Università degli Studi di Salerno

Dipartimento di Ingegneria dell'Informazione ed Elettrica e Matematica Applicata/DIEM

Dottorato di Ricerca in Informatica e Ingegneria dell'Informazione
XXIX Ciclo

TESI DI DOTTORATO

A Data-Flow Middleware Platform for Real-Time Video Analysis

CANDIDATO: **ROSARIO DI LASCIO**

COORDINATORE: **PROF. ALFREDO DE SANTIS**

TUTOR: **PROF. MARIO VENTO**

Anno Accademico 2016 -2017

For those who will come

Abstract

In this thesis we introduce a new software platform for the development of real-time video analysis applications, that has been designed to simplify the realization and the deployment of intelligent video-surveillance systems. The platform has been developed following the Plugin Design Pattern: there is an application-independent middleware, providing general purpose services, and a collection of dynamically loaded modules (plugins) carrying out domain-specific tasks. Each plugin defines a set of node types, that can be instantiated to form a processing network, according to the data-flow paradigm: the control of the execution flow is not wired in the application-specific code but is demanded to the middleware, which activates each node as soon as its inputs are available and a processor is ready. A first benefit of this architecture is its impact on the software development process: the plugins are loosely coupled components that are easier to develop and test, and easier to reuse in a different project. A second benefit, due to the shift of the execution control to the middleware, is the performance improvement, since the middleware can automatically parallelize the processing using the available processors or cores, as well as using the same information or data for different thread of execution. In order to validate the proposed software architecture, in terms of both performance and services provided by the middleware, we have undertaken the porting to the new middleware of two novel intelligent surveillance applications, by implementing all the nodes required by the algorithms.

The first application is an intelligent video surveillance system based on people tracking algorithm. The application uses a single, fixed camera; on the video stream produced by the camera, background subtraction is performed (with a dynamically updated background) to detect foreground objects. These objects are tracked, and their trajectories are used to detect events of interest, like entering a forbidden area, transiting on a one-way passage in the wrong direction, abandoning objects and so on. The second application integrated is a fire detection algorithm, which combines information based on color, shape and movement in order to detect the flame. Two main novelties have been introduced: first, complementary information, respectively based on color, shape variation and motion analysis, are combined by a multi expert system. The main advantage deriving from this approach lies in the fact that the overall performance of the system significantly increases with a relatively small effort made by designer. Second, a novel descriptor based on a bag-of-words approach has been proposed for representing motion. The proposed method has been tested on a very large dataset of fire videos acquired both in real environments and from the web. The obtained results confirm a consistent reduction in the number of false positives, without paying in terms of accuracy or renouncing the possibility to run the system on embedded platforms.

Contents

Chapter 1

Introduction

Real-time video processing and analysis is a requirement in several application domains, such as environment monitoring, industrial process control, traffic monitoring, robotics. Among these domains, intelligent video-surveillance has seen a steadfastly growing interest in both the research community and the industry. This is due in part to an increased concern for security problems, but also to the availability of larger and larger numbers of affordable surveillance cameras, producing a huge supply of video streams that would be unpractical and uneconomical to monitor using only human operators, in order to detect security-relevant events.

While basic applications (such as motion detection) are now commonplace, being often integrated into the camera firmware, the detection of more complex events involves the use of techniques that are often at the frontier of the research on video analysis. For each phase of the process, there is no single algorithm that has proved to be the best under any operating condition, so the set of techniques that have to be put together to build a solution may vary greatly depending on the specific application context. Furthermore, these techniques usually have to be finely tuned on each video stream to achieve a reasonable performance. Another aspect that cannot be neglected in video-surveillance application is the computational cost, since it is desirable that each processing station is able to analyze several video streams in real time; this usually entails the need to exploit some form of parallelism (e.g. in multi-processor systems, or in multi-core processors), which in turn raises thorny issues such as scheduling or synchronization that, if not properly dealt with, may easily lead to subtle bugs and unpredictable anomalies.

Traditionally, these problems have been faced using *ad hoc* techniques for both the design and the implementation. Often these techniques sacrifice on the altar of performance other important software qualities such as modularity, reusability, flexibility and extendability; furthermore they usually make the software tightly tied to a particular architecture, hindering the transition to a different hardware platform that may be required for scalability.

This thesis aims to bridge over these considerations: it introduces a new software platform for the development of real-time video analysis applications, that has been designed, based on the data-flow paradigm, to simplify the realization and the deployment of intelligent video-surveillance systems.

1.1 Overview

Dataflow is a computer architecture that directly contrasts the traditional von Neumann or control flow architecture; this type of architecture, according to the taxonomy of Flynn, falls within the family MIMD (Multiple Instruction Multiple Data). Dataflow architectures do not have a program counter, or (at least conceptually) the executability and execution of instructions is solely determined based on the availability of input arguments to the instructions, so that the order of instruction execution is unpredictable: i. e. behavior is indeterministic.

Although no commercially successful general-purpose computer hardware has used a dataflow architecture, it has been successfully implemented in specialized hardware such as in digital signal processing, network routing, graphics processing, telemetry, and more recently in data warehousing.

About the world of the software, Dataflow is a software paradigm based on the idea of disconnecting computational actors into stages (pipelines) that can execute concurrently and it finds broad application as a conceptual model in the development of database software architectures and, more generally, for the development of parallel applications. Dataflow can also be called stream processing or reactive programming. The most obvious example of data-flow programming is the subset known as reactive programming with spreadsheets. As a user enters new values, they are instantly transmitted to the next logical actor or formula for calculation. It is also very relevant in many software architectures today including database engine designs and parallel computing frameworks.

The proposed platform has been developed following the Plugin Design Pattern: there is an application-independent middleware, providing general purpose services, and a collection of dynamically loaded modules (plugins) carrying out domain-specific tasks. Each plugin defines a set of node types, that can be instantiated to form a processing network, according to the dataflow paradigm: the control of the execution flow is not wired in the application-specific code but is demanded to the middleware, which activates each node as soon as its inputs are available and a processor is ready.

A first benefit of this architecture is its impact on the software development process: the plugins are loosely coupled components that are easier to develop and test, and easier to reuse in a different project. A second benefit, due to the shift of the execution control to the middleware, is the performance improvement, since the middleware can automatically parallelize the processing using the available processors or cores, as well as using the same information or data for different thread of execution.

In order to validate the proposed software architecture, in terms of both performance and services provided by the middleware, we have undertaken the porting to the new middleware of two novel intelligent surveillance applications, by implementing all the nodes required by the algorithms.

The first application is an intelligent video surveillance system based on people tracking [1]. The application uses a single, fixed camera; on the video stream produced by the camera, background subtraction is performed (with a dynamically updated background) to detect foreground objects. These objects are tracked, and their trajectories are used to detect events of interest, like entering a forbidden area, transiting on a one-way passage in the wrong direction, abandoning objects and so on.

The second application integrated is a fire detection algorithm [2], which combines information based on color, shape and movement in order to detect the flame.

1.2 Organization

This thesis consists of the following chapters:

- **Chapter 1** presents an overview of this research, by analyzing the problem and briefly highlighting the novelties of the proposed approach.
- **Chapter 2** provides an overview of the state of the art methods, by deeply investigating approaches recently proposed for tracking moving objects in Section 2.1 and fire detection algorithm in Section 2.2. The chapter is concluded by Section 2.3, which analyzes the state of the art of Data-flow Platform used for video surveillance application.
- **Chapter 3** details the proposed tracking algorithm;
- **Chapter 4** is devoted to analyze the method proposed for recognizing fire inside a scene.
- **Chapter 5** details the proposed Data-flow architecture and its modules.
- **Chapter 6** analyzes the results obtained by the proposed video analytics algorithms and by the proposed data-flow architecture.
- **Chapter 7** draws some conclusions.

Chapter 2

State of the Art

The software platform introduced in Chapter 1 and the video surveillance applications plugin ported inside the middleware to test the whole system are really challenging: as briefly analyzed, each part composing the system is characterized by its intrinsic problems. For the above mentioned reason, in this chapter the state of the art of each module is separately analyzed, so to better highlight novelties and advances with respect to the state of the art introduced in this thesis.

This chapter is organized as follows:

- **Section 2.1**: state of the art of the people tracking algorithm.
- **Section 2.2**: state of the art of the fire detection algorithm.
- **Section 2.3**: state of the art of Data-flow Platform used for video surveillance application, by analyzing the problem and briefly highlighting the novelties of the proposed approach.

2.1 Tracking algorithm

The tracking problem is deceptively simple to formulate: given a video sequence containing one or more moving objects, the desired result is the set of trajectories of these objects. This result can be achieved by finding the best correspondence between the objects tracked until the previous frames and the ones identified by the detection phase (from now on the *blobs*) at the current frame, as shown in Figure 2.1. The ideal situation is shown in Figure 2.1a, where each person is associated to a single blob.

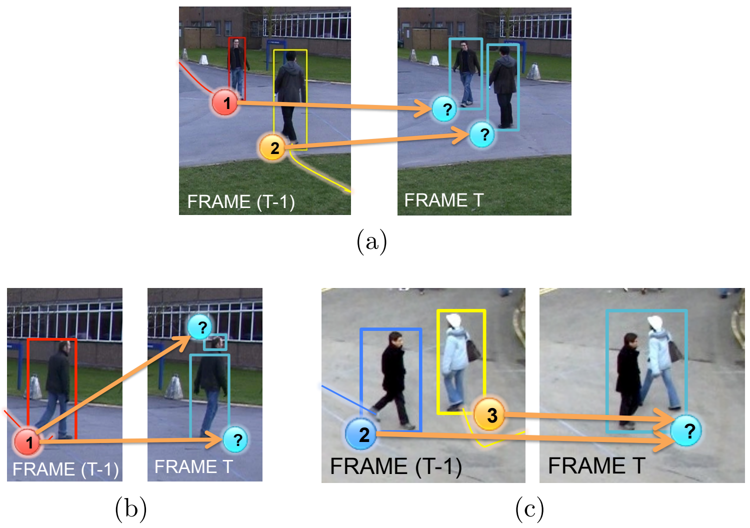


Figure 2.1 Each image is composed as follows: on the left the objects tracked until the previous frame $t-1$, on the right the blobs identified during the detection step at the current frame t. The aim of the tracking algorithm is to perform the best association, identified by the orange arrows: in (a) the ideal situation is shown, in (b) the detection splits the person in two parts while in (c) the persons are merged in a single blob because of an occlusion pattern.

