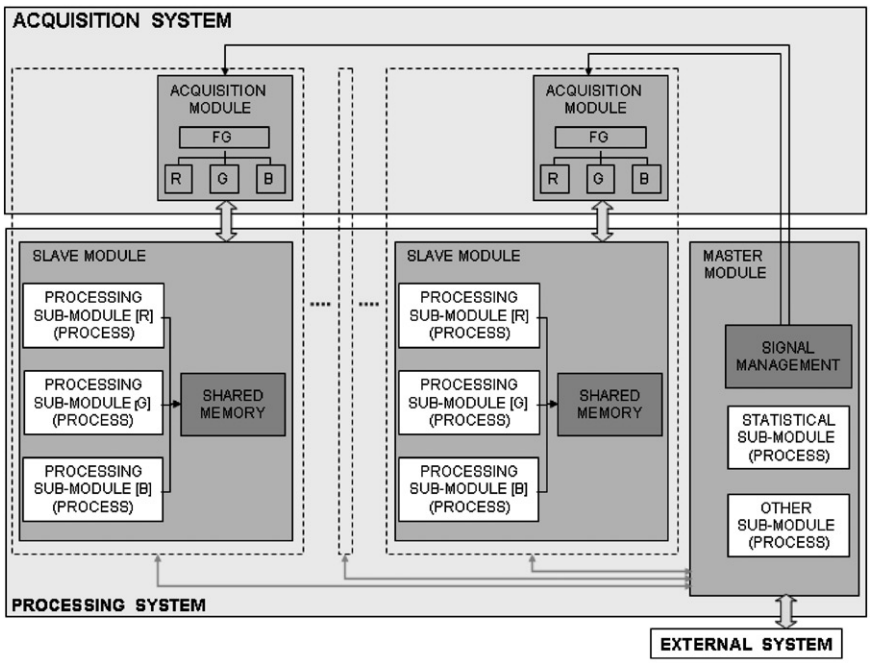

Fig. 1. Modular architecture of a PC-based vision system.

image. It depends on the dimensions of the smallest particle of the object to be detected in the image. For example, if the vision system has to detect object's defects smaller than 1 mm, it is a good practice to fix the resolution to be more than 2 pixel/mm so as to avoid dividing the smallest defect partially into two adjacent pixels (although subpixel processing algorithms could be used). If the vision system covers a large area, it is likely that more than one camera will be necessary. This is also the case when different views of the object are needed (although a mobile camera could also be used at the expenses of increasing the complexity of the complete system).

Therefore, assuming static camera machine vision applications, the number of required cameras depends on:

- the image resolution,
- the area of the object,
- the number of views of the object.

Increasing the resolution, the area, or the number of views involves either increasing the number of cameras or choosing a higher resolution camera, that is, a camera with a larger number of cells in the sensor (CCD/CMOS).

Every camera needs to be connected to a frame-grabber. There exist many frame-grabbers that accept more than one camera. In this case, the acquisition of the image from the cameras can be simultaneous or sequential (multiplexed). The maximum number of simultaneous monochrome images that the common off-the-shelf frame-grabbers can acquire is three. This is due to the fact that these frame-grabbers accept color cameras and they assume that each one of the monochrome images corresponds to one of the color bands (R, G, and B, respectively) of a hypothetic color camera. In this case, all the cameras have to be synchronized, that is, they need to share common sampling signals such as pixel clock, horizontal and vertical clock, and trigger. There exist a few frame-grabbers that can simultaneously acquire more images, not necessarily synchronized, at the expenses of a higher cost. Therefore, assuming simultaneous acquisition, the number of frame-grabbers required for a machine vision application depends on the number of cameras. It will be considered that one frame-grabber is needed for every three cameras.

We define a modular and scalable acquisition system based on the previous description. Each module is composed of a group of up to three monochrome cameras and one frame-grabber. Fig. 2 shows one of such modular acquisition systems. The number of modules that are needed for a particular machine vision system depends on the particular requirements of the application.

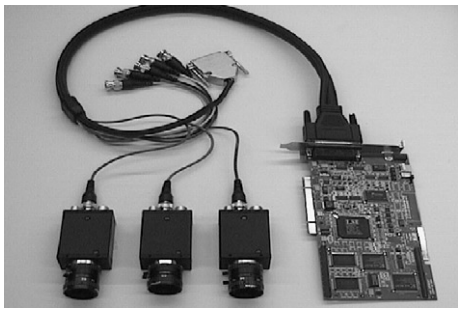

Fig. 2. Modular acquisition system.

2.2. PC-based processing system

The processing system collects the information provided by the image acquisition system as well as other external systems (a barcode reader, an external database, etc.); it manages and analyzes this information and communicates the results to some other external systems (a programmable logic controller, a robot, another PC or a human operator).

Each modular acquisition system provides up to three simultaneous monochrome images that can be processed sequentially or simultaneously. Since we focus on real-time applications, simultaneous processing is of major interest. To define a modular and scalable processing architecture coherent with the modular acquisition system described in Section 2.1, each software module has to be independent from the number of cameras per frame-grabber and the number of frame-grabber per PC. In particular, the input data of each processing sub-module will be one image and its processing will be carried out by specific functions that depends on the application. The execution of simultaneous processing sub-modules can be implemented using either *threads* or *processes*. Due to possible memory access bottleneck, processes are preferred as will be explained next.

2.2.1. Access to the image

Frame-grabbers have to acquire image data from the camera and make it available to the processor. The simplest frame-grabbers cannot acquire and transfer data at the same time; improved ones allow simultaneous acquisition and transfer by means of ping-pong memories. The transfer rate depends on the bus. Most common frame-grabbers reside on the PCI bus and the most efficient ones operate as PCI bus masters and manage the transfer themselves. High-speed systems usually transfer the image data out of the frame-grabber into the PC main memory because the processor access speed to this memory is much faster than the access to the frame-grabber's memory over the PCI bus.

