Scalable software architecture for on-line multi-camera video processing

Massimo Camplani and Luis Salgado

Grupo de Tratamiento de Imágenes, Universidad Politécnica de Madrid, 28040, Madrid, Spain

ABSTRACT

In this paper we present a scalable software architecture for on-line multi-camera video processing, that guarantees a good trade off between computational power, scalability and flexibility. The software system is modular and its main blocks are the Processing Units (PUs), and the Central Unit. The Central Unit works as a supervisor of the running PUs and each PU manages the acquisition phase and the processing phase. Furthermore, an approach to easily parallelize the desired processing application has been presented. In this paper, as case study, we apply the proposed software architecture to a multi-camera system in order to efficiently manage multiple 2D object detection modules in a real-time scenario. System performance has been evaluated under different load conditions such as number of cameras and image sizes. The results show that the software architecture scales well with the number of camera and can easily works with different image formats respecting the real time constraints. Moreover, the parallelization approach can be used in order to speed up the processing tasks with a low level of overhead.

Keywords: Multi-Camera system, on-line video processing, background subtraction, mixture of Gaussians

1. INTRODUCTION

Multi-camera systems are becoming more and more popular thanks to the growing computer processing capabilities and the decreasing prices of high quality cameras. Moreover, the demand of new and more efficient systems for several applications such as video surveillance[1,2] , 3D acquisition systems[3] , virtual rooms[4] and immersive teleconferencing systems[5] , is encouraging the researchers to find new and optimized solutions. The advantages of multi-camera systems in these contexts is that they overcome classical problems of mono camera systems such as occlusions, limited field of view and the possibility to extract from the scene 3D information. On the other hand, multi-camera systems pose challenging problems to the research community mainly related to the management and the analysis of the huge amount of data provided by the system.

A common way to tackle this problem is to develop multi-camera systems using smart cameras[6] . The main idea of the smart camera is to build a single node that encloses both acquisition and processing algorithms. These approaches are well suited for real time image processing since they rely on the presence of dedicated hardware such as FPGA and DSP[7] . Moreover, they have been also widely used in large camera systems, like sensor networks[8] , for their low power consumption characteristics. However, this solution has some drawbacks. In fact, dedicated hardware can present lack of flexibility, especially for high level image processing tasks. Furthermore, the design of cooperative tasks and distributed software is not straightforward[9] . Also the communication between cameras is a crucial point for the efficiency of the system.

An alternative solution to tackle these issues ,often used in indoor environments, is based on a centralized server architecture, where the central server collects and processes all the data. The main limitations of this model are that it is very power demanding, it requires a large bandwidth communication channel and, obviously, it does not allow a high degree of scalability[2] . An improvement of the centralized server architecture is constituted by peer to peer network of computers where different nodes manage different cameras. This model in our view is one of the most attractive one since it is an hybrid solution that guarantees a good trade off between flexibility and scalability. Many software packages have been developed for on line video processing in network of computers environments[10–13] .

E-mail: {*mac, L.Salgado*}@gti.ssr.upm.es. Web: http://www.gti.ssr.upm.es/

In this paper we propose a software architecture for efficiently managing multi-camera video processing in real time scenarios with off the shelf work stations. The aim of this architecture is to guarantee a good trade off between computational power, flexibility and scalability for single node applications. The design and the use of multi-camera systems are not straightforward and need a deep knowledge of system details ranging from process communication, resource allocation to camera data transfer and multi-threading programming. However, a final user, like a computer vision algorithm developer, can not know (and probably does not need to know) all the system details. For these reasons, our choice has been to develop an architecture that is able to hide many of the multi-camera system details from the user and that can be easily adapted to different video processing applications without a great effort. The proposed architecture is scalable since can manage many camera data streams; it is flexible because it can work on different hardware platforms and with different kinds of acquisition devices. Furthermore, the proposed architecture has a modular structure that allows to easily integrate it with custom video processing applications. Moreover, the architecture has been developed in order to be lightweight and does not require many computational resources. Finally we propose an approach that allows to easily parallelize the desired video processing application in order to exploit the processing capabilities of the host machine.

The proposed software architecture has been tested in a multi-camera system in order to efficiently manage multiple 2D object detection tasks in a real-time scenario. As background estimation algorithm the Gaussian Mixture Model, presented by Stauffer and Grimson[14] , has been used. System performance has been evaluated under different load conditions such as variable number of cameras and image sizes.

The paper is organized as follows: in sec. 2 an overview of the proposed software architecture is given. In sec 3 the implemented system and the case study are introduced. Results are shown in sec. 4, and finally conclusions are drown in sec. 5.

2. SOFTWARE ARCHITECTURE OVERVIEW

The development of a multi-camera system presents many critical points to manage such as: process communication, resource allocation, camera data transfer or efficient multi-threading management strategies. These issues can represent a bottleneck in application development for non expert users. Moreover, users like computer vision algorithm developers prefer not to deal with all these details. For these reasons we propose a software architecture that is able to guarantee a trade off between computational power and flexibility thus shielding the final user from many low level details of the multicamera system.

The proposed software architecture is a scalable, modular and flexible architecture for multi-camera system management in real time scenarios. The architecture is scalable since it can manage and process efficiently multiple data stream adding small overhead and saving useful computational resources. It is also scalable because it can be used with a variable number of cores in multi-core host machines.

It is modular since it is composed (see the following paragraphs for more details) by functional blocks that operate independently in different phases of the overall processing chain. This modularity allows the developer to modify and integrate with new software each block without affecting the structure of the others.

The architecture is flexible since it can be easily adapted to different scenarios. First of all it is flexible in the sense that it is hardware independent: it can be easily ported on different host machines and it is not related to any particular model or type of camera. It is also flexible from the point of view of the data, in fact all the modules are automatically adapted to different image formats or sizes. Moreover, the architecture is designed to operate in off the shelf machines with commercial operating systems (no Real Time Operating Systems are required) and without the use of any dedicated hardware.

The software architecture is fully developed in C++ and the management of image data and the processing tasks is based on OpenCV libraries[15] (version 2.1). This choice is motivated by the fact that this is a widely used library for computer vision applications. Moreover, OpenCV base developments can be easily implemented in the software architecture.

As shown in fig. 1 there are two main modules of the software architecture: the Central Unit and the Processing Unit (PU). These are logic modules and do not correspond necessarily with a physical node (i.e. a

work station): for example, the same work station could be the host of several PUs and the Central Unit. The Central Unit works as a supervisor of the running PUs and each PU manages the acquisition phase and the processing phase.