Unfortunately, in real world scenarios, there are several issues that make this result far from being easy to achieve: the detection may split a person in several blobs, as shown in Figure 2.1b, or an occlusion pattern may merge two or more persons in a single

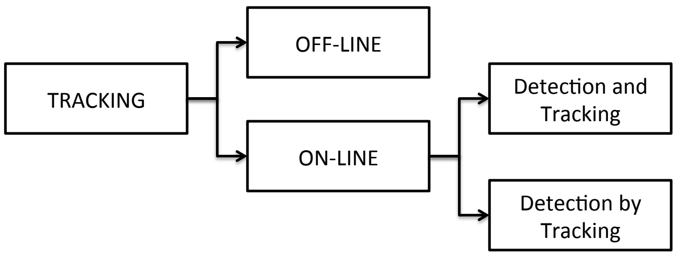


Figure 2.2 Trajectories extraction: the state of the art methods.

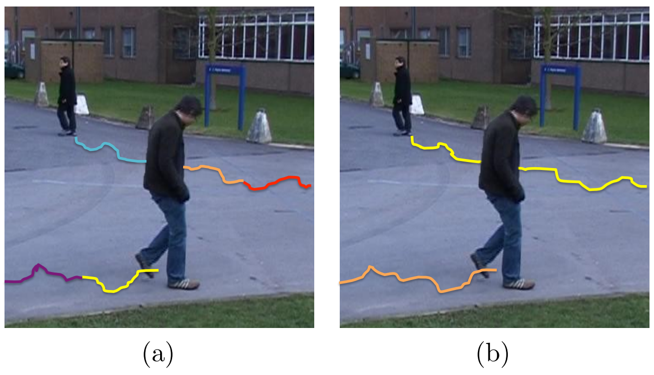


Figure 2.3 In (a) the tracklets extracted at the time instant t by using off-line tracking algorithms. The entire trajectories will be produced only when all the tracklets will be available. In (b) trajectories extracted at the same time instant by on-line algorithms.

blob, as shown in Figure 2.1c. It is evident that in such situations more complex associations need to be managed.

Because of these difficulties, many tracking algorithms have been proposed in the last years, but the problem is still considered open.

Tracking algorithms can be divided into two main categories, as summarized in Figure 2.2: the former category (off-line) contains the methods [3][4][5] which extract small but reliable pieces of trajectories, namely the tracklets, instead of entire trajectories. Once all the tracklets are available, the system needs a post-processing step aimed at linking the ones belonging to the same individual for extracting the final trajectories. An example is shown in Figure 2.3a.

The latter category (on-line) contains all those algorithms performing the analysis on-line, so that at each time instant t the entire trajectory of each object is available until t and ready to be used, as shown in Figure 2.3b. In this category, two main strategies have been proposed up to now:

- Detection and tracking: the tracking is performed after an object detection phase; in particular, objects are detected in each frame using a priori model of the objects or some form of change detection: differences from a background model, differences between adjacent frames, motion detection through optical flow and so on. Algorithms following this strategy are usually faster than the ones belonging to the other strategy, but they have to deal also with the errors of the detection phase as spurious and missing objects, objects split into pieces, multiple objects merged into a single detected *blob*.

- Detection-by-Tracking: detection and tracking are performed at once, usually on the basis of an object model that is dynamically updated during the tracking.

Some examples of algorithms following the first strategy (detection and tracking) are [6] and [7]: the criterion used to find a correspondence between the evidence at the current frame and the objects at the previous one is based on the overlap of the areas. Overlap-based methods work well with high frame rates and when objects do not move very fast, but might fail in other conditions. Positional information, obtained by taking advantage of the Kalman filter, is also used in [8] and [9]. In the former, only the distance between the detected blob and the predicted position is considered; on the contrary, in the latter the appearance information is taken into account by means of a smoothed 4D color histogram.

Dai *et al.* [10] have proposed a method able to track pedestrians by using shape and appearance information extracted from infra-red images. The method may have some problems when objects quickly change their appearance or during occlusions.

The method proposed in [11] formulates the tracking problem as a bipartite graph matching, solving it with the well-known Hungarian algorithm. It recognizes an occlusion, but is able to preserve the object identities only if the horizontal projection of the detected blob shows a separate mode. The Hungarian algorithm is also used in [12] and [13] in order to solve the assignment problem. In general, its main drawback lies in the polynomial time needed for the computation, which prevents these tracking methods from managing crowded scenarios.

The method by Pellegrini *et al.* [14] tries to predict the trajectories on the scene using a set of behavior models learned using a training video sequence. The method is very effective for repetitive behaviors, but may have some problems for behaviors that do not occur frequently.

Several recent methods [15][16][17] use the information from different cameras with overlapping fields of view in order to perform the occlusion resolution. The data provided by each camera are usually combined using a probabilistic framework to solve ambiguities. These methods, although increasing the reliability of the entire system, are limited to situations where multiple cameras can be installed; furthermore, most of the methods adopting this approach requires a full calibration of each camera, which could make the deployment of the system more complicated.

On the other hand, methods belonging to the second strategy (Detection-by-Tracking) are computationally more expensive, and often have problems with the initial definition of the object models, that in some cases have to be provided by hand.

The method by Bhuvaneswari and Abdul Rauf [18] uses edge-based features called *edgelets* and a set of classifiers to recognize partially occluded humans; the tracking is based on the use of a Kalman filter. The method does not handle total occlusions, and, because of the Kalman filter, it works better if people are moving with uniform direction and speed. The method proposed by Han *et al.* [19] detects and tracks objects by using a set of features, assigned with different confidence levels. The features are obtained by combining color histograms and gradient orientation histograms, which give a representation of both color and contour. The method is not able to handle large scale changes of target objects.

A recent, promising trend in tracking algorithms is the use of machine learning techniques. As an example, the method by Song *et al.* [20] improves the ability of tracking objects within an occlusion by training a classifier for each target when the target is not occluded. These individual object classifiers are a way of incorporating the past history of the target in the tracking decision. However, the method assumes that each object enters the scene un-occluded; furthermore, it is based on the Particle Filters framework, and so it is computationally expensive. Another example is the method by Wang *et al.* [21] that uses *manifold learning* to build a model of different pedestrian postures and orientations; this model is used in the tracking phase by generating for each object of the previous frame a set of candidate positions in the current frame, and choosing the closer candidate according to the model.

The high computational effort required by Detection-by-Tracking algorithms is the main limitation for using such strategy in the proposed system. On the other hand, it is needed that at each time instant the entire trajectory is available in order to properly interpret the event associated to it, so making also unfeasible off-line algorithms. For these reasons, the tracking algorithm proposed in this thesis is based on the first of the above mentioned strategy (Detection and Tracking): it assumes that an object detection based on background subtraction generates its input data. However, one of the main limitations of the existing algorithms using a Detection and Tracking strategy lies in the fact that they make their tracking decisions by comparing the evidence at the current frame with the objects known at the previous one; all the objects are processed in the same way, ignoring their past history that can give useful hints on how they should be tracked: for instance, for objects stable in the scene, information such as their appearance should be considered more reliable. To exploit this idea, the proposed algorithm adopts an object model based on a set of scenarios, in order to deal differently with objects depending on their recent history and conditions; scenarios are implemented by means of a Finite State Automaton, that describes the different states of an object and the conditions triggering the transition to a different state. The state is used both to influence which processing steps are performed on each object, and to choose the most appropriate value for some of the parameters involved in the processing.

It is worth pointing out that another important advantage deriving by the use of a strategy based on Detection and Tracking is that the variables characterizing the tracking process are explicitly defined and easily understandable, and so can be used in the definition and in the manipulation of the state; an approach of the second kind (Detection by Tracking), especially if based on machine learning, would have hidden at least part of the state, and thus the history of the objects would not have been explicitly manageable through a mechanism such as the Finite State Automaton.

Furthermore, although a limited a priori knowledge about the objects of interest is required, in order to differentiate between single objects and groups, the method can be rapidly adapted to other application domains by providing a small number of object examples of the various classes. Finally, the spatio-temporal continuity of moving objects is taken into account using a graph-based approach. Thanks to this novel approach, our system is able to simultaneously track single objects and groups of objects, so significantly increasing the overall reliability of the proposed approach.

More details about the proposed tracking algorithm will be provided in Chapter 3.

2.2 Fire detection

In the last years several methods have been proposed, with the aim to analyze the videos acquired by traditional video-surveillance cameras and detect fires or smoke, and the current scientific effort [22][23] focused on improving the robustness and performance of the proposed approaches, so as to make possible a commercial exploitation.

Although a strict classification of the methods is not simple, two main classes can be distinguished, depending on the analyzed features: color based and motion based. The methods using the first kind of features are based on the consideration that a flame, under the assumption that it is generated by common combustibles as wood, plastic, paper or other, can be reliably characterized by its color, so that the evaluation of the color components (in RGB, YUV or any other color space) is adequately robust to identify the presence of flames. This simple idea inspires several recent methods: for instance, in [24] and [25] fire pixels are recognized by an advanced background subtraction technique and a statistical RGB color model: a set of images have been used and a region of the color space has been experimentally identified, so that if a pixel belongs to this particular region, then it can be classified as fire. The main advantage of such algorithms lies in the low computational cost allowing the processing of more than 30 frames per second at QCIF (176x144) image resolution. Differently from [24] and [25], in [26] the authors experimentally define a set of rules for filtering fire pixels in the HSI color space. The introduction of the HSI color space significantly simplifies the definition of the rules for the designer, being more suitable for providing a people-oriented way of describing the color. A similar approach has been used in [27], where a cumulative fire matrix has been defined by combining RGB color and HSV saturation: in particular, starting from the assumption that the green component of the fire pixels has a wide range of changes if compared with red and blue ones, this method evaluates the spatial color variation in pixel values in order to distinguish non-fire moving objects from uncontrolled fires.