The acquisition module described in Section 2.1 simultaneously acquires up to three monochrome images as if they were the three color bands (RGB) of a color camera. Each processing sub-module will be able

to access a particular image if advanced frame-grabbers are used that allow: (a) physically contiguous allocation of this data and (b) the possibility to specify the allocation address. These requirements will become clear in the next sections in which the correspondence between a process and a software processing sub-module will be defined.

2.2.2. Process versus thread

Scalable processing can be accomplished using multiple processors and multitasking operating systems. Windows NT 4.0, for example, is a multitasking operating system whose basic unit of execution is the thread. A process contains one or more threads. A multi-processor computer allows for one running thread per processor. Multithreaded applications and multiprocessor systems require special considerations for memory allocation since it can have a great effect on the performance depending on the type of application and the memory allocation strategy. Contention on critical sections is a major problem. Additionally, the existence of one memory cache per processor needs to be taken into consideration.

The feasibility of *using a thread as the modular processing software component* is first analyzed. This is supported by the fact that the thread is the basic unit of execution.

A thread state can be either waiting (cannot run until a specified event occurs), ready (ready to run, but no processor is currently available) or running (currently running on a processor). Windows NT 4.0 uses a priority-based round-robin algorithm to manage the available CPU among the runnable threads. When a CPU becomes available a context switch takes place, that is, the kernel system finds the highest priority queue with ready threads on it, removes the thread at the head of the queue, and runs it. The most common reason for a context switch is when a running thread has to wait, either because it touches a page that is not in its working set and the memory manager has to resolve the page fault or because it executes some of the functions that explicitly block the execution until a specified event occurs. To prevent CPU-bound threads from monopolizing the processor, the kernel imposes a time limit, called the thread quantum. Unless there is a higher-priority thread ready, the current thread runs for one quantum and then the kernel preempts it and moves it to the end of its ready priority queue. For a single processor system, it is very rare to find contention on critical sections since the thread that owns the critical section is unlikely to be preempted. In a multi-processor system, such contention is much more likely since more than one thread is running simultaneously. When there is contention (one thread tries to acquire the critical section while another thread holds it) additional system calls and context switches are required in order for the requesting thread to wait until the owning thread has released the critical section. This is a common situation when more than one thread tries to allocate memory simultaneously. One thread will block on the critical section guarding the heap. The other thread must then signal the critical section when it is finished to release the waiting thread. This adds significant overhead. It is possible to minimize this effect by implementing multiple heaps and serializing the access to the heaps, but libraries do not support it. The additional overhead in this implementation is offset by greatly reducing the number of times a thread must wait for heap access.

Critical resources management in multi-processor applications needs special considerations due to the existence of cache memories. Computers generally have a fast memory cache between the CPU and the main memory. Each processor's memory cache must maintain a consistent view of the main memory. This is accomplished by dividing memory into small chunks (that make up a cache line). To update a cache line, a processor must first gain exclusive access to it by invalidating all other copies in other processor's caches. When the processor has exclusive access to the cache line, it may safely update it. If the same cache line is continuously updated from many different processors, that cache line will bounce from one processor's cache to another. A more stunning problem occurs when two or more variables occupy the same cache line. Updating any of the variables requires exclusive ownership of the cache line. Two processors updating different variables will access the cache line as much as if they were updating the same variable. Any function that takes proportionally more time as the number of processors increases are likely victims of cache blockade.

Therefore, the threads must be able to work independently of each other to scale effectively. This can only be guaranteed if the data structures are padded to ensure that frequently accessed variables do not share a cache line with anything else (see [8] for deeper information).

A different approach consists of *using a process as the modular processing software component*. Since Windows NT 4.0 offers virtual memory to its processes, each process has its own private address space. It is constructed on a page-based memory management scheme that divides all the memory into equal chunks called pages. Each page is 4096 bytes (4k) in size. Every process has its own page directory. Therefore, it is guaranteed that the cache line of two processes will never share the same memory page. No cache blockade between processes is possible.

Because two processes can use the same virtual address to refer to different locations in memory, processes are not able to communicate addresses one to another. Specific mechanisms are required to share memory among processes. *Memory-mapping files* are

one example of reserving a range of addresses that are contiguously by default, and can be used to share memory among processes.

As a result, we conclude that using a process as the modular processing software component is better than using a thread since no special considerations have to be taken into account with respect to memory data structures of the application's variables in order to avoid contention on memory access due to the existence of multiple memory caches.

2.2.3. Sharing information among processes

File mapping enables a process to treat the contents of a file as if they were a block of memory in their process's virtual address space. *A file view* is the portion of the virtual address space that the process uses to access the file's contents. The process can use simple pointers to examine and modify the contents of the file. When two or more processes access the same file mapping, each process receives a pointer to memory in its own address space that can use to read and modify the contents of the file. Related to the modular architecture presented up to this point, each file will contain the images acquired by one of the modular acquisition systems (that is the set of three monochrome images) and every file view refers to one of these images. Therefore, processes running in the same computer that share information can obtain it through file mapping.

A *pipe* is a shared stream that processes use for communication. The communication can be either one-way or duplex. In the first case, one process writes information to the pipe, and then the other process reads the information from this pipe. In the second case the two processes write and read to and from the pipe. The proposed modular architecture uses pipes to communicate processes in the same computer.

A *socket* is one endpoint of a two-way communication link between two processes running on a network. Sockets are created and used in a different way to pipes because they make a clear distinction between client and server. The socket mechanism naturally lends itself to the implementation of multiple clients attached to a single server. One of the processes, the client, connects to the other process, the server, typically to make a request for information or and interchange. The proposed modular architecture uses sockets to communicate processes in different computers.

An *event* is a synchronization object that allows one process to notify another that an action has occurred. The events have two states: signaled or non-signaled. A process can be blocked waiting for an event to be signaled. In fact, the calling process could enter an efficient wait state, consuming very little processor time, while waiting for the event to be signaled. The proposed modular architecture uses events to signal each processing sub-module that their images are available.

Fig. 3 summarizes the communications involved in the proposed modular architecture.