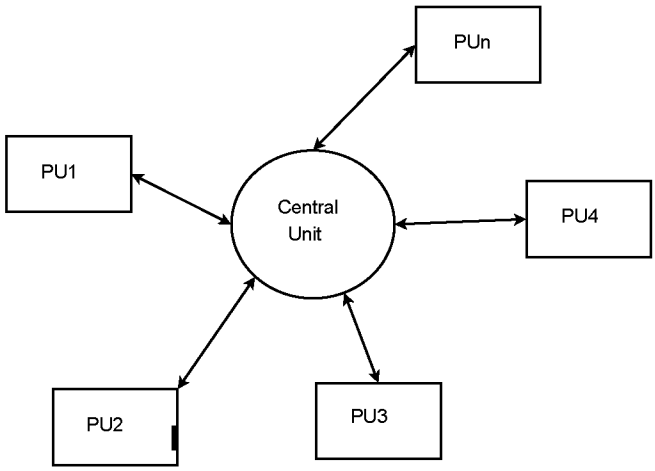

Figure 1. Software Architecture main modules.

2.1 Central Unit

The Central Unit is a lightweight component that initializes, monitors and closes all the processes. First of all it checks the state of all the cameras in the system and it assigns an identifier $camID$ to each running camera. The $camID$ is used also to label the acquired images that are univocally identified by it and the time stamp (see section 2.2.1). The camera settings are acquired (i.e. frame size, gain and other settings) and, thanks to this information, the PUs are initialized. The Central Unit starts and stops the acquisition. During the acquisition the Central Unit gathers all the information from the modules of the system, such as frame rate, processing time and the state of the image buffer (see section 2.2.3). It is also in charge to correctly close all modules and shut down the system.

Although system initialization, monitoring and shut down are the main tasks of the Central Unit, its functionality can be extended or modified. In particular more complex Central Units can be designed to collect the results of each processing module and combine them in the case that high level tasks are required by the application. An example of more sophisticated Control Units can be found in an our previous work[16] where the Control Unit tunes the parameters of a background subtraction algorithm in order to reduce the computational load.

2.2 Processing Unit

As previously mentioned, the PUs are logical modules that can be executed either in multiple or in a single physical node. The system is scalable since each PU can be connected with one ore more cameras. The PU is in charge to manage the incoming data streams and process them and it is constituted by three functional blocks as shown in fig. 2: the acquisition module (AM), the image buffer manager (IBM) and the processing module (PM). Basically AM is in charge of continuously acquiring the images and deliver them to the IBM. The IBM is the module that manages the buffer (of fixed size) where the acquired images are temporally stored. It takes care of the buffer accesses and manages undesired situations like buffer overflows. The PM reads the images stored in the buffer and processes them. These modules operate independently and this aspect confers a great modularity to the global software architecture. The decoupling between AM and PM is very attractive for the final user, since he can design his processing tasks without taking care of acquisition and camera details. Also the independence of the IBM is very interesting, since it is possible to use different management strategies as a function of the desired application. For example, the frame skipping strategy could be changed by replacing the default implementation that discards the newest frame, with a new strategy that discards the oldest one.

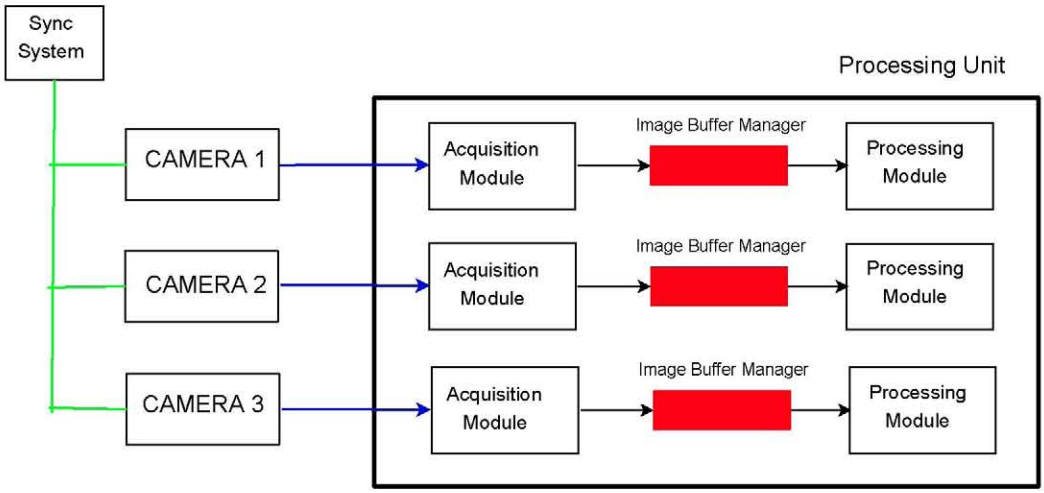

Figure 2. PU main modules.

2.2.1 Camera and Image Data

The data coming from the camera devices are converted to OpenCV data structure (Mat Class). In fact, as previously mentioned, the software architecture has been integrated with OpenCV libraries. Moreover, once the images are acquired, the camera identifier $camID$ and a frame number identifier $frameID$ are assigned to them. All the other modules of the software architecture work with this meta-data (image and identifiers).

The software architecture hides the low level details of the camera management. In particular, the access to the device features is possible throughout a class called *cameraManager*. The use of this class allows to wrap each device into its corresponding object in the software architecture. The methods of the class allow to modify device parameters (i.e. frame rate, image size, exposure time, etc.) and open and stop a data stream. This level of abstraction guarantees a high flexibility to the software architecture since it is possible to use it with different camera models, without affecting the other software modules that work always with the same object interface. In our case, we developed *cameraManager* modules for two different types of cameras: JAI[17] and AXIS[18] .

Particular attention has to be paid to the camera synchronization system. As it can be observed in fig. 2, this system does not belong to the PU modules and it can not be completely enclosed in our software architecture. Camera synchronization is a critical task in multi-camera systems and the required accuracy depends on the particular application and on the acquisition devices in use. When a very precise synchronization is needed, the most used solution is hardware synchronization. In other situations, a software synchronization system could fulfill application requirements, using for example tools and libraries like Network Time Protocol (NTP)[19] . The *cameraManager* class gives the possibility to set the synchronous (or asynchronous) acquisition, but the triggering system (software or hardware) is an independent module.

2.2.2 Acquisition Modules

The AM allows to acquire images from the devices and deliver them to the IBM. AM initializes the acquisition tasks and then it starts a continuous process that is activated when a new image is available. When a new image is acquired, it is labeled with the corresponding $cameraID$ and $imageID$ and the data is converted in a OpenCV data structure. Then the images are delivered to the IBM. It is worth noting that overflow buffer conditions do not affect the AM since it is the IBM module that is in charge to write the image in the buffer and manage the overflow condition.

2.2.3 Image Buffer Manager

The image buffer is used to compensate the different rates with which the flow of data is acquired by the AM and processed by the PM. In fact, whereas the acquisition rate remains constant, the processing rate can present deviations from a fixed value. This aspect is particularly relevant in our application since it is based on general

purpose processors where the available computational resources have to be shared among different applications. For this reason, the presence of the image buffer avoids (up to a certain limit) frame skipping. Moreover, the image buffer confers to the system another degree of modularity: in fact, PM and AM are completely decoupled and they can be modified (or substituted) without affecting each other because they interact with the IBM.

