Journal of Electronic Imaging

SPIEDigitalLibrary.org/jei

Software architecture for time-constrained machine vision applications

Rubén Usamentiaga
Julio Molleda
Daniel F. García
Francisco G. Bulnes

Software architecture for time-constrained machine vision applications

Rubén Usamentiaga
Julio Molleda
Daniel F. García
Francisco G. Bulnes
University of Oviedo
Department of Computer Science
Campus de Viesques, Gijón 33204 Asturias, Spain
E-mail: rusamentiaga@uniovi.es

Abstract. *Real-time image and video processing applications require skilled architects, and recent trends in the hardware platform make the design and implementation of these applications increasingly complex. Many frameworks and libraries have been proposed or commercialized to simplify the design and tuning of real-time image processing applications. However, they tend to lack flexibility, because they are normally oriented toward particular types of applications, or they impose specific data processing models such as the pipeline. Other issues include large memory footprints, difficulty for reuse, and inefficient execution on multicore processors. We present a novel software architecture for time-constrained machine vision applications that addresses these issues. The architecture is divided into three layers. The platform abstraction layer provides a high-level application programming interface for the rest of the architecture. The messaging layer provides a message-passing interface based on a dynamic publish/subscribe pattern. A topic-based filtering in which messages are published to topics is used to route the messages from the publishers to the subscribers interested in a particular type of message. The application layer provides a repository for reusable application modules designed for machine vision applications. These modules, which include acquisition, visualization, communication, user interface, and data processing, take advantage of the power of well-known libraries such as OpenCV, Intel IPP, or CUDA. Finally, the proposed architecture is applied to a real machine vision application: a jam detector for steel pickling lines.* *[DOI: 10.1117/1.JEI.22.1.013001]*

1 Introduction

The design and implementation of real-time image and video processing applications has always been a complex task.[1] However, due to recent trends in the hardware platform, such as a steady increase in computing power based on parallelism and the improved resolution and image acquisition rate of low-cost imaging devices, the development of applications requires increasingly skilled architects. For example, current camera interfaces provide high bandwidth which is used with sensitive sensors to increase the image resolution and the number of frames per second acquired by the cameras. This increase of information can only be processed in real time by making an efficient use of parallel execution resources,[2] a task that requires the skills of a highly trained architect.

Designing, developing, and tuning the applications to meet real-time constraints require expertise in different areas, such as parallelism, computer architecture, and image processing. Many frameworks and libraries have been proposed or commercialized to reduce the complexity of these tasks. For example, the flow scheduling framework[3,4] proposes an architecture in which applications are built using a data-flow model, where each processing node has a fixed number of inputs and outputs, each of a given data type. Similar architectures have been proposed recently.[5–7] These works confront the problem from different perspectives depending on the type of application on which they are based.

This paper will address the issues raised in previous research in this area. In general, these works do not present flexible architectures, because they are normally oriented toward a particular type of application, such as video surveillance or robotics. Also, they often impose specific data processing models that are not optimal for all types of applications. In most cases, it is assumed that the data processing model must follow a pipeline or an architecture based on Pipes and Filters,[8] without providing an option for other models. In addition, data processing models are static, preventing run-time reconfiguration if necessary. Another aspect that is not treated in sufficient depth is memory management. To avoid synchronization problems, many previous works are based on expensive data copies of information stored in memory. In other cases, they introduce memory managers that are complex to use and specific for certain types of applications. Modern dynamic memory management should address these issues. Scalability in modern hardware is another open issue. Some previous works focused on distributed systems. Others assumed an execution model based on a thread of execution per task, which significantly reduces the scalability of the system.

This paper presents a novel software architecture for time-constrained machine vision applications that addresses these issues. The goal of this work is not to recode existing image processing algorithms, but to make it possible to integrate the available options, such as OpenVL,[9,10]

Paper 12279P received Jul. 25, 2012; revised manuscript received Nov. 12, 2012; accepted for publication Dec. 6, 2012; published online Jan. 3, 2013.

OpenCV,[11] Integrating Vision Toolkit,[12] or ITK[13] in a flexible architecture.

The proposed architecture is divided into three layers: the platform abstraction layer, the messaging layer, and the application layer. The platform abstraction layer provides a high-level application programming interface for the rest of the architecture. This layer simplifies the development of the architecture and makes it operating system independent. The messaging layer provides a message-passing interface based on a dynamic publish/subscribe pattern. A topic-based filtering process in which messages are published to topics is used to route the messages from the publishers to the subscribers interested in a particular type of message. Messages can be as flexible as possible. Also, different mechanisms are used to avoid expensive data copying, and policies are provided to execute publishers and subscribers efficiently in parallel. The application layer provides a repository of reusable application modules designed for machine vision applications. These modules, which include acquisition, visualization, communication, user interface, and data processing modules, take advantage of the power of other well-known libraries, such as OpenCV, Intel IPP, or CUDA.

2 Requirements, Attributes, and Design Principles

Based on our experience designing and implementing machine vision applications,[14–17] the most important nonfunctional requirements of a real-time image processing architecture are the following:

- Scalability: The architecture must adapt to the available resources. Thus, as emerging platforms with more computational resources become available, image processing systems with higher resolution and speed may be used. This quality attribute has been addressed in previous works,[18–20] although more frequently in distributed systems than in multicore systems.
- Reusability: One of the main objectives of this architecture is that the work serves as a basis for different types of applications. The modules, the communication framework, and the parallel execution models may be reused in the future to add functionalities with no modification.[21–23]
- Flexibility: Not all applications are equal, nor will they have the same functional and nonfunctional requirements. The design of the architecture should be flexible enough to adapt to the needs of different applications.
- Extensibility: Requirements change; they are modified or mute. The design of the architecture must be extensible to take into account any new requirements that may arise.
- Reliability: This is a basic property of any system. However, in the case of real-time systems that can control and supervise critical processes, this property is vital. The architecture must be robust and resilient to failure and errors.

One of the most important recent trends in computer architecture is the steady increase in the number of cores of the CPU. Physical constraints preventing frequency scaling have conditioned the evolution of modern low-cost computers, creating CPUs with multiples cores that make them capable of running programs in parallel. Real-time applications are required to make efficient use of computer resources. Thus, they must be prepared to execute tasks in parallel. The proposed architecture must be flexible enough to deal with different types of parallelism. Three main types are considered:

- Data parallelism: This is the most common type of parallelism found in image processing applications. It consists of dividing the image into parts and processing each one independently. This type of division is not possible for all image processing algorithms, but it is very easy to apply for most of them.
- Control parallelism: In this case, the process that needs to be applied to data is divided, rather than the data itself. In this way, different subprocesses can be applied at once.
- Flow parallelism: In this case, the process applied to the image is decomposed into several stages that are applied sequentially. Two different stages can run in parallel. This way of parallelizing the processing is often called pipeline.