The common limitation of the above mentioned approaches is that they are particularly sensitive to changes in brightness, so causing a high number of false positive due to the presence of shadows or to different tonalities of the red. This problem can be mitigated by switching to a YUV color space. In [28], for instance, a set of rules in the YUV space has been experimentally defined to separate the luminance from the chrominance more effectively than in RGB, so reducing the number of false positives detected by the system. In [29] information coming from YUV color are combined using a fuzzy logic approach in order to take into account the implicit uncertainties of the rules introduced for thresholding the image. A probabilistic approach based on YUV has been also exploited in [30], where the thresholding of potential fire pixels is not based on a simple heuristic but instead on a support vector machine, able to provide a good generalization without requiring problem domain knowledge. Although this algorithm is less sensitive to variations in the luminance of the environment, its main drawback if compared with other color based approaches lies in the high computational cost required as soon as the dimensions of the support vector increase.

In conclusion, it can be observed that the methods using color information, although being intrinsically simple to configure, can be successfully used only in sterile areas, where no objects generally move inside. In fact, their main limitation concerns the number of false positives when used in normal populated areas: persons with red clothes or red vehicles might be wrongly detected as fire only because of their dominant color. In order to face this issue, in the last years several approaches have been proposed: they start from the assumption that a flame continuously changes its shape and the disordered movement of red colored regions can help in distinguishing it from rigid objects moving in the scene. For instance, in [31] the physical properties of the fire are used to build a feature vector based on an enhanced optical flow, able to analyze in different ways both the dynamic texture of the fire and its saturated flame. Dynamic textures have also been used in [32], where a two-phase texture detection process has been proposed in order to speed-up the segmentation step, very useful to extract a wide set of shape-based features, and making possible the detection of the fire in a reasonable time. In [33] the irregularity of the fire over time is handled by combining the capabilities of finite state automata with fuzzy logic: variations in wavelet energy, motion orientation and intensity are used to generate probability density functions, which determine the state transitions of a fuzzy finite automaton.

The Wavelet transform has been also used in [34] in order to properly detect the temporal behavior of flame boundaries. It is worth pointing out that the methods based on the wavelet transform, differently from those based on the color, cannot be used on still images and in general require a frame rate sufficiently high, higher than 20 fps, to guarantee satisfactory results, so limiting their applicability.

In [35] frame-to-frame changes are analyzed and the evolution of a set of features based on color, area size, surface coarseness, boundary roughness and skewness is evaluated by a Bayesian classifier. The wide set of considered features allows the system to take into account several aspects of fire, related to both color and appearance variation, so increasing the reliability in the detection. In [36] the thresholding on the color, performed in the RGB space, is improved by a multi resolution two-dimensional wavelet analysis, which evaluates both the energy and the shape variations in order to further decrease the number of false positive events. In particular, the shape variation is computed by evaluating the ratio between the perimeter and the area of the minimum bounding box enclosing the candidates fire pixels. This last strategy is as simple and intuitive as promising if the scene is populated by rigid objects, such as vehicles. On the other side, it is worth pointing out that the shape associated to non rigid objects, such as people, is highly variable in consecutive frames: think, for instance, to the human arms, that may contribute to significantly modify the size of the minimum bounding box enclosing the whole person. This evidence implies that the disordered shape of the person may be confused with the disordered shape of the fire, so consistently increasing the number of false positives detected by the system.

So, in conclusion, the main limitation of motion based approaches lies in the fact that the performance improvement is often paid from different points of view: first, in most of the cases, several sensitive parameters need to be properly set for the application at hand. Second, the motion and the shape of the flame is somehow dependent on the burning material, as well as on the weather conditions (think, as an example, of a strong wind moving the fire).

In [37] a novel descriptor based on spatio-temporal properties is introduced. First, a set of 3D blocks is built by dividing the image into 16×16 squares and considering each square for a number of frames corresponding to the frame rate. The blocks are quickly filtered using a simple color model of the flame pixels. Then, on the remaining blocks a feature vector is computed using the covariance matrix of 10 properties related to color and to spatial and temporal derivatives of the intensity. Finally, an SVM classifier is applied to these vectors to distinguish fire from non-fire blocks. The main advantage deriving from this choice is that the method does not require background subtraction, and thus can be applied also to moving cameras. However, since the motion information is only taken into account by considering the temporal derivatives of pixels, without an estimation of the motion direction, the system, when working in non sterile areas, may generate false positives due, for instance, to flashing red lights.

The idea of combining several classifiers to obtain a more reliable decision has been generalized and extended in a theoretically sounder way in the pioneering paper [38]. Fire colored pixels are identified by using a Hidden Markov Model (HMM); temporal wavelet analysis is used for detecting the pixel flicker; spatial wavelet analysis is used for the non-uniform texture of flames; finally, wavelet analysis of the object contours is used to detect the irregular shape of the fire. The decisions taken by the above mentioned algorithms are linearly combined by a set of weights which are updated with a least-mean-square (LMS) strategy each time a ground-truth value is available. This method has the advantage that during its operation, it can exploit occasional feedback from the user to improve the weights of the combination function. However, a drawback is the need to properly choose the learning rate parameter, in order to ensure that the update of the weights converges, and that it does so in a reasonable time.

2.3 Data Flow Architecture

As mentioned in Section 1, in this thesis we present our experience with a modular, component-based architecture, based on a middleware that we have specifically designed for the development of real-time video-surveillance applications and, more generally, video analysis systems. The middleware takes care of aspects such as the memory management, the scheduling or the synchronization, while the application logic is realized by assembling loosely-coupled, indepently developed software components that are plugged in the application at run time.

An early platform showing some common points with our architecture is MIT's VuSystem [39]. In this system, a video processing application is split into *modules*, each having a set of input and output ports. Modules are instantiated and interconnected using a script written in the TCL language. Modules communicate with each other exchanging *payloads*, that are objects residing in shared memory, with a *descriptor* identifying the type of data contained in the payload; extending the supported types requires a modification to the source code of the platform. Input/output ports are dynamically typed, so the platform cannot ensure the compatibility of interconnected modules; each module has to check if the payloads it receives are of the right type. The whole application runs in a single-threaded process, and so cannot benefit from parallelism available on multiprocessor system. As a consequence, there are no synchronization issues, and the platform uses a simple event-driven approach for the scheduling.

François and Medioni [40] proposed the *Flow Scheduling Framework* (FSF) as a middleware for real-time video processing. In FSF, computation is modeled using a set of processing elements called *Xcells* that interact with data streams in a data-flow oriented architecture. Xcells are activated by a scheduler when they are ready to be executed, and their execution can be carried out in parallel. The framework requires that the types of the data streams are defined in C++, and the Xcell implementations must be compiled together in order to allow type checking. Also, the interconnections between the Xcells are defined at compile time, thus preventing the possibility of altering the processing network as part of the configuration process. In successive papers [41, 42, 43], François presents the *Software Architecture for Immersipresence* (SAI), which is based on an extension of FSF supporting shared memory communication in addition to data-flow oriented message passing, and including a library of cells for common video processing operations.

Granmo [44] presents a parallel, distributed architecture for Video Content Analysis. The proposed architecture does not address in a general way the video processing operations, but is specifically aimed at the parallelization of Dynamic Bayesian Networks (DBN) and Particle Filters (PF), with provision for exchanging information about the computed approximate probability distributions among the processing nodes.

Another proposal, specifically developed for surveillance and monitoring applications, is *VSIP*, the platform presented by Avanzi et al. [45]. This platform has the main goal of ensuring modularity and flexibility of the applications. To this aim, the application is divided in *modules*, which exchange data through a *shared data manager*, that acts conceptually as a *blackboard system*. Although the authors suggest that by suitably implementing the data manager an application can be distributed among several processing nodes, it is not clear whether they have actually realized such a system, since parallelism was not the main focus of their research. Also, the set of data types that can be exchanged are wired in the platform, and so its source needs to be modified to adopt it for a different kind of application (e.g. audio-surveillance). A later paper by Georis et al. [46] extends the VSIP platform by adding a control and reasoning layer based on a declarative representation of knowledge about the system. Among the benefits of this addition, the platform is able to generate automatically the C++ glue code for assembling an application that integrates existing modules, starting from a high-level description of the desired application in the *YAKL* knowledge representation language and from a knowledge base, represented as a set of declarative rules about the modules behaviors and requirements.

The paper by Desurmont et al. [47] presents a modular distributed architecture for video security. The focus is on the distribution of the application workload over a cluster of computers connected by a network; thus, the application is divided into coarse-grained modules running on different computers, that exchange data by means of a lightweight protocol built upon UDP. The whole video analysis task is incorporated in a single module, and it cannot be parallelized; however, the architecture can exploit the cluster parallelism to process multiple video streams simultaneously.

In [48] Thoren presents *Phission*, a framework for developing vision-based applications for robots. The framework is implemented in C++, and includes support for multithreading and inter-thread communication. Phission adopts the *Pipes and filters* architecture pattern, assuming that the processing can be viewed as a pipeline in which each processing node has one input and one output, although multiple pipelines can be created within a same application. This is less general than a complete dataflow architecture, since it cannot easily accomodate for elaborations involving some kind of feedback.

Räty [49] proposes in a 2007 paper the *Single Location Surveillance Point* (SLSP) architecture, which is a distributed architecture for video surveillance with a large number of distributed sensors. In this architecture the main focus is on avoiding the communication bottleneck at the decision making computers, by introducing an intermediate processing level (the so-called Session Servers) that concentrates and pre-elaborate the data obtained from the sensors.

In a 2007 paper [50], Amatriain presents a *metamodel* for a generic architecture devoted to multimedia processing applications. The proposed metamodel is based on a dataflow paradigm, and describes the conceptual structure of the processing nodes and the semantics of the operations defined on the nodes, but it does not specify a concrete implementation. In [51], Amatriain et al. discuss CLAM, a C++ library, based on the mentioned metamodel, for audio processing. This library provides an implementation in which multithreading is used to separate the core processing functions, which are executed in a high-priority thread, from the rest of the application that is executed in a normal thread, with safe inter-thread communication mechanisms to connect the two parts.