The IBM is a module that manages the accesses to the buffer. The buffer access is thread safe since a semaphore system guarantees that only one process can write on the buffer (the AM) and one process can read an image from it (the PM). A specific policy to handle buffer overflow is defined in this module: it is possible to eliminate the oldest frame or choose to skip the newest one. The IBM communicates to the Central Unit the presence of new frames and of the occurrence of overflow condition.

The IBM is a quite lightweight module of the PU since the buffer contains pointers to the images: no images are copied during reading and writing operations: for each image the space in memory is allocated by the AM and released when PM ends the task.

2.2.4 Processing Module and Task Parallelization

The PM has one main task: continuously reading images from the buffer and processing them. It is the most customizable module of the entire software architecture. This module is the one that has to be developed by the final user (i.e. developer of video analysis algorithms) and it can be constituted by any processing task, including visualization or image storage. There are a few design specifications for this module. The PM has to be implemented as a cascade two blocks: the first block is already provided by our software architecture and handles the interface with the IBM, allowing to easily read images from the buffer (in case of empty buffer the interface suspends the PM operations until a new frame is available); the second block is the implementation of the desired application.

As aforementioned, the modular architecture guarantees great advantages for the developer since he is able to integrate his software without paying attention to other issues like acquisition. Moreover, it could give more flexibility to the user application, in the sense that it allows to implement different processing tasks on the different data streams.

The PM is the time-critical point of the entire software architecture. In general, the other modules do not perform tasks that require a lot of processing resources, hence, meeting real time constraints will depend on the efficiency of the PM implementations. Although there are many programming tricks that can help to reduce the processing time of the desired application software, in some cases they turn out to be insufficient.

An obvious solution could be found performing a parallelization of the application: this solution is indeed particularly attractive in the case that a multi-core host machine is used. However, following this approach could be time consuming during the application software development and makes more difficult the integration with the overall software architecture.

In order to ease the application software implementation and integration in the PM module, we developed a strategy that allows to easily parallelize the application without affecting the modularity and the flexibility of the overall architecture.

The proposed strategy is based on a pipeline scheme, that is one of the most suitable approaches for image processing tasks parallelization. Furthermore, this approach allows to parallelize the desired applications at a high level, thus shielding the users from thread-level programming details.

The pipeline is a cascade of several stages that are executed in parallel (i.e. using different threads). Each stage performs a specific processing task with the input data coming from the previous stage and passes the results to the next stage. This approach allows to implement the desired application using data level parallelism or task level parallelism. Data parallelism can be used in case of low level image operations such as preprocessing and filtering and all the *pixel wise* algorithms. In fact, identical image processing tasks are applied to different regions of the image. In task level parallelism, each pipeline stage corresponds to a different operation on the image. This kind of parallelization method is very convenient in high level tasks, like object tracking or objects recognition. The throughput of the pipeline is theoretically limited to the slowest pipeline stage.

Independently of the type of parallelization that will be applied, the aspect on which the developer has to concentrate is the identification of the sub-blocks into which the desired application can be divided. Sub-blocks

are simple operations that can not be further decomposed, such as filtering operations, lens distortion correction, color plane conversion etc. Once the sub-bocks of the desired application have been identified, it is possible to design the pipeline stages as single sub-block or a set of them. Then the pipeline is built as a cascade of stages. In the proposed software architecture, it has been developed a library containing pipeline stages that perform the most common image processing tasks, like filtering, image transformation etc. Also a library containing different pipeline models has been implemented. These software components can be reused, combined and extended speeding up the software development process of the desired application. The final user can implement in a seamless way data parallelism, task parallelism or a mixture of those, but it is recommended to keep the pipeline stages *balanced* as much as possible to avoid continuous pause of threads waiting the end of the previous stage. The design flow of processing task is shown in fig. 3.

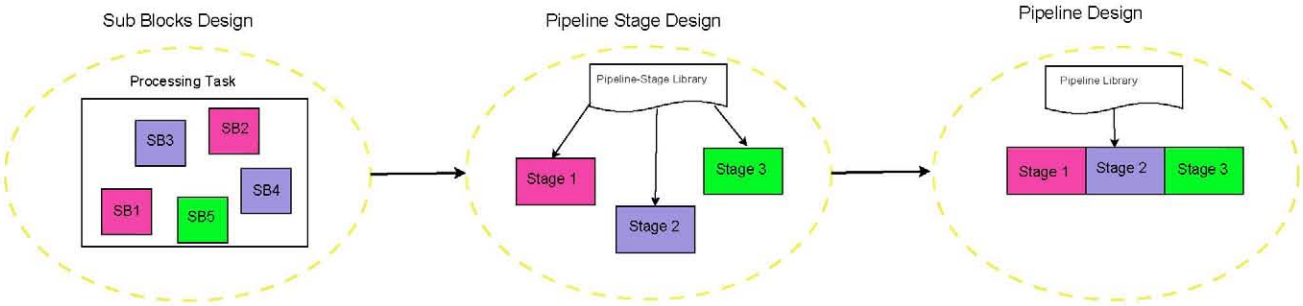

Figure 3. Processing Task design flow.

This part of the software architecture uses, as a core for the pipeline management, the Intel Thread Building Block libraries[20](TBB). These libraries allow to easily develop multi-threading and parallel applications. We have developed a software layer to adapt these libraries to our software architecture and to the image processing field.

The software development efforts to build a pipeline stage are minimal. In fact, each stage has to be implemented as a class that acts as a wrapper of the assigned sub-blocks. The parameters relative to each sub-block (i.e. coefficients, thresholds, image size etc.) are assigned to the wrapper class attributes. The wrapper class has a function that is used to execute the pipeline stage. This function has one input, corresponding to the data coming from the previous pipeline stage, and one output that is passed to the next pipeline stage. The input/output parameter *exchanged* by pipeline are basically images or more complex data structures containing more information. It is necessary to *exchange* pointers to these data structures to avoid object copy and slow down the execution of the stage. It is worth noting that the first pipeline stage has to enclose also the module to access the image buffer. The image pointer is propagated to the pipeline and processed by each stage. Usually, the acquired image memory is released in the last stage of the pipeline.

A pipeline has to be designed as an object that contains the pipeline stages and the data structure needed to monitor their execution. Information, like processing time, or results are evaluated and passed to the Central Unit. It is important to underline that only one thread at a time can execute a pipeline stage: threads synchronization is managed by the TBB.

This approach guarantees that the same pipeline will work correctly in different host machines. For example, the same pipeline can be executed in different host machines with different number of cores, as there is no hard link between number of pipeline stages and the number of available cores. It is clear that a long pipeline will take advantage of the presence of many cores; on the other hand an overestimated pipeline (very long pipeline or few data to process) will decrease the overall performance of the software architecture since the overhead introduced by the pipeline will be greater than the benefits of parallelization.