All these properties and requirements are considered when applying the following basic design principles:

- The architecture must be open for extension but closed for modification (open/closed principle).[24] Therefore, it must be possible to extend the architecture in response to new or changing requirements without modifying the existing source code.
- There must be strong abstraction between the interface of the architecture and the implementation. Therefore, they are not dependent and can vary independently.
- The architecture must follow a strict modular design. This way, reusability, legibility, and maintainability are greatly improved. Requirements change, merge, emerge, and mutate. The orthogonality of the modules not only provides a method to deal with these changes without affecting the whole system, but it also provides the foundations to treat the capabilities of each module independently.

The architecture must provide an application programming interface for computer vision applications in real-time. This programming interface must be easy to learn and use and, simultaneously, hard to misuse. Additionally, the code that uses the architecture must be legible and maintainable.

The architecture must be designed for real-time applications. However, the term "real-time" can be confusing, as it can have different interpretations. In general, a real-time system can be defined as one whose logical correctness is based both on the correctness of the outputs and their timeliness.[25] Thus, predictability, not speed, characterizes a real-time system. This definition in the software engineering sense is further classified based on the strictness attached to the deadline as hard real-time, firm real-time, or soft real-time. In this work, the term "real-time" is interpreted in the signal processing sense, that is, based on the idea of completing the processing in the time available between successive input samples.[1]

3 Architecture Design

3.1 *Overview*

The proposed architecture follows a model based on publishers and subscribers of information. This type of architecture is often called Publisher/Subscriber[26] and is related to the observer design pattern.[27] While other architecture models are possible, such as the MVC model, they are oriented toward certain types of applications and when the roles are well defined. Therefore, in these cases, the flexibility is much lower.

In order to manage the information interchange in the proposed model, an intermediate element is introduced: the broker. The publisher posts information to the broker, and the subscriber registers with that broker to receive information, as can be seen in Fig. 1.

A subscriber is not interested in receiving all the information published—only a specific part. To filter the information, the concept of topic is introduced. A topic represents a type of information to which a subscriber can be registered or in which a publisher can publish. The broker maintains a data structure with all the topics registered in the system, as well as the list of subscribers registered to each topic, as shown in Fig. 2. As can be seen, a subscriber may be interested in different topics. This interest can be established during the initial registration or later during the execution of the application, making the entire subscription process dynamic and therefore enabling run-time reconfiguration if necessary.

This type of architecture has great advantages. It is a loosely coupled architecture. Publishers are completely decoupled from subscribers; they are not even aware of each other's existence. Therefore, the robustness and the modularity of the application are greatly improved; each module can be implemented and tested independently from others. This approach also helps extensibility and reusability, and the architecture provides the opportunity for scalability. The execution of each publisher and subscriber in the application is independent, which makes the execution of these modules in parallel possible.

The messaging layer of the proposed architecture benefits from all the advantages of the Publisher/Subscriber model. Furthermore, there are several areas where we introduced improvements to the original scheme:

- Smart memory management: Memory management is automatic and very efficient, always avoiding expensive data copies. The proposed architecture includes efficient techniques to avoid memory micromanagement, making it easy use and hard to misuse.
- Decoupling of the subscribers from the information: In order to manage the execution of the subscriber effectively, different type of subscribers are defined. Depending on the type, the behavior of the subscriber changes without affecting the real subscriber, implemented in higher levels, greatly increasing flexibility and extensibility.
- Automatic parallel processing environment: The architecture is designed to enable the parallel execution of the subscribers efficiently, creating a dynamic mapping between execution modules and execution units.
- Dynamic reconfiguration: The subscription model and the creation of topics is completely dynamic. Thus, during the execution of the system, new subscribers can be added or removed. Also, subscribers can change their subscriptions or create and remove topics. The reconfiguration is finely synchronized to avoid expensive thread locks.

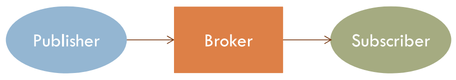


Fig. 1 Main elements of the proposed architecture: publisher, subscriber, and broker.

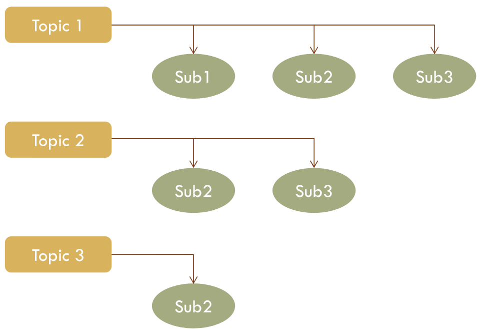


Fig. 2 Organization of topics in the broker.

3.2 *Publisher*

The publisher is only a theoretical concept. Information can be published from any point of the application without restrictions. In general, most subscribers will not only receive information, but also publish information.

3.3 *Broker*

The broker is in charge of delivering the information published to the interested subscribers. In order to receive information, subscribers must define a callback function using an object-oriented style. Compiler time errors will arise if this callback is not properly defined.

The broker maintains several data structures with information about the topics and the subscribers registered to each topic. It also contains a public interface that can be used to register or unregister topics and subscribers and to publish information. These operations are efficiently synchronized so that they can be used from multiple threads simultaneously. Also, the broker allows changes in the topics and the subscribers while the system is running, enabling hot swapping and plugging.

3.4 *Messages*

A message is used to encapsulate the information. Messages are as flexible as possible. Any data type of the language can be a message, although the system is strongly typed. Thus, if the data type of the published message is not that expected by the subscriber, an error occurs. The subscriber can also query the data type of the message, performing different actions with different content.

Memory management for messages is automatic and very efficient, always avoiding expensive data copies. Memory management is implemented using smart pointers and multithread synchronization.[28] This method to manage memory is not only very efficient, but it also avoids memory micromanagement, which often appears in programing languages without a garbage collector. When a message is published to a topic, every subscriber registered to that topic receives a copy of the smart pointer to that message with read-only access. This copy is $O(1)$ and has negligible impact on the performance of the system. A reference counter is incremented each time the smart pointer of the message is copied. Later, when subscribers finish using the message, the reference counter is decremented. When the last subscriber finishes using the message, the reference counter reaches zero, and the message is automatically destroyed. Also, read-only access for the published messages is a very effective approach, as it avoids shared memory contention and the risks of deadlock and race conditions.