The paper by Detmold et al. [52], published in 2008, presents a network based middleware for Video Surveillance. In particular, the middleware is based on the *blackboard metaphor*, but it has a distributed blackboard to avoid a central server bottleneck. The components of the system communicate using web services, and in particular the REST (Representational State Transfer) approach. The use of web services entails a large communication overhead, and so the components of the system have necessarily to be coarse grained. Thus, this middleware can support parallelism among different video streams, but is limited in the parallelization of the activities relative to a single stream.

The paper by Saini et al. [53] presents an extensible system architecture for surveillance applications, which supports the ability to dynamically configure the processing steps that have to be applied to each input stream, and the event detectors that have to be enabled. This architecture is mainly concerned with flexibility and ease of configuration, and so does not address the distribution of the workload among different processors.

A recent paper by Valera et al. [54] presents a reference architecture for distributed video surveillance which is based on the *Real Time Networks* (RTN) and *Data-Oriented Requirements Implementation Scheme* (DORIS) design methodologies for concurrent systems, that are often used in aerospace and defense applications and in other safety-critical systems. The proposed architecture is implemented on top of the CORBA middleware, and so uses the CORBA mechanisms for distributed processing and communications. The main benefit of the proposed architecture is the fact that is based on formally defined design principles and patterns, thus improving the verifiability of the system. On the other hand, the methodology is quite complex; moreover, the CORBA platform, on which it is based, is known to be more difficult to use and less efficient than other middlewares, leading in the last years to a pronounced decline in its adoption.

More recently, instead than focusing on the whole architecture, some authors have focused on the optimization of some specific components, such as for instance the scheduler. In [55], the authors evaluate the impact of different scheduling policies on the execution of a dynamic dataflow program on a target architecture; such policies, in turns, depend on some conditions of the specific application at hand.

Chapter 3

Tracking Algorithm

3.1 Rationale of the method

As introduced in Section 2.1, the trajectories extraction module is in charge of extracting moving objects trajectories starting from the analysis of raw videos. A general overview is depicted in Figure 3.1: two main steps are required, namely the detection and the tracking.

The aim of the detection step is to obtain the list of *blobs*, being each blob a connected set of foreground pixels; in order to achieve this aim, the detection module first finds foreground pixels by comparing the frame with a background model; then foreground pixels are filtered to remove noise and other artifacts (e.g. shadows) and are finally partitioned into connected components, namely the *blobs*.

The tracking algorithm receives as input the set of blobs detected at each frame and produces a set of *objects*. An *object* is any real-world entity the system is interested in tracking. Each object has an *object model*, containing such information as

- the object class (*eg* a person or a vehicle) (Subsection 3.3.2),
- state (Subsection 3.3.1),
- size, position and predicted position, trajectory and appearance (Subsection 3.3.4).

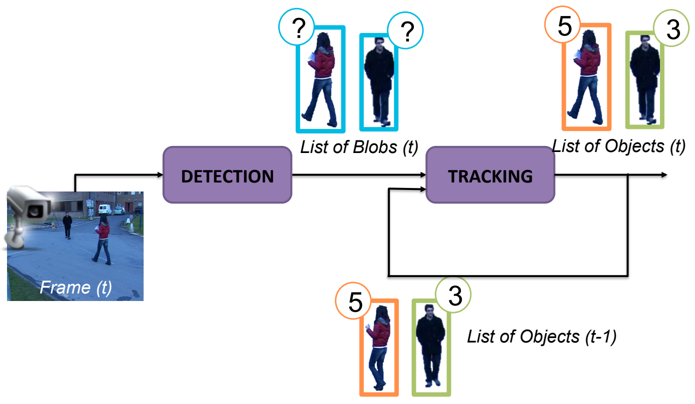


Figure 3.1 Architecture of a generic tracking system based on a background subtraction algorithm: the list of blobs is extracted by the detection module and is analyzed by the tracking algorithm in order to update the information associated to the list of objects populating the scene.

A *group object* corresponds to multiple real-world entities tracked together; if a group is formed during the tracking (*i.e.* it does not enter in the scene as a group), its object model maintains a reference to the models of the individual objects of the group.

The task of the tracking algorithm is to associate each blob to the right object, so as to preserve the identity of real-world objects across the video sequence; the algorithm must also create new object models or update the existing ones as necessary. As highlighted in Section 2.1 (see Figure 2.1), this problem is far for being simple because of occlusions, split or merge patterns that may happen because of the perspective flattening introduced by the use of a single camera or by errors of the detection step. The algorithm used in this thesis is able to efficiently deal with the follow mentioned problems.

3.2 Common problem

In this section we examine some typical problems of people detection, in order to see how they can be solved by incorporating information about the history of objects.

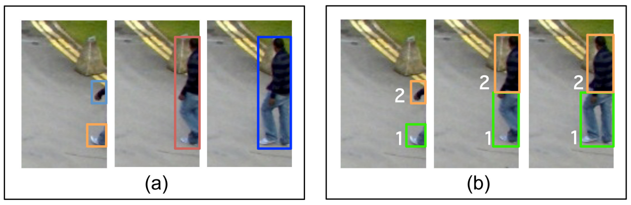


Figure 3.2 Problems with entering objects. (a) Blobs across three adjacent frames. (b) The tracking performed without considering the object history; in this case a tracking algorithm would not be able to distinguish this situation from two separate objects joining a group, and so attempts to preserve the separate identities of the two objects 1 and 2.

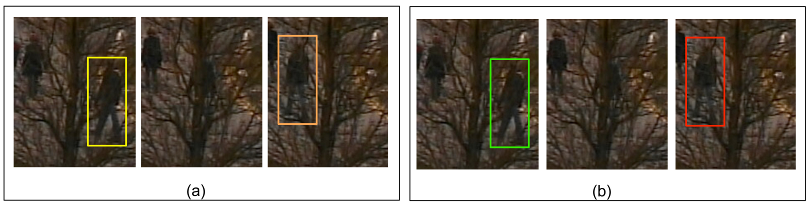


Figure 3.3 Problems with totally occluded objects. (a) The detection output across three adjacent frames. (b) The tracking performed without considering the object history; in this case a tracking algorithm would see the blob 2 as a new object in the scene, since in the previous frame there was no corresponding object.

One of the most frequently encountered issue is related to objects entering the scene, which have a very unpredictable appearance during the first frames of their life. Figure 3.2 shows a typical example, in which the person is split by the detection phase into two different blobs (*i.e.* legs and arms). The problem here is that after a few frames the parts appear to merge forming a new group of objects; since the occlusion resolution requires that object identities are preserved within a group, the tracking algorithm would continue to keep track of two separate objects (labeled 1 and 2 in the figure). To solve this problem, the tracking algorithm has to use different rules for dealing with merging objects when they are just entering or when they are stable within the scene.

Missing blobs are another typical problem affecting the detec-

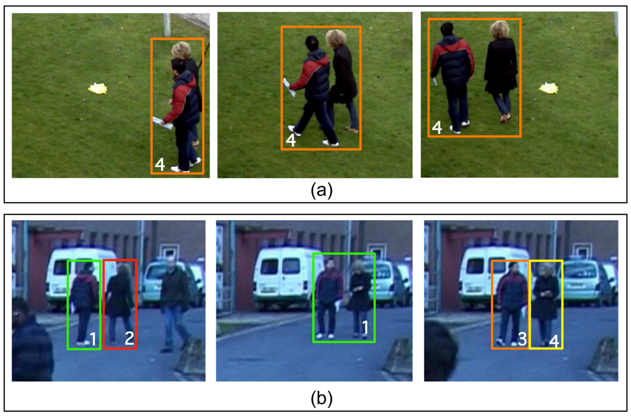


Figure 3.4 Occlusion related problems. In (a) the appearance of a group of people change from being identical to a single person to being clearly a group; if the classifier uses a classification result obtained in the first frame, it will continue to track the group considering it as an individual object. In (b) the system would not be able to correctly track the individual objects during all their life as a group if it attempts to exploit the uniformity of the motion, since the group can have strong changes in direction.

tion; they can be caused either by camouflage, occurring when the foreground object is very similar to the background, or when occlusions between moving objects arise. The latter case is really difficult to deal with as information about the occluded part is totally missing and consequently to be restored by suited reasoning.

Figure 3.3 shows an example of the above mentioned problem: the person that passes behind the tree is detected in the first and in the third frames of the sequence, but not in the second one. Thus, the tracking algorithm would find at the third frame a blob having no corresponding object in the previous frame, and would assign it to a newly created object, if it does not keep some memory about an object even when it is not visible in the scene. On the other hand, the tracking algorithm should not preserve information about objects that are truly leaving the scene: doing so it would risk to reassign the identity of an object that has left the scene to a different object that is entering from the same side.

Other issues are related to objects occluding each other, form-

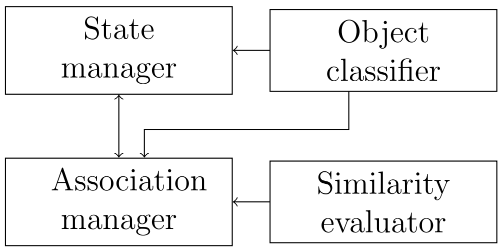


Figure 3.5 An overview of the proposed tracking system.

ing a group. Figure 3.4a illustrates a problem connected with the stability of group classification: in the first frame, the two persons in the group are perfectly aligned, and so a classifier would not be able to recognize that the object is a group. On the other hand, in the following frames the object is easier to recognize as such. Thus, in order to obtain a reliable classification the tracking algorithm has to wait that the classifier output becomes stable, before using it to take decisions.

Another problem related to groups is the loss of the identities of the constituent objects. An example is shown in Figure 3.4b, where objects 1 and 2 first join a group and then are separated again. When objects become separated, the tracking algorithm would incorrectly assign them new identities, if the original ones were not preserved and associated with the group. Note that in this case it would not have been possible to simply keep tracking separately the two objects using some kind of motion prediction until the end of the occlusion, because as a group the objects have performed a drastic change of trajectory (a 180° turn).

The analysis conducted in this Section about the typical problems in a real-world setting shows that in a lot of situations a tracking system cannot be able to correctly follow objects without additional information about their history.