In conclusion, the advantages for the developer in using this parallelization approach are several: sub-block level modularity, code reuse, no specific multi threading programming skills required and platform flexibility. The point to consider is that the sub-blocks design and the pipeline system have to be carefully designed.

3. CASE STUDY: MULTI-CAMERA OBJECT DETECTION

Multi-Camera systems are well suited for surveillance applications since they can avoid problems encountered with single camera one such as occlusions and limited field of view. In this research area one of the most challenging problems is objects tracking and positioning in 3D environment. Usually, independent 2D object detection algorithms are applied to each camera node and then this information is fused by a 3D tracking module (an example can be found in the works by Mohedano et al.[21,22]).

In this paper as case study we apply the proposed software architecture to a multi-camera system to efficiently manage multiple 2D object detection modules in a real-time scenario. Each object detection module is based on background subtraction algorithm. Background subtraction is a technique that estimates a background model of the scene and, then, any deviation from the model is considered as a potential moving object. Different background modeling techniques have been presented in the literature[23] .

As background subtraction algorithm we have implemented the one proposed by Stauffer and Grimson[14] . In this algorithm each pixel is modeled independently as a mixture of Gaussian distributions. The algorithm is composed by two fundamental steps: the first step is the estimation of whether or not a pixel belongs to the background model; during the second step the Gaussian parameters are updated. This *pixel-wise* approach is computationally demanding since for each pixel all the parameters of the distributions have to be updated. More details about mixture of Gaussians algorithms are presented in appendix A. From now on we will refer to this background subtraction algorithm as the GM algorithm.

3.1 OBJECT DETECTION: PROCESSING TASKS

The block diagram of the object detection module is shown in fig. 4. The processing chain can be decomposed in three main processing tasks: pre-processing, GM algorithm and post-processing.

The pre-processing tasks include Bayer to RGB conversion, white balance operations, lens distortion correction and color space conversion. We decide to include the tasks of Bayer to RGB conversion and white balance in the acquisition module; as it can be noted these tasks in fig. 4 are enclosed in the yellow box. Cameras have been calibrated with the calibration toolbox developed in our research group[24] and the estimated intrinsic camera parameters have been used to correct the lens distortion as suggested by Hartley et al.[25] . The color space conversion is necessary to switch from RGB color space to YCbCr color space. The original GM algorithm[14] is applied to the RGB color space: in this work the authors assume the channels of that space are independent; however, this hypothesis does not hold and other works have shown that the YCbCr is more noise resilient than RGB[26] . For these reasons, we decide to apply the GM algorithm to the YCbCr color space without subsampling on the color components (YCbCr 4:4:4).

The second task is constituted by the GM algorithm. Although the GM algorithm can be further decomposed in two blocks, foreground/background pixel estimation and parameters update, we have decided to implement it in a single one. This implementation leads to a more efficient algorithm in terms of computational cost. In fact, the two blocks choice would slow down the algorithm since it would be necessary to *iterate* on the parameters twice: one time for pixels classification and the other one for parameters update. Moreover, the parallelization of these two blocks would not be straightforward, since it would require thread synchronization at pixel level: when the classification of one pixel has finished the update procedures can start. Therefore, this solution would introduce an unacceptable overhead in the pipeline: for this reason, this option has been discarded.

In our implementation of GM algorithm in a single block, there is a single structure that contains all the distributions parameters, that allows to classify the pixel (as part of background or foreground) and update the parameters in the same iteration. The only disadvantage of this approach is that it is demanding from a point of view of the memory occupancy since it requires several contiguous memory blocks.

Post-processing tasks are needed to refine the results of the GM algorithm. A typical output of a background subtraction algorithm is a binary mask that represents the foreground object in the scene (pixels set to 1). This mask is usually corrupted by noise and errors introduced by the background subtraction algorithm. For these reasons, useful techniques such as morphological processing are used in order to reduce the errors. A comparative study[27] on different background subtraction algorithms showed that these procedures can significantly improve the results. In our application we have decided to apply an erosion and a dilation operation with different (size and shape dependent) structuring elements.

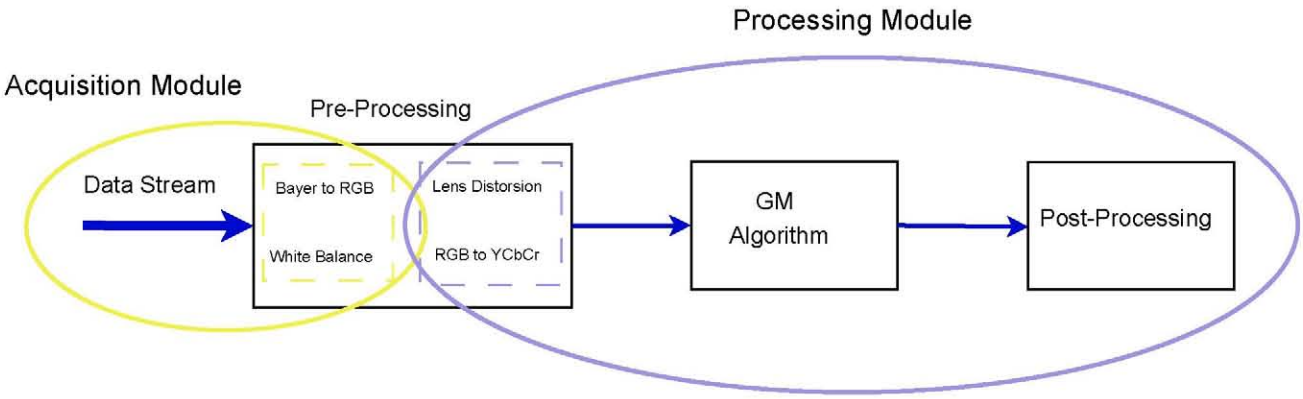

Figure 4. Block diagram of the image processing tasks.

3.2 MULTI-CAMERA SYSTEM OVERVIEW

The software architecture has been tested on a multi-camera system composed by 3 cameras and one commercial work station. Camera models are JAI CB-080 GE[28] that is a Giga ethernet camera that delivers 8bit Bayer images at a maximum resolution of 1032×778 with maximum frame rate of $30fps$. The workstation is equipped with two quad core processors of the Intel Xeon family[29](Intel Xeon X5550), 16GB of RAM and fast HD (15krpm) in RAID-0 configuration. The work station and the cameras are connected through Giga-Ethernet interfaces with a point to point connection.

It is important to highlight that in our application high accuracy of image synchronization is required to correctly fuse the information extracted by the individual 2D object detection modules. For these reasons we develop an external triggering system that allows to minimize the synchronization error. In particular, the cameras have been connected in a master-slave configuration where the timing circuit of the master camera is used to generate the trigger of the system. The acquisition of the cameras, master included, is governed by the generated trigger signal. The symmetry of the connection circuit guarantees that all the cameras are tightly synchronized with a synchronization error of the order of μs.

We decide to build an unique PU that can manage up to three camera streams.