3.5 *Subscriber*

The basic operation of a subscriber is the reception and processing of information. The subscriber can also publish the results. However, in order to manage the execution of the subscriber effectively, it is necessary to define different types of subscribers:

- Sync: This type represents a synchronous subscriber, where the publisher calls the subscriber directly. Thus, the publisher waits for the subscriber to finish processing the information. This type is designed for subscribers that perform fine-grained work, where the overhead of awaking a thread to execute the callback is not worthwhile.
- Async: This type represents an asynchronous subscriber, where the publisher continues execution immediately after publishing the information without waiting for the subscriber to finish processing it. This behavior is implemented using a synchronized queue, where messages are stored for later processing by the subscriber following a producer-consumer pattern.[29] This is one of the most efficient patterns for parallel processing. Depending on the particular implementation of the message queue and the way the subscribers access the queue, different variations of the asynchronous subscriber are available:
    - AsyncTask: In this type of subscriber, there is a specific thread for each subscriber that gets blocked until some data is available in the queue. When a message is published and stored in the queue of the subscriber, the subscriber thread awakes and notifies the subscriber. This type of subscriber represents a model that is not scalable, since the creation of a thread for each subscriber would represent unnecessary overhead in the system. This type of subscriber only exists to allow the execution of subscribers that require thread affinity. For example, a subscriber that is responsible for displaying information in a window always needs to be executed by the same thread, because this thread is the owner of the window message queue created for interaction between the system and the window.
    - AsyncPool: In this type of subscriber, there is no specific thread for each subscriber. In order to notify the subscriber, a precreated thread from the thread pool of the system is used. This subscription model is fully scalable, as the number of threads in the pool is created based on the number of cores in the machine independently of the actual number of subscribers. This way, the architecture performs an efficient dynamic mapping between execution modules and execution units, which is fundamental for system scalability. Under this execution model, mapping is performed according to the number of available execution units.

The type of subscriber is specified as a wrapper of the real subscriber, with the same interface. It acts as an intermediate layer between the broker and the real subscriber, as can be seen in Fig. 3. However, the type of subscriber is completely independent from the broker and from the real subscriber. When the broker delivers the message to a subscriber, it does not know the type; it just notifies the subscriber through the registered callback. Nor does the real subscriber know its type; it just defines the callback without worrying about who is executing it and how. In fact, one subscriber can be registered with any type. This approach greatly increases flexibility and extensibility using the principle of open for extension and closed for modification. The addition of a new type of subscriber does not in any way alter the rest of the architecture or the application. For example, there are multiple implementations of the AsyncPool subscriber that can be used without affecting the application.

The message queue in the asynchronous subscriber is a very important piece of the architecture. This component can be parametrized for each subscriber, making the definition of different policies possible. For example, one of the parameters of the message queue is the size. This parameter can be used to compensate for the different rates at which the flow of data is published and processed. The message queue decouples the subscriber from the information. This can also be used to specify maximum notification rates. For example, a subscriber that displays the acquired images on the screen does not need to be updated at the same frame rate as the rest of the subscribers. Thus, a maximum update period can be established. Also, different handlers can be installed to deal with the reception of messages when the message queue is full. Many options are available: skip the message, wait until there is room in the queue, substitute the older message, or run an *ad hoc* handler.

The design of the architecture and the parallel execution model introduces parallelism at the highest level possible. This approach avoids the inefficiencies that appear when parallelism is expressed at very low levels, making it difficult to amortize the runtime overhead.[30]

The proposed architecture also has maximum flexibility. A designer can very easily create a classic execution model such as the pipeline, where the stages are executed by a thread in the thread pool with improved scalability. However, the design of other types of execution models

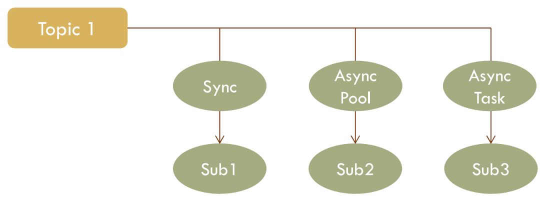


Fig. 3 Types of subscriber.

with data or control parallelism is also straightforward, depending only on the way the subscribers are connected.

3.6 *Software Platform*

The software architecture is fully developed in modern C++, which is by far the most widely used language for implementing machine vision applications. The choice was clear: C++ is a language for programming based on lightweight abstraction with direct and efficient mapping to hardware, making it perfectly fit for real-time applications. Principles of object-oriented design patterns have been applied to aid in the development of the architecture.

3.7 *Layers*

As can be seen in Fig. 4, the proposed architecture is divided into three layers: the platform abstraction layer, the messaging layer, and the application layer. The layers are outlined in the following sections.

3.7.1 *Platform abstraction layer*

The platform abstraction layer provides a high-level application programming interface for the rest of the architecture. This layer abstracts operating system services and also includes modules required for higher layers. Some of the modules available in this layer include the shared queue, smart pointer, thread, thread pool, synchronization modules, and different types of sockets. It also provides a unified way to handle errors using exceptions. Using this layer, the development of the architecture is greatly simplified. High-level layers are also independent of the operating system.

3.7.2 *Messaging layer*

The messaging layer provides the message passing interface based on a dynamic publish/subscribe pattern. The broker and all the data structures necessary to store information about the subscribers and topics are included in this layer. It also includes the implementation of the available types of subscribers and all the base classes required to create new subscribers.

Up to this layer, nothing is specific for image-processing applications; the messaging layer can be used to create any type of application. Applications can be placed above this layer using the proposed framework for information distribution and routing using the automatic parallel processing environment.

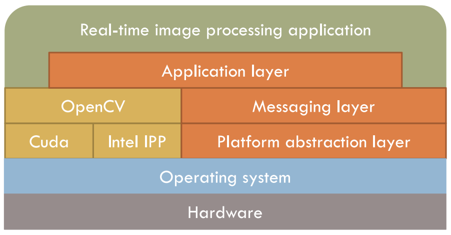


Fig. 4 Layers of the architecture.

3.7.3 *Application layer*

The application layer is composed of a repository of reusable modules that implement the most common tasks found in an image processing application.

The first module implemented is the Image class. This class encapsulates the OpenCV image structure (*IplImage*), adding automatic memory management. Automatic casting is implemented, so an Image object can be used in any OpenCV function. The Image class is also prepared to work smoothly with the new image structure used in OpenCV 2 (*cv::Mat*), allowing the applications to work with any version of OpenCV. Using OpenCV for the architecture was the most adequate choice, as it is the most widely used library for real-time computer vision applications, it is well maintained, and it can be integrated with other low-level libraries, such as CUDA or Intel IPP, greatly improving the performance of the applications.

One of the most important modules is the image acquisition module. It acquires images from a generic source of images and publishes them to a specific topic. The source of images allows the image acquisition module to work with different types of sensors and image sources. The following sources are included: *GigECam*, *DirectShowCam*, *VideoFile* and *NetworkCam*. The first two sources are used to acquire images from cameras with a GigE Vision interface and from cameras with a generic DirectShow interface such as WebCams. The *VideoFile* source is used to acquire images from a video file, which is a standard approach used during testing. Support for many types of containers and codecs, such as avi, mkv, mpg, xvid, or h264, is included. The *NetworkCam* can be used to acquire images coming from the network. This source of images can be used to create distributed applications. Any other type of source could be implemented without altering the image acquisition module or the application using the published images.