3.3 Proposed Architecture

The main lack of most of existing algorithms lies in the fact that they make their tracking decisions by comparing the evidence at the current frame with the objects known at the previous one; all objects are managed in the same way, ignoring their past history that can give useful hints on how they should be tracked: for instance, for objects, stable in the scene, information such as their appearance should be considered more reliable.

To exploit this idea, the proposed algorithm adopts an object model based on a set of scenarios in order to deal differently with objects depending on their recent history and conditions; the scenarios are implemented by means of a Finite State Automaton (FSA), that describes the different states of an object and the conditions triggering the transition to a different state. The state is used both to influence which processing steps are performed on each object, and to choose the most appropriate value for some of the parameters involved in the processing.

Although a limited a priori knowledge about objects of interest is required, in order to differentiate between single objects and groups, the proposed method can be rapidly adapted to other application domains by providing a small number of object examples of the various classes.

Figure 3.5 gives an overview of the modules composing the tracking system and their interdependencies:

- the *state manager*, which maintains and updates an instance of the FSA for each object;

- the *association manager*, which establishes a correspondence between objects and blobs, solving split events and performing occlusion reasoning;

- the *object classifier*, which assigns objects to a set of predefined classes; the object class is used both during the update of the FSA state and during the association between objects and blobs to solve split/merge cases;

```
procedure Tracking(obj_models, blobs)
  Classify(blobs)
  S := ComputeSimilarity(obj_models, blobs)
  FindAssociations(obj_models, blobs, S)
  UpdateModels(obj_models, blobs)
  UpdateState(obj_models)
end procedure
```

Figure 3.6 The structure of the tracking algorithm.

- the *similarity evaluator*, which computes a similarity measure between objects and blobs, considering position, size and appearance; this similarity is used during the association between objects and blobs.

The above modules share a set of objects models, which, as previously said, contain all the relevant information about each object.

Figure 3.6 shows an outline of the tracking algorithm. The algorithm operates at the arrival of each new frame, receiving as inputs the existing object models and the blobs discovered by the detection phase for the current frame; for the first frame of the sequence, the existing object models are initialized as an empty set. The output of the algorithm is a set of updated object models, which possibly includes new objects. The steps of the algorithm are the following:

- first, the classifier is applied to the current blobs, which are annotated with the information on the assigned class;

- then, the algorithm computes the similarity between each object and each blob, activating the similarity evaluator; the similarity information is kept in a similarity matrix `S`;

- at this point, the algorithm is ready to perform the association between objects and blobs, including the split/merge and occlusion reasoning;

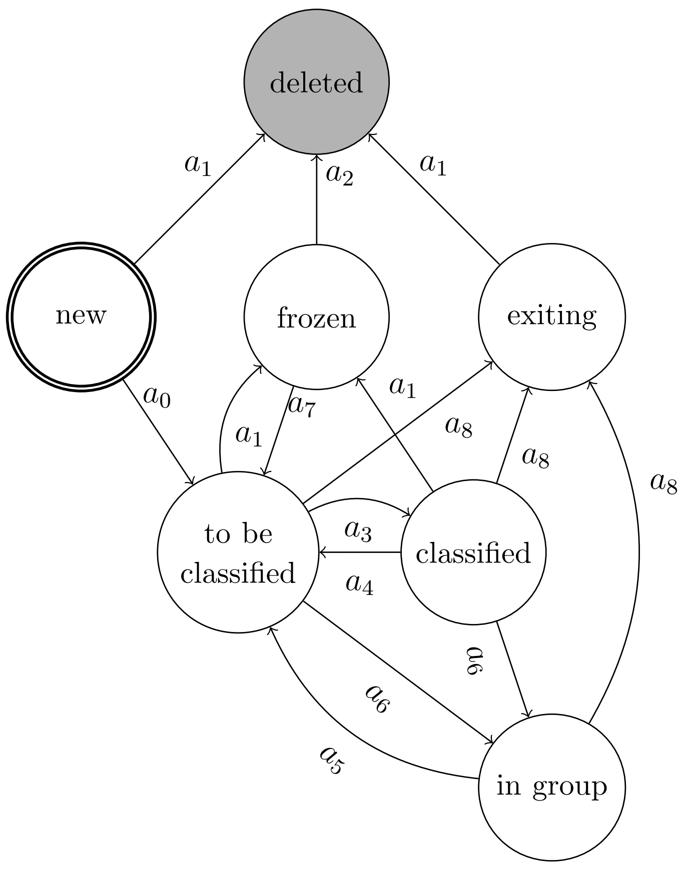


Figure 3.7 The state diagram of the object state manager.

- on the basis of the associations found, the algorithm updates the models for each object; if new objects are detected, their models are created at this step;
- finally, the state manager updates the FSA states, using the previous state and the information gathered by the previous steps and stored in the object models.

In the following sections, more details are provided for each module of the system.

3.3.1 Object state management

The state manager has the task of maintaining and updating the FSA state of each object; the FSA state embodies the relevant information about the past history of the object, which can be used by the other parts of the tracking system. What pieces of information are actually relevant depends somewhat on the specific application; different problems may require the algorithm to keep different information in order to deal appropriately with them, and so may require an entirely different FSA.

Although we present only a single formulation of the FSA, the methodology remains general and easily extendable to other cases, since the knowledge about the states and the transitions between them is declaratively specified in the automaton definition, and not hidden within procedural code.

In order to deal with the issues discussed in Section 3.2, we propose a state manager based on the Finite State Automaton $\mathcal{A}$ depicted in Figure 3.7. It can be formally defined as:

$$\mathcal{A} = \langle S, \Sigma, \delta, s_0, F \rangle \tag{3.1}$$

where $S = \{s_0, \ldots, s_m\}$ is the set of the states; $\Sigma = \{a_0, \ldots, a_m\}$ is the set of the transition conditions, *i.e.* the conditions that may determine a state change; $\delta : S \times \Sigma \rightarrow S$ is the state-transition function; $s_0 \in S$ is the initial state and $F \subset S$ is is the set of final states.

The proposed Finite State Automaton states and transitions are shown in Table 3.8. In particular, the set of states S is shown in Table 3.8.a; we choose s_0 as initial state, since each object enters the scene by appearing either at the edge or at a known entry region (*e.g.* a doorway). Furthermore we choose s_5 as final state, since each object necessarily has to leave the scene. The set Σ of transition conditions and the state-transition function δ are shown respectively in Table 3.8b and 3.8c.

It is worth noting that each state has been introduced in order to correctly solve one of the issues described earlier, as we will detail below. So it is possible to extend the FSA with the addition

Id	**Description**
s_0	new
s_1	to be classified
s_2	classified
s_3	frozen
s_4	in group
s_5	exiting
s_6	deleted

(a)

Id	**Description**
a_0	obj is completely within the scene
a_1	obj disappears from the scene
a_2	obj does not reappear in the scene for a time T_d
a_3	obj classification is the same for two frames
a_4	obj classification changes
a_5	obj leaves the group
a_6	obj occludes with one or more objects
a_7	obj reappears inside the scene
a_8	obj is not completely within the scene

(b)

	a_0	a_1	a_2	a_3	a_4	a_5	a_6	a_7	a_8
s_0	s_1	s_6	-	-	-	-	-	-	-
s_1	-	s_3	-	s_2	-	-	s_4	-	s_5
s_2	-	s_3	-	-	s_1	-	s_4	-	s_5
s_3	-	-	s_6	-	-	-	-	s_1	-
s_4	-	-	-	-	-	s_1	-	-	s_5
s_5	-	s_6	-	-	-	-	-	-	-

(c)

Figure 3.8 The Finite State Automaton. (a) The set S of the states. (b) The set Σ of the transition conditions. (c) The state-transition function δ; for entries shown as '-', the automaton remains in the current state.

of other states and transitions, in order to deal with some other problems that should arise in a specific application context.

The meaning of the states and the conditions triggering the transitions are detailed below:

- new (s_0): the object has been just created and is located at the borders of the frame; if it enters completely, and so does not touch the frame borders (a_0), it becomes *to be classified*; otherwise, if it leaves the scene (a_1), it immediately becomes *deleted*.

  The introduction of new state solves the problem related to the instability of the entering objects, since it makes the system aware of such scenario and then capable to react in the best possible way, as shown in Figure 3.9a. Moreover, this state allows the algorithm to quickly discard spurious objects due to detection artifacts, since they usually do not persist long enough to become *to be classified*.

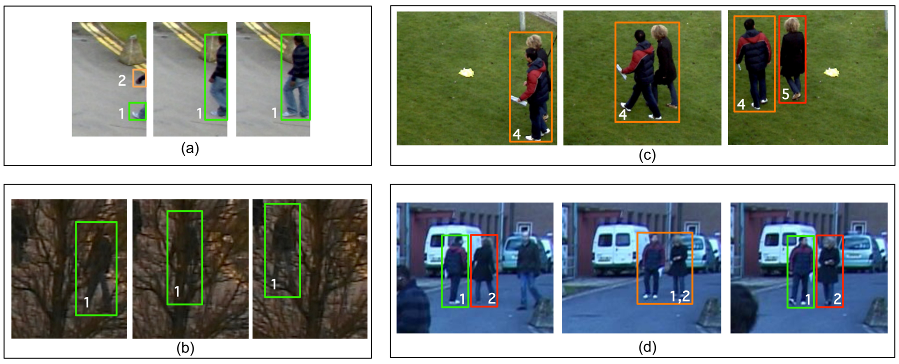


Figure 3.9 Output of the tracking algorithm based on the proposed FSA, when applied to the problems discussed in Section 3.2. (a) The entering object is correctly recognized as a single object, and not a group. (b) The object identity is preserved when the person passes behind the tree. (c) A group initially classified as a single person is correctly handled when classification becomes stable. (d) The group object maintains the constituent object identities.

- *to be classified* (s_1): the object is completely within the scene, but its class is not yet considered reliable; if the classifier assign the same class for at least two frames (a_3), it becomes *classified*; if the association manager detects that the object has joined a group (a_6), it becomes *in group*; if the object disappears (a_1), it becomes *frozen*; if the object is leaving the scene, *i.e.* it is not completely within it (a_8), it becomes *exiting*.