4. RESULTS

In this section the performance of the proposed software architecture will be analyzed taking as testbed the case study that has been previously described. Two version of the architecture will be compared: one with a PM module integrated with the pipeline system (PV_{ap}) and the other one with the original version of the application(O_{ap}). In particular, the proposed software system has been tested in different scenarios, varying the number of cameras, image size and color space. Results will show that the architecture is able to efficiently manage multiple data streams and that the parallelization approach allows obtaining a sensible reduction of the processing time.

As far as the image size is concerned, we tested the software architecture with four different image sizes: the maximum size of the images provided by the camera 1032×778; 900×670; a SVGA resolution 800×600 and a VGA resolution 640×480. From the point of view of the color space we have implemented two options: applying background subtraction algorithm using all the three components of the YCbCr(4:4:4) space or, conversely, using only the luminance component Y.

From now own we will use the acronym pT to indicate a processing time and fr_{max} to indicate the maximum frame rate achievable by a processing task. Moreover, each pT reported will be expressed in ms and each fr_{max} will be expressed in frame per second (fps).

In table 1 the pT and the fr_{max} of the overall application (O_{ap}) and of the pT of the three sub-blocks are reported for different image sizes and different color spaces. In table 1 the first two columns indicate the color space and the image size. It is worth noting that the implemented GM algorithm has been carefully optimized.

In fact, thanks to our optimized implementation, when applied to the YCbCr space, it does not require a pT triple respect to the pT of the algorithm when only the Y component is considered. However, as it can be noticed, it is still the bottleneck of the application. For instance O_{ap} at full image size could work with a fr_{max} of about $\approx 13fps$ if only the luminance plane is considered during the background subtraction algorithm, and $\approx 9fps$ in the case of the YCbCr space.

These results show that, when considering the full image size, it is not possible to operate on a real time scenario, for this reason a speed up of the processing task can be obtained using the pipeline approach. In the next sections two examples of the task parallelization and data parallelization using the proposed architecture will be presented. It is shown how the original application can be easily redesigned to be adapted to the pipeline. Moreover, results show that the overhead introduced by the pipeline system is small with respect to the increasing performance of the system.

Table 1. Processing Time of the image processing task and its sub-blocks (expressed in ms).

Color Plane	**Img. Size**	$\mathbf{pTO_{ap}}$	$\mathbf{fr_{max}O_{ap}}$		pT Pre-proc.	pT GM	pT Post-proc.
YCbCr	640x480	40.30	24.81		12.56	25.87	1.88
	800x600	62.98	15.88		19.70	40.41	2.86
	900x680	79.57	12.57		24.87	51.08	3.62
	1032x778	106.96	9.35		33.59	68.59	4.78
Y	640x480	29.60	33.79		12.76	14.94	1.89
	800x600	45.42	22.02		19.59	22.99	2.84
	900x680	57.49	17.39		24.83	29.05	3.61
	1032x778	75.69	13.21		32.76	38.27	4.66

4.1 PIPELINE FOR TASK PARALLELIZATION

In this section a task parallelization strategy for the proposed case study is presented. The processing module could be parallelized using a three stage pipeline where each stage is constituted by one of the three sub-blocks previously identified: pre-processing (including lens distortion correction and color space conversion), the GM algorithm and the post-processing block. The proposed pipeline model and its stages are reported in fig 7.

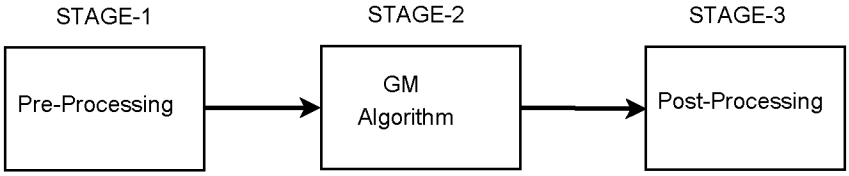

Figure 5. Implemented three stages pipeline.

In table 2 the fr_{max} of the O_{ap} and of the pipeline version of the application (PV_{ap}) are reported for different image sizes and in the case of the luminance color space. The first column represents the image size, the second and the third one are the fr_{max} of the algorithm expressed in fps for O_{ap} and PV_{ap}. As it can be noticed, PV_{ap} leads to a gain of $\approx 90\%$ for all image sizes. By using PV_{ap} the processing task can work with a fr_{max} of about $\approx 25fps$ with full image size. The speed up gain has been calculated as:

$$G_s = \left(\frac{fr_{max}^{new}}{fr_{max}^{orig}} - 1 \right) \bullet 100 \tag{1}$$

where fr_{max}^{new} and fr_{max}^{orig} are respectively the fr_{max} of the application that we want to evaluate and the fr_{max} of the application used as a reference. By Considering eq. 1, a negative value of G_s corresponds to a decrease of the performance.

These results show that the idea to design the processing algorithm in a pipeline fashion is very positive since it allows to speed up the entire application without a great effort from the point of view of the final user.

Table 2. fr_{max}, G_s and overhead of PV_{ap}.

Image Size	fr_{max} O_{ap}	fr_{max} PV_{ap}	G_s (%)	Overhead (ms)
640x480	33.79	64.39	90.57	0.59
800x600	22.02	41.44	88.22	1.14
900x680	17.39	32.37	86.10	1.84
1032x778	13.21	24.98	89.08	1.77

Moreover, these results show that the overhead introduced by the pipeline of the software architecture, presented in the fourth column of table 2, is very low ($\leq 2ms$). The overhead has been calculated as the difference of the pT of PV_{ap} with respect to pT of the ideal pipeline, where the throughput is limited by the GM algorithm. The throughput of the ideal pipeline, in the case of full size images, is $\approx 26fps$. As it can be noticed, with the proposed pipeline the obtained throughput is $\approx 25fps$.

This software architecture and this pipeline model have been applied to all data streams coming from more than one camera simultaneously. Results of these experiments are reported in fig. 6 and table 3. Straight lines fig. 6 represent the $\frac{1}{fr_{max}}$ of O_{ap}, and dashed lines represent $\frac{1}{fr_{max}}$ of PV_{ap}. Different colors are used to identify scenarios with different number of cameras.