In addition to *NetworkCam*, two other network modules have been included in the architecture: *NetPublisher* and *NetReceiver*. These modules can be used to send and receive any message published through the network. Therefore, these modules pave the way to transforming a parallel application using shared memory into a parallel application using distributed memory. Nor does this transformation affect the subscribers or the rest of the architecture, which is an indication of the flexibility of the proposed architecture.

Many other miscellaneous modules have been included in the application layer. The *VideoStream* module can be used to convert a flow of images into a RTP flow codified in h264. The *VideoWriter* can be used to write images to disk using virtually any container and format. The *VideoDisplay* shows images in a window based on DirectX. The *SubProfiler* can be used to profile subscribers. The *Serializer* is used to serialize information. The *CgiPubSub* can be used to connect the application to a Web application for display or control purposes. Many more modules are available for other purposes. Each module is independent and can be used to add functionality to the application.

3.8 *Overhead*

In order to measure the overhead introduced by the architecture, an experiment has been carried out comparing a direct function call with a subscriber that calls the same test function when a topic is published. The test function is very simple, and it performs some random calculations (with optimizations disabled).

First, a test program calls the function 100 times. The elapsed time for each call is calculated using the high-resolution performance counter (using the functions *Query Performance Frequency* and *QueryPerformanceCounter*). Next, another test program publishes a topic the same number of times. The same test function is called in a subscriber each time the topic with empty data is published. After the call to the test function, the subscriber publishes another topic. The program also measures the elapsed time from the moment the first topic is published until the moment the second topic is published. This measurement includes the time taken by the test function and the overhead introduced by the architecture.

In order to measure the time taken by the subscriber, the *SubProfiler* was used. This module of the application layer of the architecture measures the time elapsed from when a first topic is published until a second topic is published. Thus, it is a very useful method to measure the time used by any subscriber that receives information in a topic and publishes the results in others. The *SubProfiler* records the elapsed time and calculates statistics that can be published, generally to high-level modules in an application.

The results indicate that the proposed architecture has a very low overhead. The differences between the direct function call and the call through a subscriber were around 100 μs on a hexa-core Intel Xeon 5620 running at 2.4 GHz. Figure 5 shows a box-and-whisker plot with the time taken by the direct function call and by the subscriber.

3.9 *Parallel Performance*

The parallel performance of the architecture has been tested with a basic application that detects edges in color images. The image is split into three planes: red, green, and blue. Then, the canny edge detector is applied to each plane. Finally, the resulting edge maps are combined into a single edge map.

The aplication was first designed sequentially using OpenCV functions (*cvSplit*, *cvCanny*, and *cvMerge*). The results indicate that, for the Lena image with a resolution of 256×256, the sequential version was able to achieve

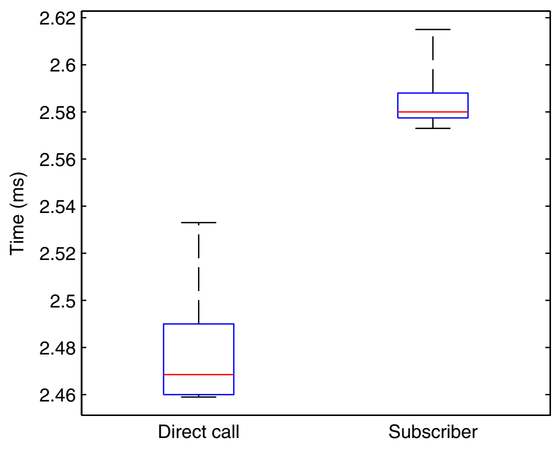


Fig. 5 Overhead introduced by the architecture.

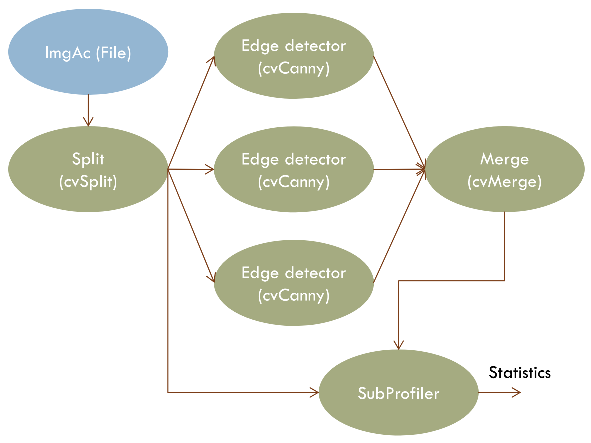


Fig. 6 Organization of the parallel performance test.

215.88 fps on a hexa-core Intel Xeon 5620 running at 2.4 GHz.

The same application was designed using the proposed architecture, as can be seen in Fig. 6. The same functions were used, but in this case, the detection of edges for each color plane is carried out in parallel branches. The *SubProfiler* module is used to obtain statistics about the execution. The application using the proposed architecture with the same image and running on the same machine was able to achieve 520.29 fps. This is an increase of ×2.41. A linear speedup is not achieved due to the sequential parts of the application. However, this result indicates that the architecture is using the resources of the machine efficiently.

3.10 *User Interface in Applications*

A very important aspect of any application is the user interface. A set of modules that serve to facilitate the development of such interfaces has been included in the repository of reusable modules. The user interface must allow users to interact with two elements: video and application configuration. Since we are working with image processing, the user should be able to see the images in real time. This is where *VideoWriter* is used. The user must also be able to send commands to the system and receive various types of information. The module *CgiPubSub* is used for these purposes.

The platform selected for developing the user interface is the Web, following the current technological trends. This platform has a major advantage: It can be used by any user without requiring the installation of additional software. It also allows multiple users to interact with the system concurrently. Within Web technology, we have opted to use modern open technologies, specifically Javascript, Ajax, and HTML 5.