2.2.4. Modular processing architecture

Based on the previous analysis we define a scalable processing system divided into a set of modules and

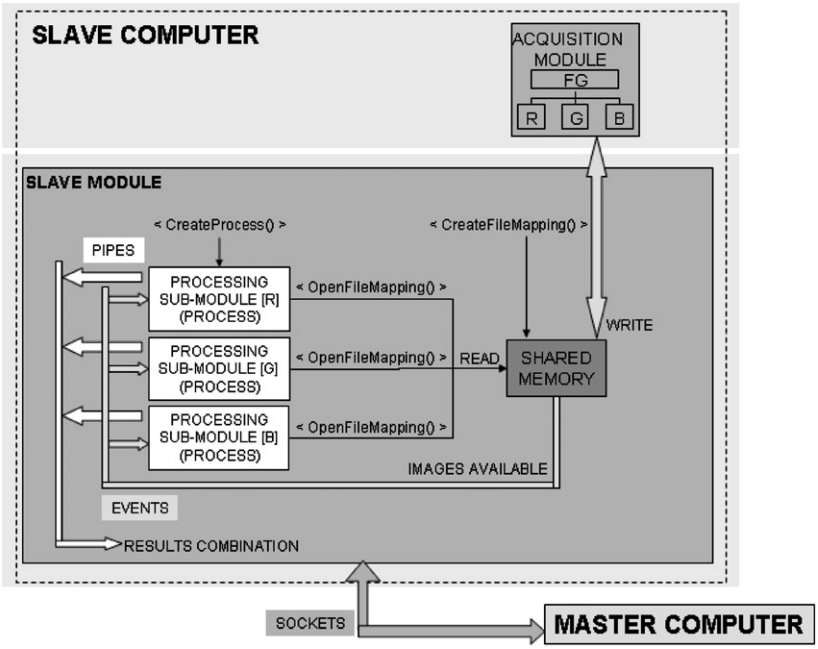

Fig. 3. Communication involved in the PC-based modular architecture.

sub-modules:

- a *slave module*, which contains *processing sub-modules*;
- a *master module*, which contains *additional sub-modules*.

Each module and sub-module is implemented as a process. The slave module and its sub-modules run in a *slave computer*. Each slave computer also contains its acquisition modules. The master module and its sub-modules run in a *master computer*. Inter-computer communication is carried out between modules by means of sockets. Intra-computer communication is carried out among a module and its sub-modules by means of pipes. The slave module and the processing sub-modules share the information by means of a file mapping. Whenever a new image is available, the corresponding processing sub-module is signaled by means of an event (see Fig. 3).

These are the tasks associated to each module and sub-module:

- The slave module: manages the acquisition module, gathers the results of the processing sub-modules and communicates with the master module.
- The processing sub-module: analyzes the images provided by one camera and transfers the results to the slave module.
- The master module: communicates with the external systems (user, external computer or PLC), gathers the result from the slave modules and provides the control signals to the acquisition system.
- The additional sub-module: provides utilities which are related to the particular application (for instance, a statistical sub-module).

Notice that a maximum of three processing sub-modules are needed for a particular acquisition module. However, the maximum number of simultaneous processing sub-modules executed on a PC depends on the number of processors, which is up to two in common off-the-shelf computers. If concurrent execution of more than two processing sub-modules is required to achieve a real-time application, an increasing number of computers will be necessary.

3. Real-time considerations

A real-time vision system involves both real-time acquisition and real-time processing. Real-time acquisition means that the camera has to acquire the image of an object or a particular part of this object whenever it is in the camera's field of view. Real-time processing means that the result of the computations must be provided before another acquisition occurs. The correctness of the results not only depends on the logical correctness of the computations, but also on the time at which both the acquisition and the end of processing take place. Therefore, the relevant aspect of real-time systems is not the whole time required to finish but the variation of this time due to uncertainties. Each component or process involved in a computer vision system adds uncertainties that can be either relevant or meaningless depending on their implication in the whole system or their relative value with respect to that of other components or processes of the system. Let us analyze these uncertainties in each one of the components.

3.1. Real-time acquisition

Triggering the cameras, that is, forcing them to immediately acquire an image, must be a real-time operation. It may depend on the response of an external sensor that signals the presence of the object in front of the camera. If more than one image of the same object has to be acquired, the cadence of acquisition may rely on the conveyor's encoder that signals the conveyor belt's speed. It is quite common that both the sensor and the encoder signals be manipulated or combined to generate the trigger signal to the cameras.

The manipulation of external signals in order to obtain the trigger signal can be done either using an input/output board or an advanced frame-grabber. The later will be preferred, if possible, because the response only depends on the board circuitry. On the other hand, input/output boards rely on how the operating system manages the execution of the instructions addressed to this board.

3.1.1. I/O boards

An I/O board has several software components: an initialization routine (that sets up data structures to the main memory of the PC), an interrupt service routine [ISR] (that executes time-critical processing: handle the interrupt, save the state necessary for processing the interrupt and queue a DPC), a deferred processing call [DPC] (that executes non-time-critical processing excluding all other processing except for ISRs) and system processes (that executes low-priority jobs).

Windows NT 4.0 handles interrupts on a preemptive basis: when an interrupt occurs, all executions at lower interruption levels are suspended and execution begins immediately on the highest level request. The interrupt service routine continues until the highest level process has been completed. On the other hand, DPCs are placed in a FIFO queue. There is no notion of priority in a DPC queue, so that DPCs of lower priority can be handled before DPCs of higher priority if they have entered the queue later. Therefore, the execution of a particular I/O DPC depends on the DPCs of other

devices. Moreover, since device drivers in Windows NT 4.0 are kernel-level code that extend the capabilities of the kernel, processes executing in kernel mode can mask some or all interrupts by raising the CPU's current Interrupt request levels (IRQL), thought it is considered a violation of the NT device driver rules. Therefore, it is possible that an unimportant interrupt can block a real-time priority user-level process since the execution time of some device drivers can be quite large. See [9] for a review on the subject.