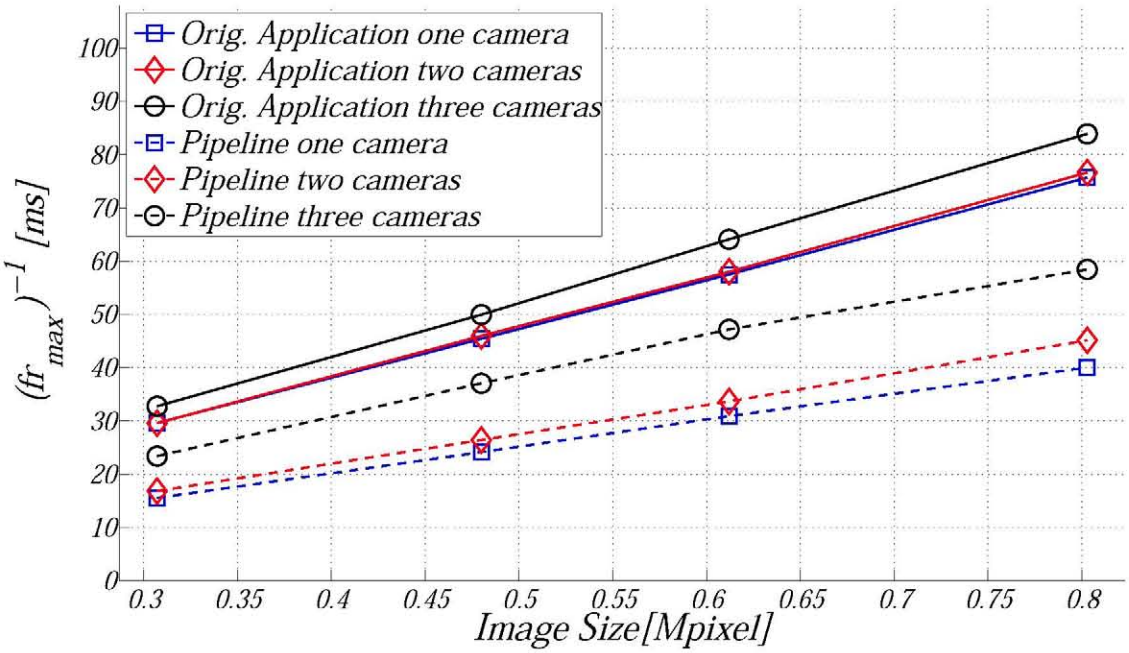

Figure 6. $\frac{1}{fr_{max}}$ of O_{ap}(solid line) and $\frac{1}{fr_{max}}$ of PV_{ap} (dashed line) as a function of the image size. Blue lines refer to one camera scenario, red and black lines refer respectively to two and three camera scenarios

Let us consider the proposed software architecture without the pipeline approach (solid lines in fig. 6), it is possible to see that the software architecture guarantees the same performance (good scalability) with one or two cameras (where blue and red line are overlapped). The three cameras case leads to a increase of the $\frac{1}{fr_{max}}$. This is probably due to the intrinsic limitations of the specific case study. In fact the GM algorithm is not only computational demanding (continuous update of Gaussian parameters), but it requires a lot of available memory since it is necessary to store several parameters for each Gaussian model of each pixel. The software architecture guarantees good performance since a three cameras system could be able to process an image stream of the size 800×600 ($\approx 0.5Mpixel$) with a frame rate of $\approx 20fps$, thus satisfying real time requirements. On the contrary,

the three cameras case is too slow for a real time application when using a full image size (frame rate of $\approx 12fps$); also in this case the performance can be improved using the pipeline approach.

Let us focus on the performances of the proposed software architecture coupled with the pipeline approach: the corresponding curves are depicted with dashed lines in fig. 6. It is important to notice that PV_{ap} with multiple cameras performs even better (obtaining a positive G_s) if compared with O_{ap} using a single camera. In fact, as it can be noticed from fig. 6 dashed lines stay below the solid line corresponding to O_{ap} with one camera. It is worth noting that also in this case the software architecture scales well with the number of cameras.

These results are clearly reported in table 3 where the fr_{max} of the O_{ap} and the PV_{ap} are reported for different image sizes and number of cameras. The G_s is calculate with respect to O_{ap} applied to a single camera (see second column of table 2). As it can be noticed, if O_{ap} is used, a small and negative G_s is obtained for the two cameras system and a negative one is obtained ($\approx 10\%$) in the three cameras system. By using PV_{ap}, positive G_s is always obtained. For example, in the case of three cameras and at the maximum image size, the gain is $\approx 22\%$, this means that the multi-camera system can work with a frame rate of $\approx 17fps$, whereas O_{ap} with one camera has a frame rate of $\approx 13fps$.

Table 3. fr_{max} and G_s in multi-camera scenarios.

# Cameras	Img. Size	fr_{max} O_{ap}	fr_{max} PV_{ap}	G_s O_{ap}	G_s PV_{ap}
2	640x480	33.83	59.43	0.14	75.91
	800x600	21.76	37.85	-1.19	71.91
	900x680	17.24	29.72	-0.88	70.85
	1032x778	13.05	22.16	-1.21	67.74
3	640x480	30.52	42.75	-9.66	26.52
	800x600	20.02	26.98	-9.08	22.55
	900x680	15.60	21.19	-10.33	21.84
	1032x778	11.92	17.11	-9.79	29.51

4.2 HYBRID PIPELINE

In this section, another example is given in order to prove the modularity and flexibility of the proposed architecture and that will demonstrate the possibility to easily re-design and re-integrate a different PM. This example highlights the advantage of using the pipeline approach that combines both data and tasks parallelism.

Let us consider our case study with one camera, full image size and the GM algorithm applied to the entire color space. By following the results obtained in sec. 4.1, it is possible to see that a task parallelization approach would lead theoretically to a fr_{max} very close to $\approx 14fps$, limited by the speed of the GM algorithm.

A more efficient solution could be obtained combining data and task parallelization. In particular, we decide to implement a four stage pipeline where the first stage is composed by the preprocessing sub-block, and the other three pipeline stages apply the GM algorithm and the post-processing tasks to three different regions of the image. The proposed pipeline model and its stages are reported in fig 7.

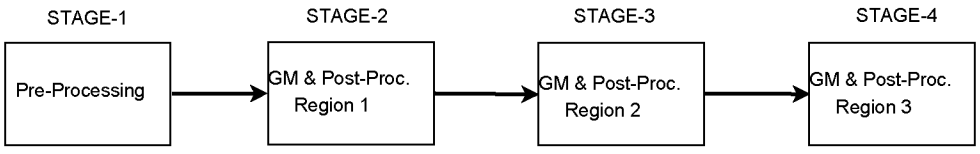

Figure 7. Implemented hybrid pipeline.

Table 4 reports the fr_{max} for different image sizes (first column). The second column contains the fr_{max} of O_{ap}; the third one contains the fr_{max} of PV_{ap}; the fourth one represents the obtained G_s and the last one contains the overhead introduced by the pipeline. As it can be noticed, the designed pipeline allows to reach a considerable value of G_s, always greater than 130%. For example considering a data stream of full size images

with O_{ap} it is possible to manage a stream of $\approx 9fps$, using PV_{ap} it is possible to process a stream of about $\approx 25fps$. Furthermore, it has to be noticed that in the case with full size images, an higher gain is obtained respect to the low resolution case. In the low resolution case the regions on which each pipeline stage operates are probably too small and the overhead effect is much more evident. In this case, it is more suitable to implement a smaller pipeline. The overhead has been calculated with respect to the ideal pipeline limited by the slowest stage. In this case, by splitting the GM algorithm in three parts, the slowest task would become the pre-processing one (see table 4). For instance, let us consider the fifth row (YCbCr case, full size image): the GM algorithm has a pT equal to 68.59 and the pre-processing stage a pT of 33.59; if we split the GM algorithm into three sub-block, having pT equal to 22.86 then the pre-processing stage will be the slowest. As it can be noticed in table 4, the overhead introduced by the pipeline is small with respect to the obtained gain. In this case the introduced overhead is higher with respect to the results shown in table table 2 since this pipeline model has one stage more.