Figure 7 shows a possible configuration for an edge detection application created using the proposed architecture. The main application acquires images from a file and applies the edge detector in the image processing module. It also publishes the images through the network. In another machine, another application acquires the images coming from the network and creates an RTP image flow in h264, which is sent to a media server running in a third machine. Additionally, the configuration of the application is serialized and sent to a Web server, where the *CgiPubSub* module interacts with the Web application. The resulting application can be seen in

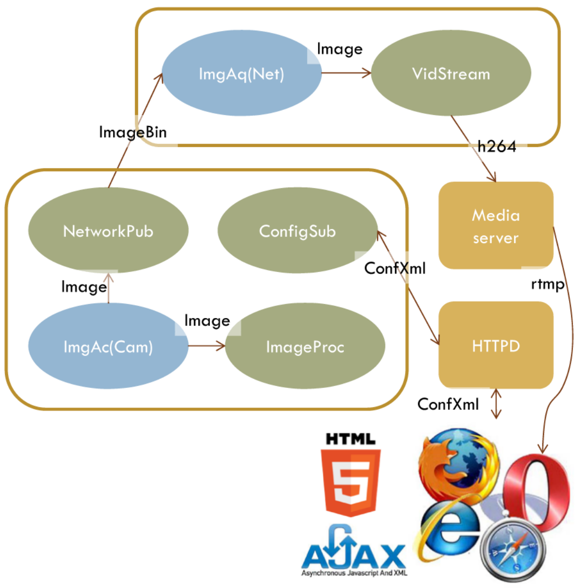


Fig. 7 Updated organization of the edge detection application.

Fig. 8. In this case, three video sources are shown on the screen in real time, including the input image, the resulting edge map, and a camera of the laboratory.

It is worth noting that the user interface modules used in this application are by no means particular to the application. The application only needs to fill a configuration object with information that can be read or written as a list of properties and publish it to a specific topic. This list of properties will be serialized and shown in a Web property grid. Within these properties, there is information about the available video sources, which is used by the interface to connect to the media server and show the video on the browser. When the user changes a property in the Web interface, the information is serialized back to the application and published again with the updated values.

4 Case Study: Jam Detector for Steel Pickling Lines

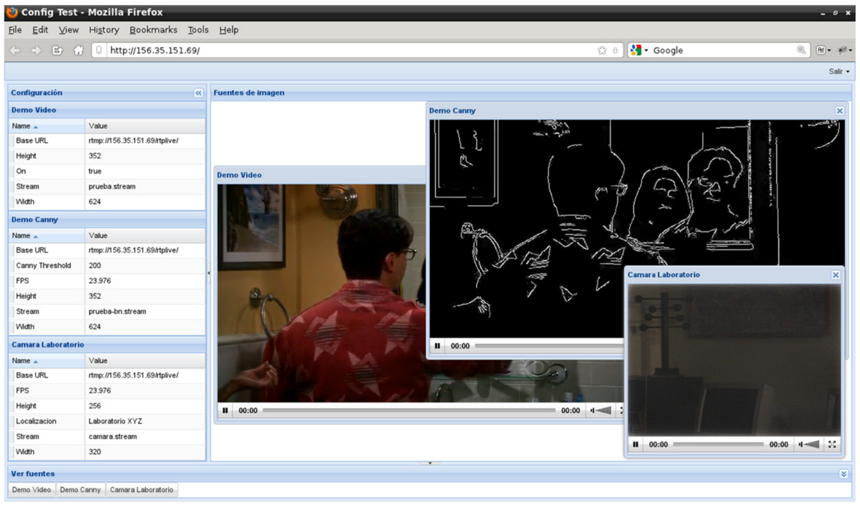
The proposed architecture has been used to create a machine vision application for the detection of jams in steel pickling lines.[31]

Pickling is a crucial step in steel manufacturing that cleans the steel, removing the oxide scale that forms on the steel strips after hot rolling and leaving a clean surface. Also, in a pickling line, there is a side trimmer in which the edges of the steel strip are trimmed by rotary shear knives to provide more uniform width and edge condition. In this process, the steel is side-trimmed to the customer's specifications for width. The resulting wasted material for each edge is chopped into small pieces, which pass through a nozzle to a conveyor belt. The side trimmer is a frequent location of jams in the pickling line. The material cut from each edge can block the nozzle and stop the processing of the steel strip. Industrial processing lines are highly optimized, and jams are not frequent. However, when jams occur, they can have catastrophic consequences on the productivity of the plant if they are not fixed immediately. The objective of the machine vision application is to monitor the side trimmers of a pickling line and to detect jams in real time in order to perform corrective actions quickly.

The proposed approach is based on a machine vision system that counts the number of chopped pieces ejected from the nozzles of the side trimmers. There are two nozzles: north and south, one for each edge of the strip. Detecting the absence of pieces that come out from one of the nozzles is equivalent to detecting a jam in the pickling line. Thus, the machine vision application will acquire images from the nozzles and will count the number of pieces ejected from both of them. Decisions concerning the jam in the pickling line will be based on these numbers.

The proposed approach for jam detection is broken down into several steps. Figure 9 shows the organization of the modules in the application using the proposed architecture.

The application acquires images from a GigE camera. The interface of the camera is a great advantage for industrial applications, as it allows the camera to be installed up to 100 meters from the computer. In particular, the camera used in the system is the Mikrotron EoSens MC1365, which is installed inside a robust aluminum housing to protect the camera from the harsh industrial environment. The EoSens MC1365 is a CMOS high-speed camera with a sensibility of 2.000 ASA and 10 bits per pixel. At full resolution (1280×1024 pixels), 80 fps can be output via the GigE



Demo Video

Name	Value
Base URL	rtmp://156.35.151.69/rtplive/
Height	352
On	true
Stream	prueba.stream
Width	624

Demo Canny

Name	Value
Base URL	rtmp://156.35.151.69/rtplive/
Canny Threshold	200
FPS	23.976
Height	352
Stream	prueba-bn.stream
Width	624

Camara Laboratorio

Name	Value
Base URL	rtmp://156.35.151.69/rtplive/
FPS	23.976
Height	256
Localizacion	Laboratorio XYZ
Stream	camara.stream
Width	320

Fig. 8 User interface of an edge detection application.

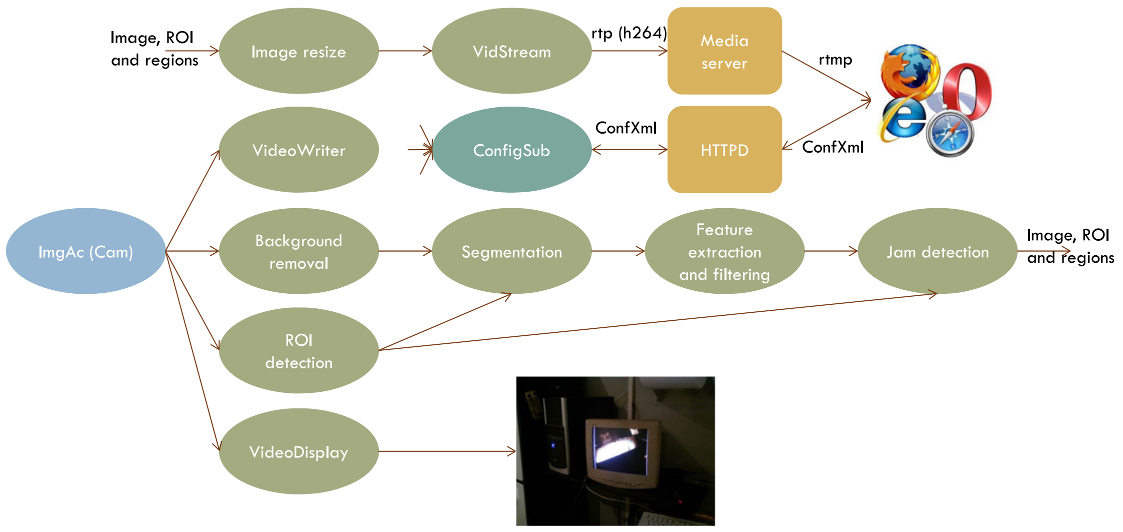


Fig. 9 Organization of the jam detector application.