Triggering a camera has to be a deterministic action. If the I/O board manages the triggering signals (input signals: external sensor and/or encoder; output signal: trigger to the camera), it must run in a deterministic way. This cannot be guaranteed in a user-level management of the I/O board since the user is only available to use high-priority processes but not the highest priority tasks deferred to interrupt routines. All user-defined tasks can be preempted by this higher priority task, so their determinism depends on both the operating system and the highest priority task's worst-case code path length.

Real-time developers in different areas are conscious of these limitations and there exist some interesting articles [10–14] that analyze and test standard PC configurations using Windows NT for real-time operations. Real-time vision system developers can minimize the effect of uncertainty by means of increasing the field of view of the camera, to guarantee that the object is still in the image although the acquisition instant has been modified, or increasing the overlapped areas in continuous acquisition. The increment of field of view or overlapped area depends on the maximum uncertainty in the system. Notice that augmenting the field of view involves reducing the image resolution and augmenting the overlapped areas involves increasing the number of images required to cover the same object. Therefore, accounting for uncertainties cannot be compatible with the requirements of the applications [15]. In this case, more deterministic solutions need to be taken into account for image acquisition.

3.1.2. Frame-grabber

Advanced frame-grabbers have specific circuitry to manage trigger signals. In this case, the triggering does not depend on the operating system but the frame-grabber's hardware. Although hardware circuitry response also has uncertainties, their value (in time units) are much smaller than those associated to software responses (at least three orders of magnitude). Therefore, using the frame-grabber as the triggering device can be a solution for many computer vision applications since the modification of the field of view or the overlapped area will be much smaller than in the previous case (Section 3.1.1).

These kinds of frame-grabbers allow configuring trigger signal characteristics such as the time delayed between input and output signals, to combine input signals or to define the active duration of the output signals. If further manipulation of the signal is required, such as noise filtering or frequency modification, an external circuitry has to be designed.

3.2. Real-time processing

Processing will be carried out using commercial off-the-shelf computers running a multitasking operating system. The speed of computation will mainly depend on how efficiently the code is implemented and how well-managed are the resources, specially the memory. It will increase as the processor speed increases. This is the main advantage of using commercial off-the-shelf computers since it is likely that by simply changing the processor the whole system increases performance. Processing speed can also be increased by scaling the number of processor. Commercially available multiprocessor systems are dual PC. If more processors are required, additional PCs can be used. In this case they should be interconnected using either ethernet or fiberoptic, depending on the amount of information that they have to exchange (the fiberoptic allows higher transfer rates).

An important property associated with user-defined processes is the priority of execution. The priority model, in Windows NT 4.0, includes 32 priority levels divided into real-time classes (labeled from 16 to 31) and dynamic classes (labeled from 0 to 15). Only kernel mode processes (related to IRQL) will have a higher priority than real-time time-critical user-defined processes. Therefore, the real-time processing is bounded by these kernel mode processes [9]. It is recommended to avoid installing unnecessary drivers to minimize their influence.

4. Application

In this section, a real-time vision system for TV screen quality inspection is introduced. The system, which is already installed in a manufacturing company, has been developed using the scalable off-the-shelf modular architecture introduced in the previous sections.

The object to be inspected is the glass surface of TV screens. The defects on this surface can be either due to bubbles inside the glass or inhomogeneities in the phosphor distribution. The phosphor coats the inside surface of the TV screen in such a way that it forms triads of vertical lines of three consecutive colors: red (R), green (G) and blue (B). Whenever the line continuity breaks or it becomes thinner or it overlaps adjacent lines, a defect in phosphor distribution occurs.

There exist many variations of this simple classification of defects that make the inspection process computationally intense. Fig. 4 shows some of the defects to be inspected.

The size of the TV screen is 244×314 mm. The smallest defect is about 0.1 mm. Therefore, the minimum required resolution is 2 pixels per 0.1 mm, that is 20 pixels/mm. Since the size of a standard CCIR cameras is 762×562 pixels, eight cameras are required to cover the height of the TV screen and 14 images must be acquired by each camera to cover its width as it moves along the conveyor belt (see Fig. 5). The

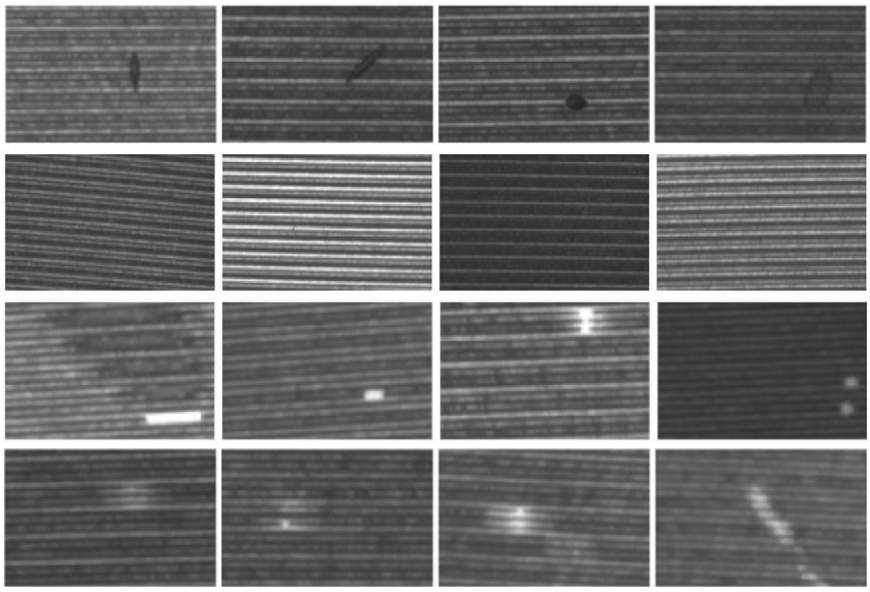

Fig. 4. Some examples of defects to be detected: (a) bubbles, (b) landing, (c) line-out and (d) line reduction, respectively.