Table 4. fr_{max} and G_s for hybrid pipeline.

Image Size	fr_{max} O_{ap}	fr_{max} PV_{ap}	G_s PV_{ap}	Overhead (ms)
640x480	24.81	58.26	134.79	4.61
800x600	15.88	38.63	143.27	6.18
900x680	12.57	31.60	151.40	6.78
1032x778	9.35	24.90	166.32	6.58

5. CONCLUSIONS AND FUTURE WORKS

Multi-camera systems are used in several computer vision applications. Some systems are based on smart cameras, others are centralized server systems. The first approach lacks of flexibility, the second one does not scale well with the number of cameras. An attractive solution for the multi-camera system is based on a network of peer to peer computers that guarantees a good trade off between flexibility and scalability.

In this paper we have presented a software architecture for easily manage multi-camera systems in single node application. The proposed software architecture, developed in C++, is a scalable, flexible and modular architecture for multi-camera system management in real time scenarios. The architecture is scalable since it can manage and process efficiently multiple data stream adding small overhead and saving useful computational resources. It is scalable since can manage different camera data streams; it is flexible because it can work on different platform and with different acquisition devices and it is very easy to integrate thanks of its high degree of modularity.

Moreover, we propose a strategy that guarantees a trade off between reduction of computational time, flexibility and modularity. We offer the option to integrate the desired application in a *parallel-fashion* in a transparent way from the thread-level programming point of view. In this way it is possible to speed up the processing algorithm enabling data or task parallelism.

The proposed software architecture has been tested in a multi-camera system in order to efficiently manage multiple 2D object detection tasks in a real-time scenario. System performance has been evaluated under different load conditions such as number of cameras and image sizes. Results show that the software architecture can be efficiently used in multi-camera environments for on line video processing. The software architecture scales well with the number of cameras and can easily works with different image formats. The desired application can be easily designed and integrated in the software architecture. Moreover, the parallelization approach can be used in order to speed up the processing tasks.

Future research will be related to the extension of the software architecture in order to manage a more complex multi-camera system, based on a peer to peer network of computers. Furthermore, we will test our system with more sophisticated Central Unit models that allow to perform high level tasks such as the 3D tracking system.

APPENDIX A. MIXTURE OF GAUSSIANS MODEL

One of the most popular approaches to detect moving objects in video sequences is based on background subtraction techniques. The main idea of the background subtraction techniques is to estimate a background model of the scene and consider any deviation from it a moving (foreground) object. In literature several background modeling techniques have been presented[23] .

The algorithm presented by Stauffer and Grimson[14] aims at estimating a background model of the scene as a mixture of Gaussian distributions. The algorithm does not take into account the spatial correlation in the image, since it models each pixel independently. The algorithm is composed by two fundamental steps. The first step is the estimation of whether or not a pixel belongs to the background model. In the second step the Gaussian parameters are recursively updated.

The main idea of the algorithm is to model each pixel of the background as a mixture of Gaussians. The probability to find a pixel at time t of intensity X is defined as:

$$P\left(X_t\right) = \sum_{i=1}^{K} \omega_{i,t} \cdot \eta\left(X_t, \mu_{i,t}, \Sigma_{i,t}\right) \tag{2}$$

Where K is the number of Gaussians, $\omega_{i,t}$ is the weight associated at time t to the i_{th} Gaussian at the time t with mean $\mu_{i,t}$ and the covariance matrix $\Sigma_{i,t}$. If the RGB planes are considered independent the form of the covariance matrix can be considered as $\Sigma_{i,t} = \sigma_{i,t} I$.

In the first step for each incoming pixel there is the estimation of whether or not it belongs to the background model. The K distributions are ordered using as criterion the ratio

$$r_{i,t} = \frac{\omega_{i,t}}{\sigma_{i,t}}. \tag{3}$$

The first B distributions that exceed a certain threshold T are used for the background model:

$$B = arg\min_{b}\left(\sum_{i=1}^{b} \omega_{i,t} \geq T\right) \tag{4}$$

T is a measure of the minimum portion of the data that should be accounted for by the background. For small value of T is obtained a background modeled by few distribution, at the limit a unimodal Gaussian distribution. I T is higher is obtained a multi-modal background model, that can include more than one color in the background model.

A pixel, at time $t+1$, belongs to one of the K distributions if the equation (5) is satisfied:

$$\sqrt{\left(X_{t+1} - \mu_{i,t}\right)^T \Sigma_{i,t}^{-1} \left(X_{t+1} - \mu_{i,t}\right)} < 2.5\sigma_{i,t} \tag{5}$$

If the pixel belongs to one of the background distributions it will be classified as a background pixel, otherwise it is classified as a foreground pixel. If a match is found, the parameters of the matching Gaussian are updated with the following equations:

$$\omega_{i,t+1} = \omega_{i,t}(1-\alpha) + \alpha \tag{6}$$

$$\rho = \alpha \cdot \eta\left(X_{t+1}, \mu_{i,t}, \Sigma_{i,t}\right) \tag{7}$$

$$\mu_{i,t+1} = \mu_{i,t}(1-\rho) + \rho X_{t+1} \tag{8}$$

$$\sigma_{i,t+1}^2 = \sigma_{i,t}^2(1-\rho) + \rho\left(X_{t+1} - \mu_{i,t+1}\right)\left(X_{t+1} - \mu_{i,t+1}\right) \tag{9}$$

where α is the so called learning rate. The learning rate α determines the speed of adaptation to changes in the scene (i.e, illumination) and the speed of the incorporation of foreground objects to the background. For the unmatched Gaussians all their parameters remain unchanged except the weight:

$$\omega_{i,t+1} = \omega_{i,t}(1 - \alpha) \tag{10}$$

If no match is found the least probable distribution is substituted by a distribution with a low weight, a high variance and a mean equal to the pixel value. When all the parameters have been update, the weights are normalized in order to obtain $\sum_{i=1}^{K} \omega_{i,t+1} = 1$.

Many variations of the algorithm have been presented in order to overcome different problems like, sudden changes of illumination, shadows of moving objects, real time constraints, memory requirements etc. For a complete review of these algorithms see the review of Bouwmans et al.[30] . However we have implemented the original proposal of Stauffer and Grimson except for a small change in the parameters update. In fact, in our experiments we slightly modified the original algorithm in order to reduce the processing time. Following the solution proposed by Power and Schoonees[31] the equation (7) is substituted by:

$$\rho = \frac{\alpha}{\omega_{i,t}} \tag{11}$$

ACKNOWLEDGEMENTS

This work has been supported by the Ministerio de Ciencia e Innovacin of the Spanish Government under project TEC2007-67764 (SmartVision).