Vision Interface. The application is using the image acquisition module to acquire images from this camera. Thus, the application is independent from the camera and from the camera interface.

Several modules are registered to receive the published images. The *VideoWriter* creates a video with the acquired images and is used to create a database for offline tests. The *VideoDisplay* shows the images in full screen in the control room of the pickling line, where the computer is installed. This module provides visualization for the technicians monitoring the processing line. These two modules are optional and can be disabled without affecting the rest of the application.

Images are also received by two other modules: the ROI detection module and the background removal module. The ROI detection module identifies the regions of interest for each nozzle. These regions must be identified so that the chopped pieces ejected from one nozzle are not confused with the pieces ejected from the other. This module starts by applying the Canny edge detector. Next, the Hough transform is applied to the edge map. Three lines are detected: the right limit of the north nozzle, the left limit of the south nozzle, and the top limit of the conveyor belt. Using the lines detected by the Hough transform, a new line is calculated: the line that divides the space in the image between the two nozzles. The region of interest for the north nozzle is bounded by the line between the nozzles and the top limit of the conveyor belt. The region of interest for the south nozzle is bounded only by the line between the nozzles.

It is not necessary to calculate the ROI for each acquired image, because the limits of the regions do not change often. Therefore, this module is registered only to receive images once every 10 s. A periodical update of the ROI also makes the system robust against possible movements of the camera caused by accidental bumps.

The background removal module is the first step in the image processing pipeline. This module creates a reference frame, which is a representation of the scene with no moving objects, and subtracts it from each acquired frame. The reference image is estimated using a running average, as it offers acceptable accuracy while achieving a high frame rate with limited memory requirements.[32]

The foreground image obtained from the background removal step is a gray scale image. In order to convert this image to a binary image in which white pixels represent chopped pieces and black pixels represent the background, an image segmentation process is necessary. The proposed method used to distinguish between these two types of pixels is based on the absolute luminance level of the foreground image. Pixels corresponding to chopped pieces (moving objects) have a higher luminance level in the foreground image obtained after subtracting the background (scene with no moving objects) from the acquired image (scene with moving and nonmoving objects). Figure 10(b) shows the resulting segmentation of the image in Fig. 10(a).

The next step is the feature extraction and region filtering. For each segmented region in the previous step, two features are extracted: the area and the perimeter. Only those regions in which these two features are within certain limits are considered valid. This process filters regions that are not related to chopped pieces of material and that could be detected under an anomalous operation of the system. The results can be seen in Fig. 10(c) with the boundary lines of the regions of interest.

Finally, jam detection is based on the number of regions for each nozzle. However, analyzing a single image and counting the number of regions ejected from each nozzle is not enough to detect jams in the pickling line robustly. The solution proposed is to create a historical log of the number of chopped pieces ejected from each nozzle. Thus, when a new image is acquired and the numbers of chopped pieces for each nozzle are counted in this image, a weighted moving average will be calculated with the historical data. The decision about the existence of a jam is based on the current value of the weighted moving average. If this value is lower than a determined decision threshold, a jam is detected for that nozzle. Due to the different number of chopped pieces detected for each nozzle, two different decision thresholds are used—one for each nozzle.

The application also provides a remote interface using the modules of the architecture. Images are resized and streamed using an RTP image flow in h264, which is sent to the Wowza media server. Also, the configuration of the