Fig. 5. A picture of the acquisition system.

calculation of the required number of cameras and images also takes into account: (a) an overlapped area between adjacent images (about 1 mm in each direction), which is necessary to facilitate the installation of all the cameras so that a complete coverage of the TV screen could be guaranteed, and (b) the looseness in the location of the TV screen on the conveyor belt with respect to its width direction (about ± 3 mm). See Fig. 6. Although cameras of larger CCD format could be used, the barrel-like geometry of the TV screen surface makes difficult to keep in focus larger areas of the surface using standard optics. This is the reason why this configuration has been chosen.

The conveyor belt moves at 23 cm/s. The cadence of TV screens is about one TV screen every 6 s. Since the vision system has to perform real-time quality control, the result of the inspection has to be known before the next TV screen is ready for acquisition. Therefore, 112 images must be acquired and processed in 6 s. The acquisition is triggered by means of an external signal that combines the information of the conveyor belt's speed (encoder) and the presence of TV screen in front of the cameras (laser-barrier sensor). Taking into account the resolution of the image and the conveyor belt's speed, it ends up that, each camera acquires one image nearly every 90 ms. Therefore, to acquire 14 images per camera, the system requires 1.17 s, and 4.86 s remain for processing. It means that, 347 ms per image are available if one processor could be devoted to processing the set of images obtained by only one camera. As the number of cameras associated to a processor increases, the time available for processing each image decreases in the same factor. Therefore, the number of processors required by the whole system depends on the amount of processing, the efficiency of implementation and the processor speed. This particular application, that uses *PIII*-600 MHz, requires one processor per camera. A dual PC that handles 1 acquisition module composed of two cameras is used.

The system has been designed with the scalability property in mind. Scalability allows for higher resolution, if required, or bigger TV screens. Next sections describe particular considerations of this architecture applied to the application.

4.1. Modular acquisition system

The whole acquisition system is composed of eight cameras and four frame-grabbers. Each acquisition module is defined by two cameras and one frame-grabber. In particular, the cameras are JAI CV-M10BX CCIR and the frame-grabber is MATROX METEOR-II/MC.

The relevant characteristics of the camera for this application are:

- progressive scan, since the images must be acquired while the object keeps moving;
- asynchronous reset, since the initiation of exposure must be synchronized to external signals;
- square pixels, since precise measurements of some of the defects are required;
- availability of input and output synchronization signals, which is important in order to synchronize both cameras of the acquisition module;
- 762×562 (CCIR) pixels, which allows obtaining the required resolution while covering an area which is in focus in spite of the barrel-like geometry of the TV screen;
- high sensitivity and low S/N, which is always desirable for accuracy in computations.

The relevant characteristics of the frame-grabber for this application are that:

- it is a PCI bus master board, so, it supports real-time image transfer to the memory of the PC;
- it has 4 Mbytes of SGRAM for temporary frame storage, which is useful in long-bus access latencies;
- it captures progressive scan RGB color cameras, so that the two identical monochrome cameras of the acquisition module can be acquired simultaneously as if they were the R and G bands of a color camera;
- it provides synchronization and trigger input/output signals.

Acquisition is performed when an external signal indicates it. This signal initiates the exposure of the camera; when the image is available, the camera signals the frame-grabber, which starts the capture. In fact, this frame-grabber also allows delaying the capture a specified time after the trigger signal occurs. This is useful for the case in which the capture signal from the camera is not available.

4.2. Real-time acquisition

The cameras are located above the conveyor belt, looking down the TV screen surfaces that move along with the conveyor belt (see Fig. 5). The eight cameras cover the height of the screen (that is, the direction of shorter dimensions of the surface). Their location guarantees some overlap between the images acquired by adjacent cameras. The minimum amount of overlapped area is chosen to be the same size as the minimum defect to be detected, so that it is guaranteed that the smallest detectable defects will not be partitioned into different images. However, because of camera dimensions and limited space, the cameras

cannot be placed on a single line along the height of the TV screen. They have to be distributed in two parallel lines so that the images obtained from cameras in one line intercalate the images obtained from the cameras in the other line.

Acquisition is governed by external signals. One of them is the signal provided by the encoder. This signal generates a pulse every time the conveyor belt moves 133 μm. Since the field of view of the camera covers about 20 mm in the direction of movement of the TV screen, acquisition has to be triggered every 150 encoder pulses, during the time that the TV screen is viewed by the cameras. In order for the system to know when the TV screen is under the cameras, a second signal is required. This signal is provided by a laser-barrier sensor located in the same line as the cameras. Since there are two lines of cameras, there are also two sensors, which trigger the cameras in each line, respectively.

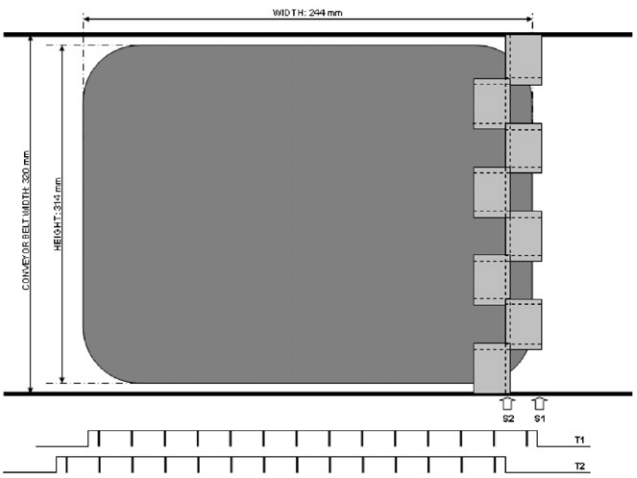

Fig. 6. Acquisition configuration: two lines of four cameras triggered by signals T1 and T2 that depend on the sensor signals S1 and S2, respectively.

Therefore, the acquisition system needs a counter that keeps track of the encoder pulses and generates a triggering signal after 150 pulses have occurred. Whenever the sensor signal of one of the line of cameras is active and the triggering signal is generated, the set of cameras in this line are triggered for acquisition. The same occurs with the other line of cameras. Fig. 6 shows this configuration.