  The *to be classified* state solves the issues of the objects entering the scene as group, discussed in Figure 3.4a. Thanks to this state, the class of the object is only validated when the system is sure about them. An example is shown in Figure 3.9c. Note that objects class is very important, since it makes the association manager able to take the correct decisions about the resolution of split and merge patterns.

- *classified* (s_2): the object is stable and reliably classified; if the classifier assigns a different class (a_4), it becomes *to be classified*; if the association manager detects that the object has joined a group (a_6), it becomes *in group*; if the object disappears (a_1), it becomes *frozen*; if the object is leaving the scene, then it is not completely within it (a_8), it becomes *exiting*.

The distinction between *classified* and *to be classified* objects is used by the association manager when reasoning about split objects and group formation.

- *frozen* (s_3): the object is not visible, either because it is completely occluded by a background element, or because it has left the scene; if the object gets visible again (a_7), it becomes *to be classified*; if the object remains suspended for more than a time threshold T_d (a_2), it becomes *deleted*; currently we use $T_d = 1$ sec.

  The *frozen* state avoids that an object is forgotten too soon when it momentarily disappears, as it happens in Figure 3.3.

- *in group* (s_4): the object is part of a group, and is no more tracked individually; its object model is preserved to be used when the object will leave the group; if the association manager detects that the object has left the group (a_5), it becomes *to be classified*; if the object is located at the borders of the frame (a_8), it becomes *exiting*.

  The *in group* state has the purpose of keeping the object model even when the object cannot be tracked individually, as long as the algorithm knows it is included in a group. Thanks to this state, the proposed method is able to correctly solve the situation shown in Figure 3.4b, related to group objects.

- *exiting* (s_5): the object is located at the borders of the frame; if it disappears from the scene (a_1), it becomes *deleted*.

  The *exiting* objects differ from the *frozen* ones because of the system have not to preserve their identity, since their are leaving the scene;

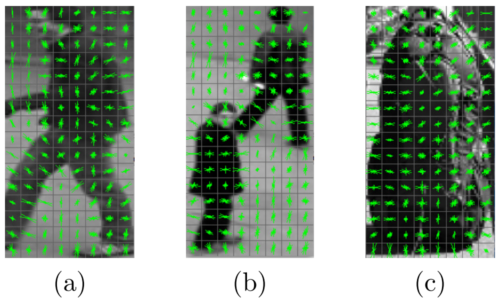


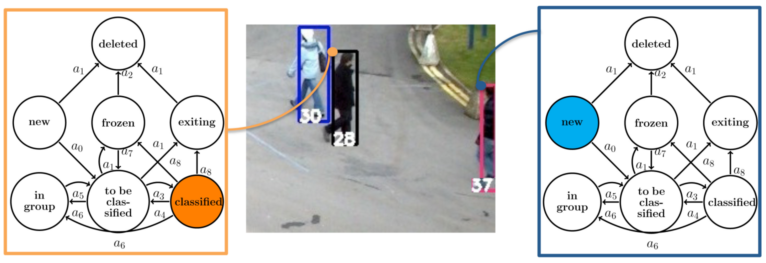
Figure 3.12 Examples of different entities by HOG descriptors: (a) a single person, (b) a small group of persons and (c) a backpack.

- *deleted* (s_6): the object is not being tracked anymore; its object model can be discarded.

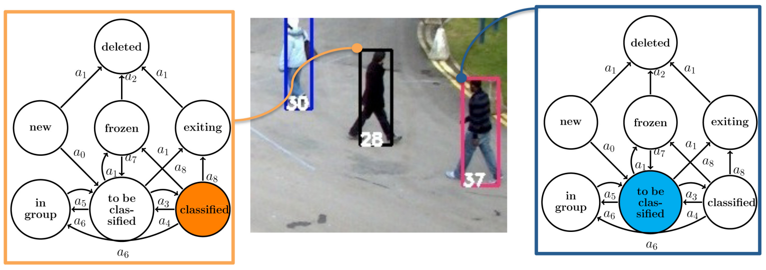
Figures 3.10 and 3.11 show a detailed example of how the object state management works.

3.3.2 Object classification

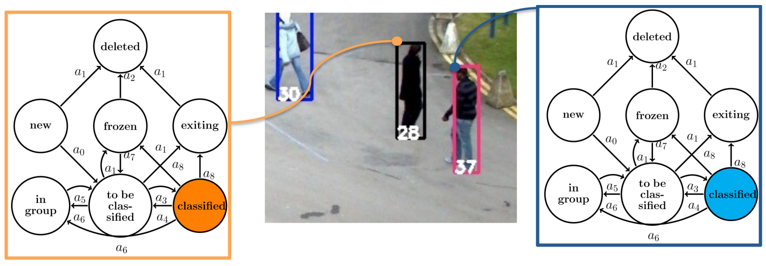
The tracking system needs an object classifier to determine if a blob corresponds to a group, an individual object, or an object part. In particular, two classes of individual objects have been considered in this thesis: person and baggage. We adopt a multi-class Support Vector Machine (SVM) classifier using the Histogram of Oriented Gradients (HOG) [56] as descriptor. HOG descriptor, which has already proved to be very effective for pedestrian detection, allows to describe the patterns by using the distribution of local intensity gradients or edge directions: the image is partitioned into *cells* and a local 1-D histogram of gradient directions or edge orientations over the pixels of each cell is computed. The accuracy of the descriptor is improved by contrast-normalizing the local histograms: a measure of the intensity across a larger region of the image, a *block*, is computed and it is used to normalize all the cells within the block. Such normalization makes the descriptor invariant in changes in illumination and shadowing. In Figure 3.12 we show the description of different entities using HOG.



1. The object 37 enters the scene as a *new* object while the object 28 is a *classified* object.

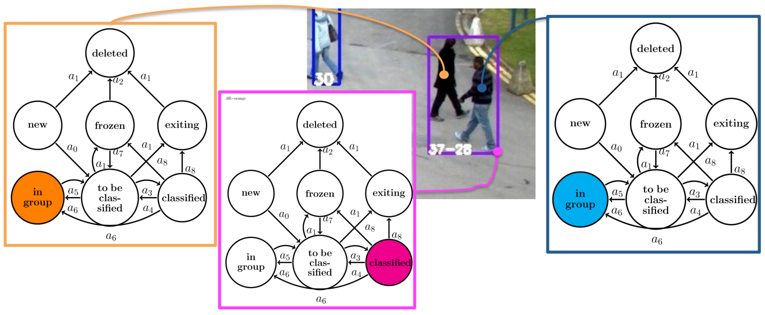


2. The object 37 becomes a *to be classified* object since it completely enters the scene.

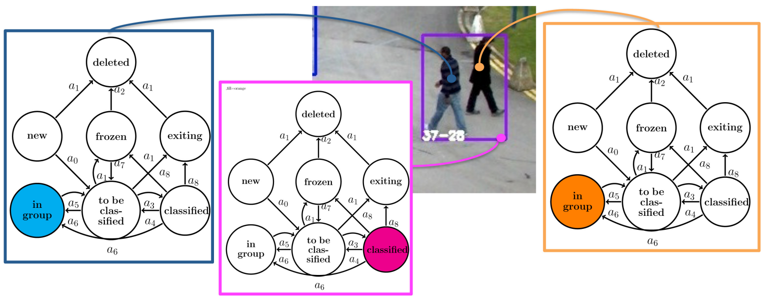


3. The classification as a person of the object 37 becomes reliable, then it becomes *classified*.

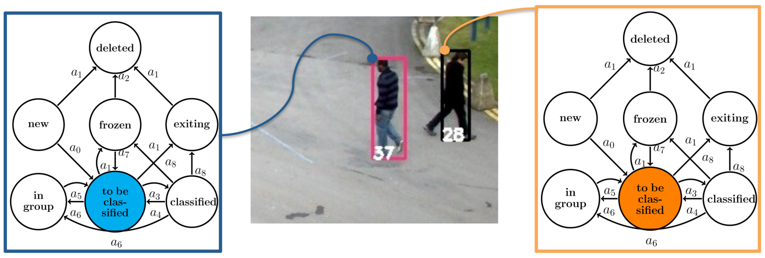
Figure 3.10 Evolution of the states of the FSA of each object along a short frame sequence (part 1). The second part is in Figure 3.11.



4. After a few frames, the association manager shows an occlusion between objects 37 and 28. Both these objects become *in group* and a new group object is created.



5. The objects 37 and 28 do not leave the group, then they do not change their states.



6. Finally, the group object splits and both the objects 37 and 28 become *to be classified*.

Figure 3.11 Evolution of the states of the FSA of each object along a short frame sequence (part 2). The first part is in Figure 3.10.

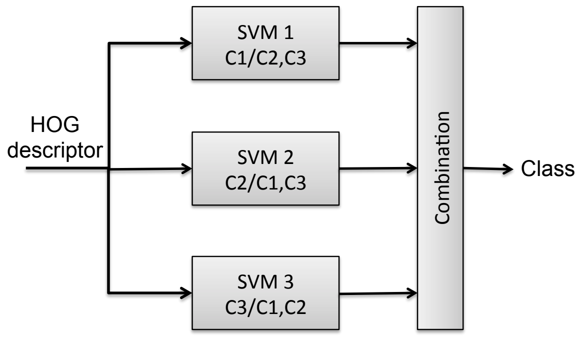


Figure 3.13 Multi class SVM: the i-th classifier is labeled considered as positive samples the ones belonging to the i-th class, while as negative samples the ones belonging to the other classes.

Once extracted the features vector, a multi-class SVM has been applied. Being our problem multi-class, a *N one-against-rest* strategy has been considered (see Figure 3.13);

in particular N different classifiers, one for each class, are constructed. The i-th classifier is trained on the whole training data set in order to classify the members of i-th class against the rest. It means that the training set is relabeled: the samples belonging to the i-th class are labeled as positive examples, while samples belonging to other classes are labeled as negative ones. During the operating phase, a new object is assigned to the class with the maximum distance from the margin.