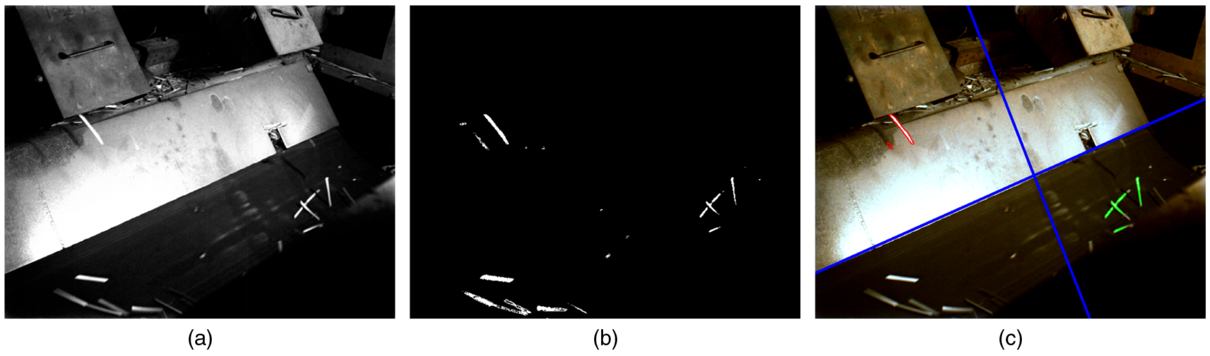


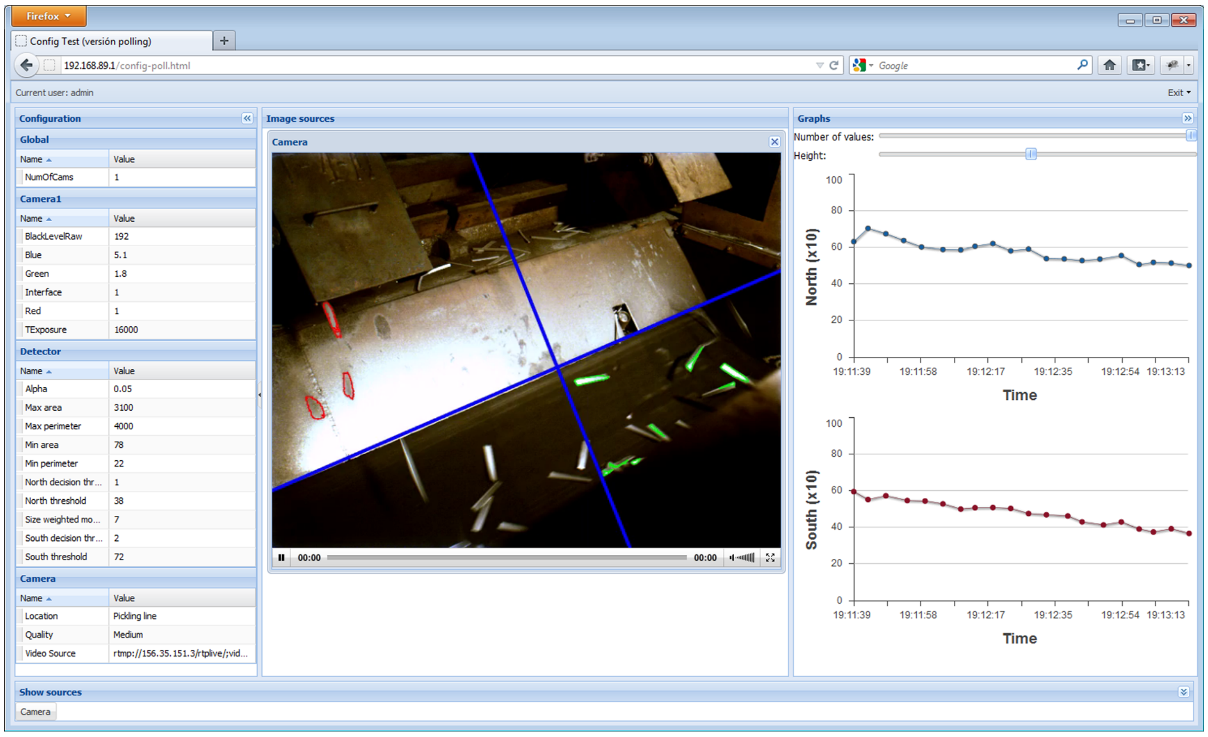
Fig. 10 Detection of chopped pieces. (a) Image acquired during pickling. (b) Edge map. (c) Detected pieces for each nozzle.

application is serialized and sent to the LightHttpd Web server using the *CgiPubSub* module. The resulting application can be seen in Fig. 11. In this case, two graphs are also provided with the evolution of the number of chopped pieces in the north and south nozzle.

The proposed procedure to detect jams in a pickling line was first applied to an extensive dataset of videos acquired during pickling, containing thousands of frames. These experiments were used to adjust the parameters of the system and to validate the procedure. The proposed machine vision application provided fast and robust detection, detecting all the jams in the dataset with negligible delay caused by the size of the weighted moving average. After testing, the jam detector was installed in the Number 1 pickling line of ArcelorMittal in Avilés, Spain. The system has been running there successfully for 14 months.

The number of frames per second processed by the proposed system is limited by the illumination of the nozzles. Using two halogen lamps of 500 watts each, the configured exposure time is 16 ms. The system runs at full camera resolution: 1280×1024. Without illumination constraints, the system is able to work in real time at full frame rate on a hexa-core Intel Xeon 5620 running at 2.4 GHz. The system is able to process each frame consistently before the new one arrives. Thus, the system can be considered to have a predictable behavior.

Using the proposed architecture for the jam detector greatly simplified the development of the application. Many tasks, such as acquisition, visualization, or remote management, were completely trivial. This way, the developer could be focused on the main image processing modules, rather than wasting time creating interfaces and low-level acquisition modules for the particular camera. Moreover, the parallelization of the application was automatic. The developer designed the modules required to extract the necessary information from the images. The architecture executed these modules efficiently using the available resources of the machine. The architecture also required the developer to design the application as a set of independent modules, which produced a better orthogonal code, where a piece could be substituted by another without affecting the rest. For example, there were several versions of the background



Configuration

Global

Name	Value
NumOfCams	1

Camera1

Name	Value
BlackLevelRaw	192
Blue	5.1
Green	1.8
Interface	1
Red	1
TExposure	16000

Detector

Name	Value
Alpha	0.05
Max area	3100
Max perimeter	4000
Min area	78
Min perimeter	22
North decision thr...	1
North threshold	38
Size weighted mo...	7
South decision thr...	2
South threshold	72

Camera

Name	Value
Location	Pickling line
Quality	Medium
Video Source	rtmp://156.35.151.3/rtplive/;vid...

Fig. 11 User interface of the jam detector application.

removal modules using other algorithms, such as Gaussians mixtures.

5 Conclusions

Recent trends in hardware platforms, such as the steady increase in computing power based on parallelism, make in-depth understanding of many technologies necessary in order to build efficient real-time image processing applications. One solution to reduce this complexity is to use frameworks that can be used to simplify the design and tuning of the application.

This paper presents a novel software architecture for time-constrained machine vision applications. The design is based on some of the most important nonfunctional requirements of a real-time image processing architecture, including scalability, reusability, flexibility, extensibility, and reliability. It is designed to deal with different types of parallelism, allowing efficient use of the parallel execution resources required to process heavy images and videos in real time.

The core of the architecture is based on a notable extension of the efficient publish/subscribe pattern. A topic-based filtering in which messages are published to topics is used to route the messages from the publishers to the subscribers interested in that particular type of message. The architecture is loosely coupled; publishers are completely decoupled from subscribers. The proposed approach greatly improves the robustness, modularity, extensibility, and flexibility of the architecture. In addition, this architecture provides the opportunity for scalability, as subscribers can be executed in parallel automatically. The overhead introduced by the proposed design of the architecture is very low (less than 100 μs on modern hardware), enabling it for real-time work.

The architecture has been applied to a real machine vision application, where the possibilities of the proposed architecture are demonstrated. The use is very easy and intuitive yet rich enough to create a wide variety of real-time applications. Also, the repository of reusable modules that implement the most common tasks in an image processing application greatly simplifies the development of applications. Many modules have been included in the architecture, from image acquisition modules to user interface modules used to create rich Web interfaces.

The proposed architecture provides the foundations of a modern, efficient, and well-designed machine vision application. The design and implementation has carefully followed the most important quality metrics in software applied to the special case of machine vision applications where reliability and maintainability are major requirements. The work presented in this paper is very likely to find potential applications not only in the design of image processing frameworks, but also in the organization and design of any machine vision application.

Acknowledgments

This work was partially supported by the Asturian Regional Ministry of Education and Science under Project No. PC10-03.