The combination of these signals cannot be managed by the Matrox Meteor-II/MC board because it does not include a counter. Therefore, either an I/O board or an external circuitry is necessary to manage this signal. Since the use of an I/O board does not guaranty real-time responses, as already justified in Section 3.1.1, an external circuit has been designed for this application. The specific board designed for signal management allows making the acquisition more robust by adding digital filters that preserve the integrity of the signal. This is a relevant benefit in industrial environments, which use to be very noisy. Additionally, filtering proved to be necessary in this particular application, because the sensor signal generated glitches due to the presence of water on the screens, which disturbed the laser-barrier sensor. Fig. 7 summarizes the filtering and triggering scheme designed for this application. It is implemented using an ADuC812 microconverter. In this figure, signal S refers to $(S1)\,OR\,(S2)$, where S1 and S2 are the two laser-barrier sensor signals. The value of these signals is high while the TV screen is located between the emitter and the receiver. However, if a drop of water is situated on the surface of the screen in front of the emitter, the

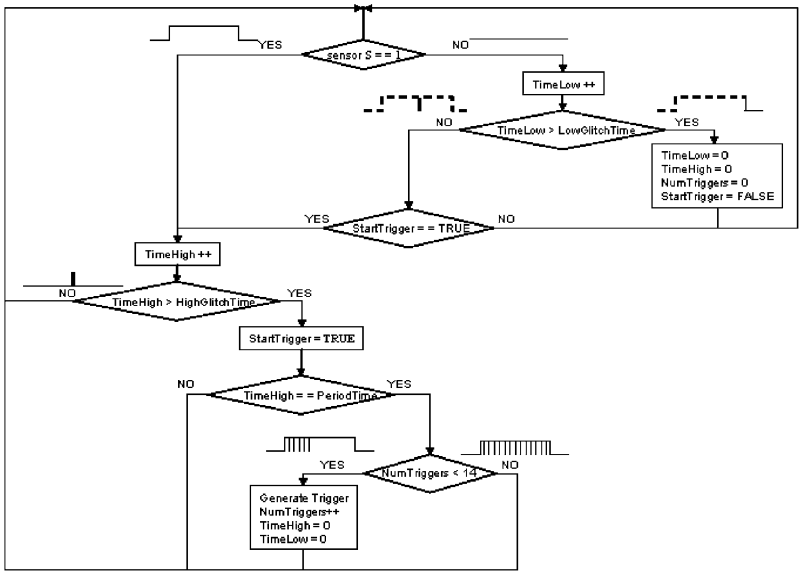

Fig. 7. Digital filter implementation.

signal can still reach the receiver (because of a gradation in the refraction indexes due to the layer of water followed by the layer of glass that reduces the amount of reflected light and increases the transmitted one). The reception of this signal interrupts for a while the high-level of the sensor signal. We call this spurious low-level signal a *LowGlitch*. It can also occur that the sensor activates high although there is no TV screen in the system due to some electronic noise. We call this spurious high-level signal a *HighGlitch*. In order to avoid it, a digital filter has been implemented . This filter samples the sensor signal every time that a pulse of the encoder occurs. If it detects that the level is high for a period of time longer than what is considered to be caused by an error, $HighGlitchTime = 75$ encoder signals, it is assumed that there is a TV screen. In this case, a trigger signal is generated after $PeriodTrigger = 150$ encoder periods if the number of already generated triggers, *NumTriggers*, is less than 14. In order to avoid the influence of a drop of water, it is verified that the time while the signal is low be longer than what is considered to be caused by a drop of water, $LowGlitchTime = 140$ encoder signal, before it is decided that the TV screen is no longer in the system. The result of filtering can be seen in Fig. 8. In this figure, the first and last signals are $S1$ and $S2$, respectively; and the second and third are $T1$ and $T2$, respectively. It can be seen that, although spurious have occurred, the output signals have been correctly generated and their periodicity has been preserved.

4.3. Modular processing system

The complete processing system is composed of five computers: one of them is called the master and the other ones are called slaves. The master is responsible for the interactions among the system, the user and the I/O board; it also administrates the communication among the slaves. Slaves deal with acquisition and processing tasks. Each slave acquires the images of two cameras and processes the data. Therefore, three types of processes are distinguished:

- a master process, which executes in the master computer;
- a slave process, which executes in each slave computer;
- a processing process, which executes in duplicate in each slave (slaves computers are dual processor).

Two types of communication have been defined to exchange information among the processes:

- inter-computer communication;
- intra-computer communication.

Inter-computer communication is established between the master and each one of the slaves to synchronize all the computers and to exchange processing results. Its implementation follows a master–slave real-time design pattern, as defined in [16]. It is carried out via Ethernet, so sockets are used. Intra-computer communication allows that processes executing in the same computer share information. In this case the information is the images and the processing results. Shared memory and pipes are used. Shared memory is the best option to move a large sum of data between processes in the same computer because it is fast if the access is not concurrent, as it is in our case. The acquisition module places the images in a shared memory and the two processing sub-modules pick them up to start their

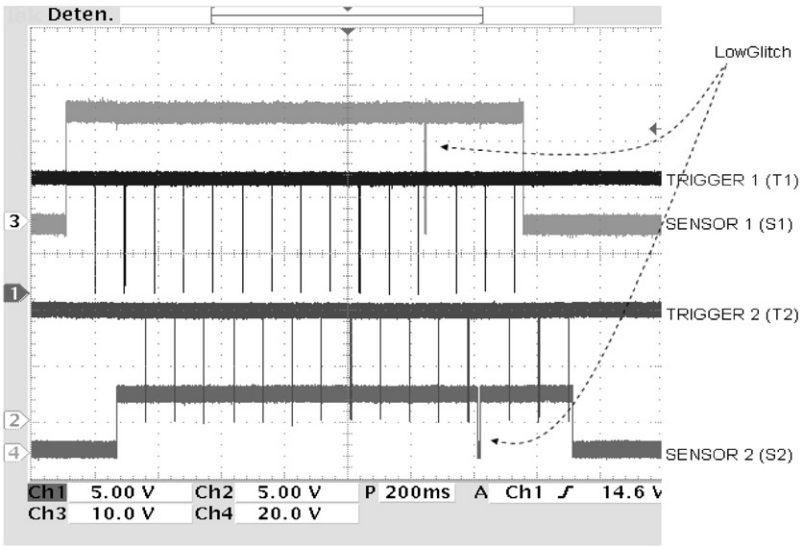

Fig. 8. Result of digital filtering.

processing. A pipe gives a duplex communication between processes. When the process is finished the processing sub-module sends the processing results via a pipe.