At this point one can note that object evolution is not dependent on its class (*e.g.* group or individual object), but only on its actual state. As a matter of fact, only object information is related to object class, while object state only determines the reliability of such information. In particular, for individual object we have information about appearance and shape: we consider the area and the perimeter of an object, its color histograms and its real dimensions, namely width and height, both obtained using an Inverse Perspective Mapping. Moreover we have information about the observed and predicted position of the object centroid. The predicted position is obtained using an extended 2D Position-Velocity

(PV) Kalman Filter, whose state vector is:

$$\xi = \left[x_c, y_c, w, h, \dot{x}_c, \dot{y}_c, \dot{w}, \dot{h},\right] \tag{3.2}$$

where (x_c, y_c) is the centroid of the object, w and h are the width and the height of the object minimum bounding box in pixels, $(\dot{x}_c, \dot{y}_c)$ and $(\dot{w}, \dot{h})$ are respectively the velocity of the object centroid and the derivative of the minimum bounding box size. It is worth noting that such a PV Kalman Filter is very effective when the object motion is linear and the noise has a Gaussian distribution.

Group objects contain also information about occluded objects. In this way the system can continue to track the *in group* objects when they leave the group.

3.3.3 Association management

The aim of this module is to determine the correspondence between the set of blobs $B = \{b_1, ..., b_n\}$ and the set of objects $O = \{o_1, ..., o_m\}$, and properly update the information about the annotated objects: the current position is added to the trajectory, the appearance model updated and the state properly recomputed; this is made also in presence of detection errors: objects split into more parts, objects merged together or objects hidden by foreground elements.

To this aim, the system uses a graph based approach by taking into account the spatio-temporal information of each object and at the same time reduces the computational cost needed to perform all the possible association: this is done by taking into account spatio-temporal information of each object.

The associations between blobs and objects is represented by a matrix $T = \{t_{ij}\}$, where:

$$t_{ij} = \begin{cases} 0 & \text{if object } o_i \text{ is not associated to blob } b_j \\ 1 & \text{if } o_i \text{ is associated to } b_j \\ -1 & \text{if the pair } o_i, b_j \text{ hasn't been evaluated} \end{cases} \tag{3.3}$$

It can be obtained by evaluating the similarity matrix $S = \{s_{ij}\}$, being s_{ij} the index of similarity between the blob b_i and the object o_j. More details about the computation of this matrix will be provided in Section 3.3.4.

In simple situations, there is a one-to-one correspondence: the single blob b_i is associated to the single object o_i, as shown in Figure 3.18a. However, in presence of split or merge, the association manager needs to take into account more complex associations (one-to-many, many-to-one, and even many-to-many).

To this purpose, we evaluate the similarity matrix over an augmented set of blobs B^I and objects O^I in order to take into account all those above mentioned situations:

$$B^I = B \cup B_d; \quad O^I = O \cup O_d. \tag{3.4}$$

B_d and O_d are respectively the set of derived blobs and derived objects, virtually created at the current frame; their introduction allows the system to simulate all the possible splits and occlusions occurring in real scenarios, so as to take the best possible decision in terms of association between one or more blobs and one or more objects.

In order to clarify this, consider two objects o_1 and o_2 meeting in the scene: at the frame $t-1$ the algorithm correctly tracks these objects one by one. At the frame t, a merge occurs and the detection phase detects a single blob b_A (instead of two). The similarity s_{A1} between the blob b_A and the object o_1 and the similarity s_{A2} between the blob b_A and the object o_2 are very low, and then a simple association manager could fail. Thanks to the introduction of the derived object $o_{1\cup 2}$, composed by the union of the objects o_1 and o_2, a merge situation is simulated. In this way, the similarity between the blob b_A and the derived object $o_{1\cup 2}$ is very high, and then the association can be correctly performed. The states of o_1 and o_2 are updated to <u>*in group*</u> and their information are stored inside the *group* object $o_{1\cup 2}$, whose state is initialized to <u>*classified*</u>. In this way, the system is able to correctly track the two objects as a single group object.

Derived Blobs and Objects Creator This modules generates the sets B_d and O_d of derived blobs and objects. The most simple, but also the most inefficient way, to perform this task is to evaluate all the possible combinations of k blobs (objects), with $k = 1...n(m)$.

$$B_d = \{b_1b_2, ..., b_1b_n, ..., b_1b_2b_n, ...\}; \tag{3.5}$$

$$O_d = \{o_1o_2, ..., o_1o_m, ..., o_1o_2o_m, ...\}. \tag{3.6}$$

The inclusion of all the combinations determines a very high computational cost and an explosion of the size of the similarity matrix; note that the number of possible combinations C for n blobs and m objects is:

$$C = \sum_{i=1}^{n} \binom{n}{i} \cdot \sum_{j=1}^{m} \binom{m}{j} . \tag{3.7}$$

Consider that if the scene is populated by only 10 objects and the detection phase finds 10 blobs, we need to verify more than 1000 possible associations (1274). In order to decrease the number of associations to be evaluated, we propose to exploit the spatio-temporal continuity of the tracked objects so as to select only the subset of feasible combinations.

In particular, the following heuristics have been considered to obtain the sets B_d and O_d:

- The distance between the centroids of the blobs (objects respectively) composing a derived blob (object) has to be less than an adaptive threshold τ. τ is dynamically chosen and depends on the maximum velocity of objects representing the maximum displacement (in pixels) of an object between two frames. This value is strongly related to the position of the object inside a scene: once fixed the maximum real velocity of an object inside the scene, the maximum distance d_{max} in pixel can be computed by means of the IPM algorithm: $\tau = \alpha \cdot d_{max}$, being α a weight set in our experiments to 2, which determines the area where find out possible split

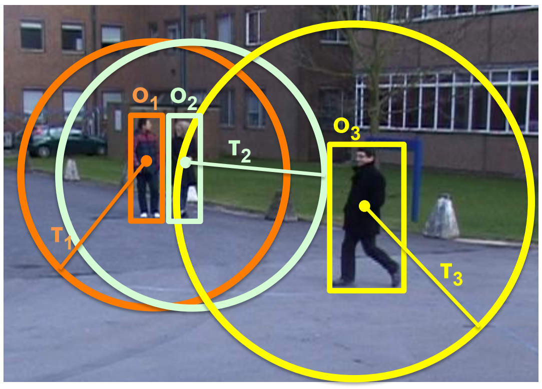


Figure 3.14 Heuristic for pruning the derived blobs.

and merge of blobs. An example is depicted in Figure 3.15: the only derived objects that the system can evaluate are o_1o_2 and o_2o_3, so discarding the pair o_1o_3, whose distance between the centroids is over the thresholds τ_1 and τ_3.

- The reciprocal position of the blobs (objects) is taken into account. For instance, starting from the situation depicted in Figure 3.15a, only the combination $b_1b_2b_3$ (Figure 3.15b), b_1b_2 (Figure 3.15c) and b_2b_3 (Figure 3.15d) can be accepted, while the association b_1b_3 (Figure 3.15e) doesn't make sense, since it is an impossible merge, implicitly including b_2. Note that in this very simple situation, often recurring also in real scenarios, we are able to obtain a 25% decrease in the number of association to deal with; furthermore, it does not affect the reliability of the system, since it only discards unfeasible associations.

The main steps of the algorithm for obtaining the set of derived blobs B_d are shown in Figure 3.16. A similar algorithm applies for derived objects O_d.

The algorithm starts by computing the distance between the boxes belonging to B, whose relative centroids distance is less than τ, so obtaining the weighted undirected graph $G = \{V, E, w\}$; the vertices $V = \{V_1, V_2, ..., V_n\}$ are associated to blobs, while the

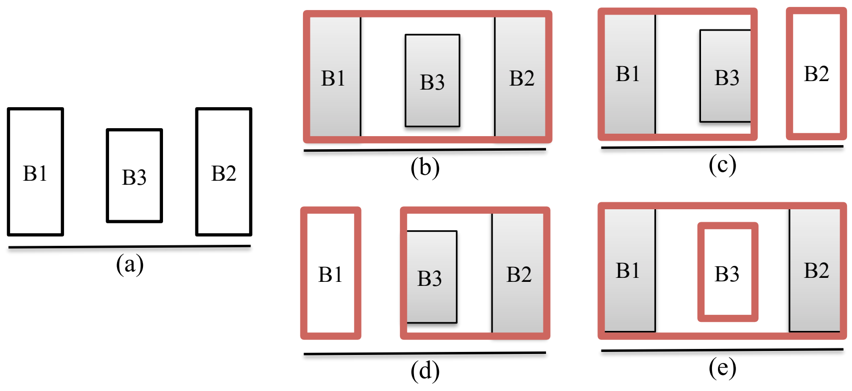


Figure 3.15 Examples of feasible (b,c,d) and unfeasible (e) merge of the boxes in (a) for the composition of derived boxes.

edges $E = \{E_1, E_2, ..., E_k\}$ to the distances between the blobs. Each edge is associated to a weight $e_{i,j}$ corresponding to the distance between the blob b_i and the blob b_j. Note that if the distance between two blobs is under τ, then the corresponding edge does not belong to E.

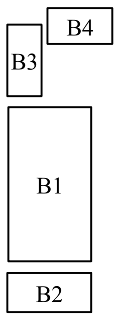
For each blob b_i, the shortest path to reach all the other blobs belonging to B and not yet explored is computed by using the Dijkstra's algorithm; it is implemented by a min priority queue with a Fibonacci heap [57], so that the global computational complexity of the algorithm is $O(|E| + |V| \cdot \log |V|)$.

For each found shortest path, a new derived blob b_d is created, composed by the blobs corresponding to the vertices crossed during the path. An example is shown in Figure 3.17: in this case, the method reduces by 40% the number of derived objects (the combinations b_2b_3, b_2b_4, b_1b_4 and $b_2b_3b_4$ are not generated).