References

1. N. Kehtarnavaz and M. Gamadia, *Real-time Image and Video Processing: From Research to Reality*, Morgan & Claypool Publishers, San Rafael, California (2006).
2. C. C. Weems, "Architectural requirements of image understanding with respect to parallel processing," *Proc. IEEE* **79**(4), 537–547 (1991).
3. A. R. Francois, "Software architecture for computer vision: beyond pipes and filters," in *Inst. for Robotics and Intelligent Systems*, University of Southern California, Los Angeles (2003).
4. A. François and G. Medioni, "A modular software architecture for real-time video processing," in *Proc. Computer Vision Systems: Second International Conference*, Vol. 2, pp. 35–49 (2001).
5. M. Camplani and L. Salgado, "Scalable software architecture for on-line multi-camera video processing," in *Electronic Imaging 2011: Real-time Image and Video Processing*, Vol. 7871, pp. 1–15, SPIE, San Francisco, California (2011).
6. T. Morwald et al., "Blort-the blocks world robotic vision toolbox," in *Proc. ICRA Workshop Best Practice in 3D Perception and Modeling for Mobile Manipulation*, IEEE, Anchorage, Alaska (2010).
7. T. Muller, B. A. Tran, and A. Knoll, "Automatic distribution of vision-tasks on computing clusters," in *Electronic Imaging 2011: Parallel Processing for Imaging Applications*, Vol. 7872, pp. 1–102, SPIE, San Francisco, California (2011).
8. G. R. Andrews, *Foundations of Multithreaded, Parallel, and Distributed Programming*, Vol. 11, Addison-Wesley, Reading, Massachusetts (2000).
9. C. Shen, S. S. Fels, and J. J. Little, "Openvl: towards a novel software architecture for computer vision," in *2007 IEEE Conference on Computer Vision and Pattern Recognition*, pp. 1–8, IEEE, Minnesota (2007).
10. C. Shen, J. J. Little, and S. Fels, "Towards openvl: improving real-time performance of computer vision applications," in *Embedded Computer Vision*, B. Kisacanin, S. S. Bhattacharyya, and S. Chai, Eds., pp 195–216, Springer, London, United Kingdom (2009).
11. G. Bradski and A. Kaehler, *Learning OpenCV: Computer Vision With The OpenCV Library*, O'Reilly Media, Sebastopol, California (2008).
12. P. Azad, T. Gockel, and R. Dillmann, *Computer Vision: Principles and Practice*, Elektor Electronics Publishing, Netherlands (2008).
13. T. S. Yoo, *Insight Into Images: Principles and Practice for Segmentation, Registration, and Image Analysis*, CRC Press, Wellesley, Massachusetts (2004).
14. D. F. Garcia et al., "Shape inspection system for variable-luminance steel plates with real-time adaptation capabilities to luminance variations," *Real-Time Imag.* **8**(4), 303–315 (2002).
15. J. Molleda et al., "Real-time flatness inspection of rolled products based on optical laser triangulation and three-dimensional surface reconstruction," *J. Electron. Imag.* **19**(3), 031206 (2010).
16. R. Usamentiaga et al., "Algorithms for real-time acquisition and segmentation of a stream of thermographic line scans in industrial environments," *J. Imag. Sci. Technol.* **49**(2), 138–153 (2005).
17. R. Usamentiaga et al., "Real-time line scan extraction from infrared images using the wedge method in industrial environments," *J. Electron. Imag.* **19**(4), 043017 (2010).
18. N. A. Barendt et al., "A distributed, object-oriented architecture for platform-independent machine vision," in *Proc. Int. Conf. on Robotics and Manufacturing*, pp. 50–55, IEEE, Lueven, Belgium (1998).
19. J. Martinez et al., "A modular and scalable architecture for PC-based real-time vision systems," *Real-Time Imag.* **9**(2), 99–112 (2003).
20. P. Schalk et al., "Framework for automatic quality control in industrial environments using distributed image processing," *J. Electron. Imag.* **13**(3), 504–514 (2004).
21. C. J. Neill, "Leveraging object-orientation for real-time imaging systems," *Real-Time Imag.* **9**(6), 423–432 (2003).
22. C. J. Neill and P. A. Laplante, "Imaging frameworks: design for reuse in real-time imaging," *Proc. SPIE* **5297**, 1 (2004).
23. R. S. Sangwan et al., "Building reusable components for real-time imaging systems," *J. Imag. Sci. Technol.* **49**(2), 154–162 (2005).
24. B. Meyer, *Object-Oriented Software Construction*, Vol. 2, Prentice Hall, New York (1988).
25. E. R. Dougherty and P. A. Laplante, *Introduction to Real-Time Image Processing*, SPIE Press/IEEE Press, Bellingham, Washington (1995).
26. P. T. Eugster et al., "The many faces of publish/subscribe," *ACM Comput. Surv. (CSUR)* **35**(2), 114–131 (2003).
27. E. Gamma et al., *Design Patterns: Elements of Reusable Object-Oriented Software*, Vol. 206, Addison-Wesley, Reading, Massachusetts (1995).
28. A. Alexandrescu, *Modern C++ Design: Generic Programming and Design Patterns Applied*, Addison-Wesley Professional, Upper Saddle River, New Jersey (2001).
29. T. Mattson, B. Sanders, and B. Massingill, *Patterns for Parallel Programming*, Addison-Wesley Professional, Boston (2004).
30. D. Leijen, W. Schulte, and S. Burckhardt, "The design of a task parallel library," *ACM SIGPLAN Notices* **44**(10), 227–242 (2009).
31. R. Usamentiaga et al., "Jam Detector for Steel Pickling Lines Using Machine Vision," in *2012 IEEE Industry Applications Society Conference*, Vol. 1, pp. 1–8, IEEE, Las Vegas, Nevada (2012).
32. M. Piccardi, "Background subtraction techniques: a review," in *2004 IEEE International Conference on Systems, Man and Cybernetics*, Vol. 4, pp. 3099–3104, IEEE, The Hague, Netherlands (2004).

Rubén Usamentiaga is a tenured associate professor in the Department of Computer Science and Engineering at the University of Oviedo, where he received his MS and PhD in computer science in 1999 and 2005, respectively. In recent years, he has been working on several projects related to computer vision and industrial systems. His research interests include real-time imaging systems and thermographic applications for industrial processes.

Julio Molleda is an associate professor in the Department of Computer Science and Engineering at the University of Oviedo, where he received his MS and PhD in computer science in 2005 and 2008, respectively. In recent years, he has been working on real-time imaging research projects in computer engineering. His research interests include real-time imaging systems and range measurement techniques.

Daniel F. García is a full professor in the Department of Computer Science and Engineering at the University of Oviedo, where he received his PhD in electrical engineering in 1988. Since 1994, he has been responsible for the computer engineering area at the University of Oviedo. His current research interest is in the development of high-performance real-time and embedded systems applied to quality assurance and production inspection in industry, where he has more than 100 published papers. For the last 10 years, he has been conducting research projects in the area of information technologies applied to industry at the national and European levels. He is a member of ACM and the IEEE Computer Society.

Francisco G. Bulnes is an associate professor in the Department of Computer Science and Engineering at the University of Oviedo, where he received his MS in computer science in 2007 and is currently working toward his PhD. His current research interest is in the area of real-time imaging systems.