4.4. Real-time processing

Processing is carried out under Windows NT 4.0 Workstation operating system plus Service Pack 6a. This operating system cannot be considered as a real-time one [8,17]. However, as far as it satisfies the timing requirements of the application, it can be said to be a soft real-time operating system. The relevant characteristics of the operating system that make it suitable for our real-time application are twofold: on one hand, it is a multitasking operating system, on the other, it is possible to assign priorities to the user-defined processes.

The application benefits from this real-time characteristics using a Dual processor and defining the processing sub-modules as REALTIME-PRIORITY-CLASS processes.

This configuration is able to provide a mean processing time around 2556 ms. This time includes the complete processing of 14 images to detect the kinds of defects shown in Fig. 4, that is 182 ms/image to detect four types of defects with much variability. Processing includes contour detection, morphological operations, binarization, blob analysis, Fourier transformations and filtering. The result is transferred to the slave process which merges the results from both processing processes and communicates them to the process in the master PC. Additional time is required for merging the results and for the communications. Fig. 9 shows the distribution of time when the processes are executed 6850 times using (a) processes and (b) threads, in the following system (this is the configuration installed in the manufacturing company):

- Motherboard: *SuperMicro P6DGE/DBE* (*Dual Motherboard*),
- BIOS: *AMIBIOS rev. 3.1*,
- Processor: *Intel PIII* 600 *MHz* 512 *Kb Cache x2*,
- Memory: *SDRAM* 256 *Mb PC-100 Viking*,
- Hard Disk: *8,2Gb (ST38410A-Seagate)*,
- VGA Card: *Matrox Millenium G400* 16 *Mb* (*AGP*),
- Frame-grabber Card: *Matrox Meteor II Mc/4 on PCI Slot 1* (30 *Mb memory allocation on Host System*),
- Ethernet Card: *3Com 905B-TX FastEthernet 10/100 Mb*,
- Operating System: *Windows NT 4.0 WorkStation + Service Pack 6*(*A*).

It can be seen that the execution time in both processes is slightly different due to the operating system tasks being carried out in one of the processors. However, it compares favorably with respect to the execution time required by the threads.

Fig. 10 shows the distribution of time required to execute the same processes as the ones shown in Fig. 9a when faster processors are used in a single processor PC. Therefore, as the processor's speed improves, additional modules can be included in a single computer.

5. Conclusions

We have defined a modular and scalable architecture for PC-based real-time vision system that has been exemplified with a successful application already installed in a manufacturing company. This architecture encompasses both the acquisition and the processing system.

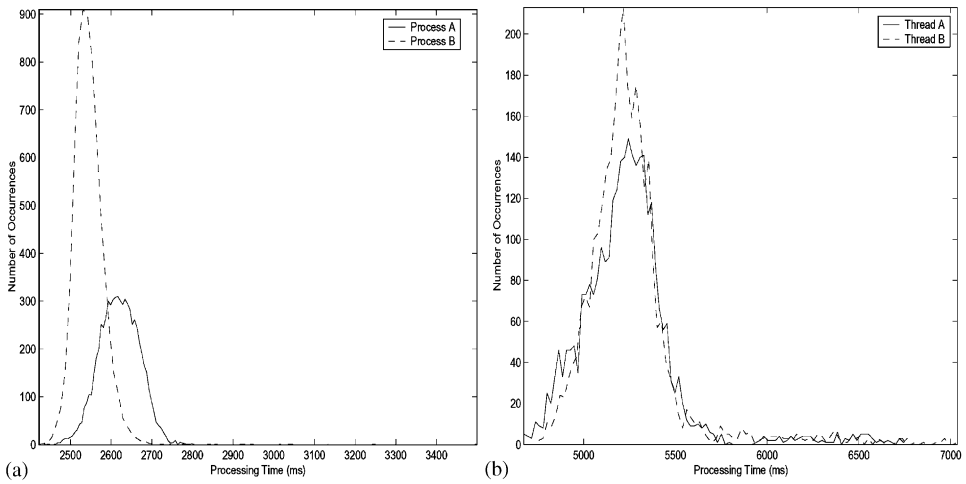

Fig. 9. Distribution of the execution time of (a) 2 processes and (b) 2 threads, using a dual processor PIII-600 MHz.

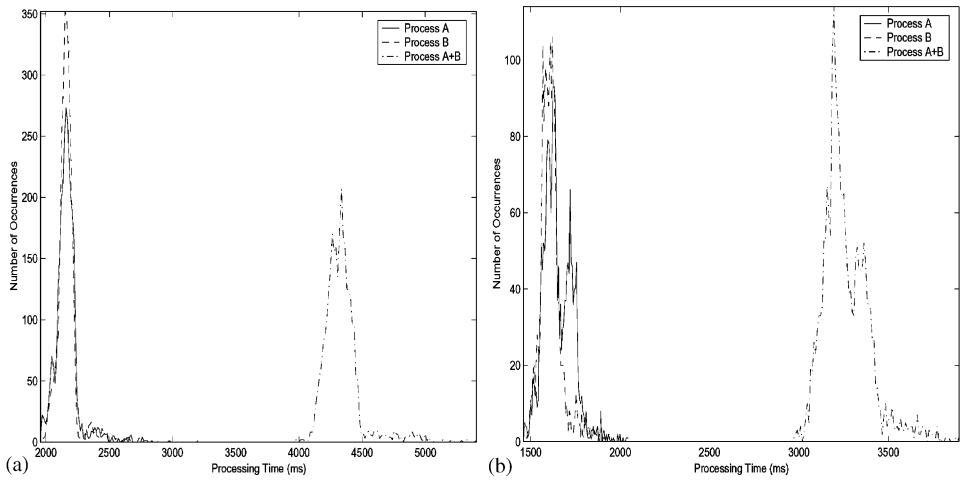

Fig. 10. Distribution of the execution time of two sequential processes using (a) a single processor PIII-800 MHz and (b) a single processor PIV-1.8 GHz.