REFERENCES

[1] Collins, R., Lipton, A., Kanade, T., Fujiyoshi, H., Duggins, D., Tsin, Y., Tolliver, D., Enomoto, N., and Hasegawa, O., "A system for video surveillance and monitoring," Tech. Rep. CMU-RI-TR-00-12, Robotics Institute, Pittsburgh, PA (May 2000).

[2] Bellotto, N., Sommerlade, E., Benfold, B., Bibby, C., Reid, I., Roth, D., Fernández, C., Gool, L. V., and Gonzàlez, J., "A distributed camera system for multi-resolution surveillance," in [*Proc. of the 3rd ACM/IEEE Int. Conf. on Distributed Smart Cameras (ICDSC)*], (2009).

[3] Stoykova, E., Alatan, A., Benzie, P., Grammalidis, N., Malassiotis, S., Ostermann, J., Piekh, S., Sainov, V., Theobalt, C., Thevar, T., and Zabulis, X., "3-d time-varying scene capture technologies—a survey," *Circuits and Systems for Video Technology, IEEE Transactions on* **17**, 1568 –1586 (nov. 2007).

[4] Svoboda, T., Hug, H., and Gool, L. V., "Viroom - low cost synchronized multicamera system and its self-calibration," in [*Pattern Recognition, 24th DAGM Symposium, number 2449 in LNCS*], 515–522, Springer (2002).

[5] Baker, H. H., Bhatti, N., Tanguay, D., Sobel, I., Gelb, D., Goss, M. E., Culbertson, W. B., and Malzbender, T., "Understanding performance in coliseum, an immersive videoconferencing system," *ACM Trans. Multimedia Comput. Commun. Appl.* **1**(2), 190–210 (2005).

[6] Lin, C. H., Wolf, M., Koutsoukos, X., Neema, S., and Sztipanovits, J., "System and software architectures of distributed smart cameras," *ACM Trans. Embed. Comput. Syst.* **9**(4), 1–30 (2010).

[7] Appiah, K., Hunter, A., Owens, J., Aiken, P., and Lewis, K., "Autonomous real-time surveillance system with distributed ip cameras," in [*Distributed Smart Cameras, 2009. ICDSC 2009. Third ACM/IEEE International Conference on*], 1 –8 (sept. 2009).

[8] Rinner, B., Winkler, T., Schriebl, W., Quaritsch, M., and Wolf, W., "The evolution from single to pervasive smart cameras," in [*Proc. Second ACM/IEEE Int. Conf. Distributed Smart Cameras ICDSC 2008*], 1–10 (2008).

[9] Valera, M. and Velastin, S. A., "Intelligent distributed surveillance systems: a review," *IEE Proceedings -Vision, Image and Signal Processing* **152**(2), 192–204 (2005).

[10] Straw, A. and Dickinson, M., "Motmot, an open-source toolkit for realtime video acquisition and analysis," *Source Code for Biology and Medicine* **4**(1), 5 (2009).

[11] Wang, S., Bevans, A., and Antle, A. N., "Stitchrv: multi-camera fiducial tracking," in [*TEI '10: Proceedings of the fourth international conference on Tangible, embedded, and embodied interaction*], 287–290, ACM, New York, NY, USA (2010).

[12] Doubek, P., Svoboda, T., and Van Gool, L., "Monkeys a software architecture for viroom low-cost multicamera system," in [*Computer Vision Systems*], Crowley, J., Piater, J., Vincze, M., and Paletta, L., eds., *Lecture Notes in Computer Science* **2626**, 386–395, Springer Berlin / Heidelberg (2003).

[13] Vezzani, R. and Cucchiara, R., "Event driven software architecture for multi-camera and distributed surveillance research systems," in [*Computer Vision and Pattern Recognition Workshops (CVPRW), 2010 IEEE Computer Society Conference on*], 1 –8 (2010).

[14] Stauffer, C. and Grimson, W. E. L., "Adaptive background mixture models for real-time tracking," *Computer Vision and Pattern Recognition, IEEE Computer Society Conference on* **2**, 2246–252 Vol. 2 (August 1999).

[15] "Opencv 2.1 reference guide." `http://opencv.willowgarage.com/documentation/cpp/index.html`.

[16] Camplani, M. and Salgado, L., "Adaptive multi-camera system for real time object detection," in [*Consumer Electronics (ICCE), 201 Digest of Technical Papers International Conference on*], (2011).

[17] `http://www.jai.com`.

[18] `http://www.axis.com`.

[19] `http://www.ntp.org/`.

[20] `http://www.threadingbuildingblocks.org/`.

[21] Mohedano, R., del Bianco, C., Jaureguizar, F., Salgado, L., and Garcia, N., "Robust 3d people tracking and positioning system in a semi-overlapped multi-camera environment," in [*Image Processing, 2008. ICIP 2008. 15th IEEE International Conference on*], 2656 –2659 (12-15 2008).

[22] Mohedano, R. and Garcia, N., "Robust multi-camera 3d tracking from mono-camera 2d tracking using bayesian association," *IEEE Transaction on Consumer Electronics* **56**(1), 1–8 (2010).

[23] Piccardi, M., "Background subtraction techniques: a review," *Systems, Man and Cybernetics, 2004 IEEE International Conference on* **4**, 3099 – 3104 vol.4 (oct. 2004).

[24] "Gti camera calibration toolbox user manual," tech. rep., G.T.I. Image Processing Group, Universidad Politcnica de Madrid, Spain (2010).

[25] Hartley, R. I. and Zisserman, A., [*Multiple View Geometry in Computer Vision*], Cambridge University Press, ISBN: 0521540518, second ed. (2004).

[26] Ribeiro, H. and Gonzaga, A., "Hand image segmentation in video sequence by gmm: a comparative analysis," in [*Computer Graphics and Image Processing, 2006. SIBGRAPI '06. 19th Brazilian Symposium on*], 357 –364 (8-11 2006).

[27] Parks, D. and Fels, S., "Evaluation of background subtraction algorithms with post-processing," in [*Advanced Video and Signal Based Surveillance, 2008. AVSS '08. IEEE Fifth International Conference on*], 192 –199 (1-3 2008).

[28] `http://www.jai.com/Pages/RegisterToDownload.aspx?ReturnUrl=/ProtectedDocuments/Manuals/Manual_CM-080GE_CB-080GE.pdf`.

[29] `http://www.intel.com/p/en_US/products/server/processor/xeon5000`.

[30] Bouwmans, T., El Baf, F., and Vachon, B., "Background modeling using mixture of gaussian for foreground detection," *Recent Patent on Computer Science* **1** (2008).

[31] Power, W. P. and Schoonees, J. A., "Understanding background mixture models for foreground segmentation," *Imaging and Vision Computing NewZeland 2002* (nov. 2002).