The acquisition system is composed of acquisition modules. Each module is composed of a group of up to three monochrome cameras and one frame-grabber. The scalability applies to both the number of cameras that compose the module and the number of modules that compose the acquisition system. This system is designed so that it may be possible to simultaneously acquire the images from all the cameras and all the modules if they reside in different slave computers. The acquisition is triggered by an external signal when the object is located in the camera's field of view. Therefore, acquisition is a hard real-time operation in the sense that unexpected delays in the triggering instant can make the acquisition system to fail (the object can be out of the field of view of the camera). These delays can be caused by latencies, that is, uncertainties in the response time of the system's components or processes. Since the response uncertainties of hardware circuitry are much smaller than software's response uncertainties associated with the operating system, the triggering signal is better generated by dedicated circuitry (either included in the frame-grabber or specially designed for the application). An I/O board residing on a PC that runs Windows NT 4.0 (or similar non-real-time multitasking operating systems) cannot guaranty a deterministic trigger because a user-level management of the I/O board does not allow using the highest priority tasks deferred to interrupt routines. All user-defined tasks can be preempted by this higher priority tasks. A specific circuitry that filters and manages the signals involved in the acquisition system for real-time applications has been designed.

The processing system is composed of a set of modules and sub-modules: the master module, the slave module and the processing sub-modules. Slave module and processing sub-modules run in a slave computer. The slave computer is a multi-processor PC (a dual processor, in most cases) and contains the acquisition module. Scalable processing can be accomplished using multiple processors and a multitasking operating system. The master module runs in a master computer. Each module is implemented as a process. Processes are preferred over threads because of memory considerations. Contention on critical sections is a major problem in a multi-processing system because of additional system calls and context switches. Additionally, the existence of one memory cache per processor needs to be taken into account. Each processor's memory cache must maintain a consistent view of the main memory. This is accomplished by dividing memory into cache lines. If the same cache line is continuously updated from many different processors, that cache line will bounce from one processor's cache to another. Therefore, processes are preferred because they have their private address space and no special consideration has to be taken into account with respect to the memory data structures of the application's variables in order to avoid contention on memory access due to existence of multiple memory caches. Inter-computer communication is carried out between modules by means of sockets. Intra-computer communication between each processing sub-module and the slave module is carried out by means of pipes. The slave module and the processing sub-modules share the information provided by the acquisition module by means of a file mapping. Whenever a new image is available, the corresponding processing sub-module is signaled by means of an event. Processing uncertainties are minimized as much as possible by means of assigning real-time priority to the processes and avoiding the installation of unnecessary drivers.

This architecture is generalizable to many applications since there exist common characteristics of the real-time computer vision applications that allow defining a general modular architecture for the acquisition and processing systems.

References

[1] Meribout M, Nakanishi M, Ogura T. Accurate and real-time image processing on a new PC-compatible board. Real-Time Imaging 2002;8(1):35–51.

[2] Ranganathan N, Sastry R, Venkatesan R. Smac: a VLSI architecture for scene matching. Journal of Real-Time Imaging (Special Issue on VLSI for Image Processing) 1998;4(3):171–80.

[3] Wiatr W. Median and morphological specialized processors for real-time image data processing. EURASIP Journal on Applied Signal Processing 2002;2002(1):115–28.

[4] Timmerman M, Monfret J-C. Windows NT as real-time OS. Real-Time Magazine 1997;2:6–14.

[5] Yasu Y, Carcassi G. Evaluation of a real-time extension (RTX) on Windows/NT, Atlas DAQ. Available from http://citeseer.nj.nec.com/171536.html (April 1999).

[6] Burke MW. Image acquisition. London: Chapman & Hall, 1996.

[7] Demant C, Streicher-Abel B, Waszkewitz P. Industrial image processing. Berlin: Springer, 1999.

[8] Microsoft Corporation. MSDN home. http://msdn.microsoft.com/.

[9] Regehr J, Stankovic JA. Augmented CPU reservations: towards predictable execution on general-purpose operating systems. In: Proceedings of the Seventh IEEE Real-Time Technology and Applications Symposium, Taipei, Taiwan, 2001. p. 141–8.

[10] Baril A. Using Windows NT in real-time systems. In: Proceedings of the Fifth IEEE Real-Time Technology and Applications Symposium, Institute of Electrical and Electronics Engineers, Inc., 1998. p. 132–41.

[11] Cota-Robles E. A comparison of Windows driver model latency performance on Windows NT and Windows 98. In: Proceedings of the Third Symposium on Operating Systems Design and Implementation, New Orleans, LA, 1999. p. 159–72.

[12] Jones MB, Regehr J. The problems you're having may not be the problems you think you're having: results from a latency study of Windows NT. In: Proceedings of the Seventh Workshop on Hot Topics in Operating Systems (HotOS VII), IEE Computer Society, Rio Rico, AZ, 1999. p. 96–102.

[13] Malina R. Windows NT for soft real-time control. White Paper, Rockwell Automation. Available form http://quite.teknowledge.com/WorkingGroups/ (1997).

[14] Ramamritham K, Shen C, Gonzalez O, Sen S, Shirgurkar SB. Using Windows NT for real-time applications: experimental observations and recommendations. In: Proceedings of the Fourth IEEE Real-Time Technology and Applications, Denver, CO, 1998. p. 102–11.

[15] West P. High speed, real-time machine vision. Machine Vision Online. Available from http://www.imagenation.com/ (2001).

[16] Douglass BP. Real-Time UML: developing efficient objects for embedded systems, 2nd ed., The Addison-Wesley Object Technology Series. Reading, MA: Addison-Wesley Longman, 2000.

[17] Microsoft Corporation. Real-time systems and microsoft Windows NT, MSDN Library. Available from http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dndllpro/html/msdn_realtime.asp.