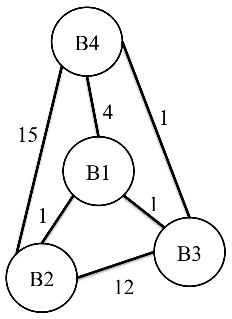
The algorithm The algorithm operates in two distinct phases, as shown in Figure 3.19: in the first one, it finds the correspondence for stable objects (*i.e.* objects in the *to be classified*, *classified* or *frozen* state); in the second phase it tries to assign the remaining blobs to objects in the *new* state, possibly creating such objects if necessary. The motivation for this distinction is that the

```
procedure $B_d$ = FindDerivedFeasibleBlobs(Blobs $B$)
 $B_d$={ }
 $d$ := ComputeDistancesBetweenCentroids($B$)
 $graph$ := ComputeGraph($B$,$d$)
 foreach $b_i$ in $B$
    foreach $b_j$ in $B$, $j > i$
        $path$ := ComputeShortestPath($graph$, $b_i$, $b_j$)
        $b_d$ := CreateBlob($graph$, $path$)
        $B_d$:= AddBlob($graph$,$b_d$)
end procedure
```

Figure 3.16 The algorithm for finding out the derived feasible blobs (objects).



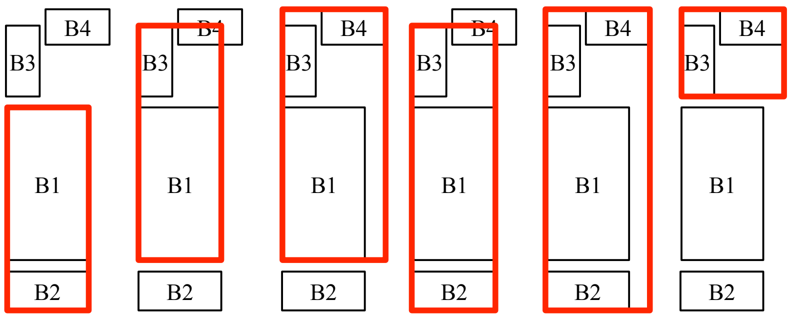
(a)



(b)

FROM	TO	PATH
B1	B2	B1 B2
B1	B3	B1 B3
B1	B4	B1 B3 B4
B2	B3	B2 B1 B3
B2	B4	B2 B1 B3 B4
B3	B4	B3 B4

(c)



(d)

Figure 3.17 An example of the algorithm in charge of creating derived blobs (objects) for a split pattern: starting from the detected blobs in (a), the algorithm creates the graph in (b) and finds the shortest paths in (c). The output of the algorithm is in (d).

management of split-merge and occlusions can be performed only for stable objects, since for new ones the system would not have enough information to do it in a reliable way.

During the first phase, the algorithm starts by choosing the maximum element s^*_{ij} of the matrix, corresponding to the blob b_i and to the object o_j and records the corresponding associations in T. At each step, the algorithm selects the pair b_i–o_j such that s_{ij} is the maximum value corresponding to $t_{ij} = -1$ and records the obtained associations.

The following situations can occur:

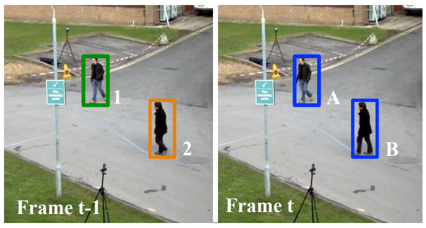
one-to-one correspondence: $b_i \in B$, $o_j \in O$ (see Figure 3.18a). The associations corresponding to the blob b_i and to the object o_j need to be updated, together with all the derived blobs and objects containing b_i and o_j; in particular:

$$\begin{aligned} t_{i,j} &= 1; \\ t_{z,j} &= 0 \quad \forall\, b_z \in B^I,\ b_z \neq b_i; \\ t_{z,k} &= 0 \quad \forall\, b_z \supset b_i,\ \forall\, o_k \in O^I; \\ t_{i,k} &= 0 \quad \forall\, o_k \in O^I,\ o_k \neq o_j; \\ t_{z,k} &= 0 \quad \forall\, o_k \supset o_i,\ \forall\, b_z \in B^I; \end{aligned} \tag{3.8}$$

one-to-many correspondence: $b_i \in B$, $o_j \in O_d$ (see Figure 3.18b). In this case the occlusion pattern has to be solved. The associations corresponding to the blob b_i and to the object o_j need to be updated, together with all the associations of the objects composing the derived object o_j:

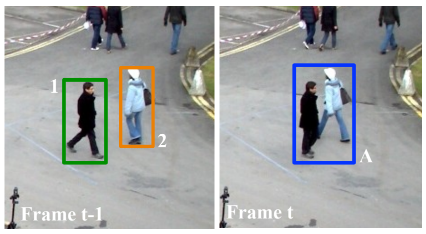
$$\begin{aligned} t_{i,j} &= 1; \\ t_{z,j} &= 0 \quad \forall\, b_z \in B^I,\ b_z \neq b_i; \\ t_{z,k} &= 0 \quad \forall\, b_z \supset b_i,\ \forall\, o_k \in O^I; \\ t_{i,k} &= 0 \quad \forall\, o_k \in O^I,\ o_k \neq o_j; \\ t_{z,k} &= 0 \quad \forall\, o_k \subset o_j,\ \forall\, b_z \in B^I; \end{aligned} \tag{3.9}$$

Furthermore, the state of o_j is initialized to *classified* while the



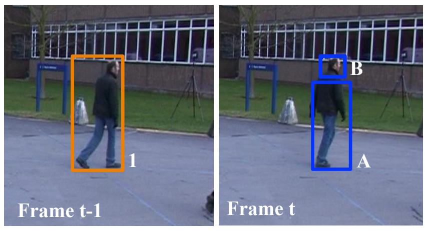
	Obj 1	Obj 2	Obj 1 U 2
Blob A	0,96	0	0,32
Blob B	0	0,93	0,26
Blobs A U B	0,25	0,31	0,63

(a)



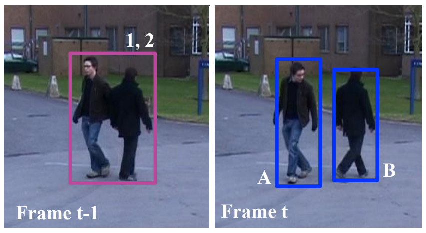
	Obj 1	Obj 2	Obj 1 U 2
Blob A	0,48	0,53	0,86

(b)



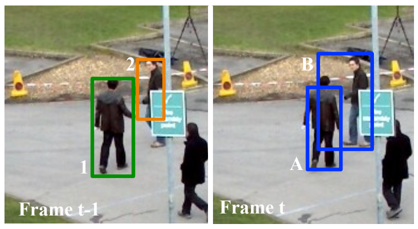
	Obj 1
Blob A	0,65
Blob B	0,25
Blobs A U B	0,89

(c)



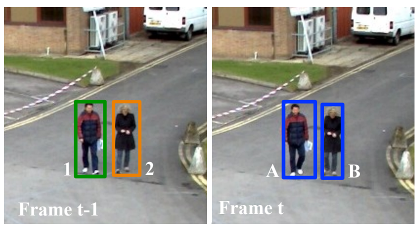
	Obj 1,2
Blob A	0,38
Blob B	0,45
Blobs A+B	0,89

(d)



	Obj 1	Obj 2	Obj 1 U 2
Blob A	0,59	0,20	0,43
Blob B	0,62	0,53	0,71
Blobs A U B	0,43	0,27	0,88

(e)



	Obj 1	Obj 2	Obj 1 U 2
Blob A	0,90	0,15	0,23
Blob B	0,62	0,89	0,18
Blobs A U B	0,25	0,27	0,91

(f)

Figure 3.18 Different kinds of associations: one-to-one association (a), one-to-many association (b), many-to-one associations (c,d), many-to-many associations (e,f).

state of the objects composing o_j is updated to *in group*; finally, an instance of each object is stored inside the derived object, which is classified as a *group* object. Starting from this frame, the object o_j will be tracked as a group until it will split in the next frames.

many-to-one correspondence: $b_i \in B_d$, $o_j \in O$. In this case a split problem has to be solved. Two different situations have to be taken into account; if o_j is classified as a person (see Figure 3.18c), then the associations corresponding to the blob b_i and to the object o_j need to be updated, together with all the associations of the blobs composing the derived blob b_i:

$$\begin{aligned}
t_{i,j} &= 1; \\
t_{z,j} &= 0 \quad \forall\, b_z \in B^I,\ b_z \neq b_i; \\
t_{z,k} &= 0 \quad \forall\, b_z \subset b_i,\ \forall\, o_k \in O^I; \\
t_{i,k} &= 0 \quad \forall\, o_k \in O^I,\ o_k \neq o_j; \\
t_{z,k} &= 0 \quad \forall\, o_k \supset o_i,\ \forall\, b_z \in B^I;
\end{aligned} \tag{3.10}$$

A different situation occurs when the object o_j is classified as a group (see Figure 3.18d), since the split refers to the end of an occlusion pattern. The system, by exploiting the set of occluded objects (1 and 2 in Figure 3.18d) and their history, does not solve the split but it is able to correctly associate each blob to the most similar object. A similarity matrix based approach is exploited in this case in order to find the best association between occluded objects and blobs.

many-to-many correspondence: $b_i \in B_d$, $o_j \in O_d$ (see Figure 3.18e). This situation arises when a merge and a split contemporaneously happen. In this case we decide to create a group object, so to avoid any kind of decision which could be reveal too risky. In this scenario, the associations corresponding to the blob b_i and to the object o_j need to be updated, together with all the associations of the blobs composing the derived blob

b_i and the objects composing the derived object o_j:

$$\begin{aligned}
&t_{i,j} = 1; \\
&t_{z,j} = 0 \quad \forall\, b_z \in B^I,\ b_z \neq b_i; \\
&t_{z,k} = 0 \quad \forall\, b_z \subset b_i,\ \forall\, o_k \in O^I; \\
&t_{i,k} = 0 \quad \forall\, o_k \in O^I,\ o_k \neq o_j; \\
&t_{z,k} = 0 \quad \forall\, o_k \subset o_i,\ \forall\, b_z \in B^I;
\end{aligned} \tag{3.11}$$

Furthermore, as in the *one-to-many correspondence*, the state of o_j is initialized to *classified* while the state of the objects composing o_j is updated to *in group* and their information are stored inside the derived object, which is classified as a *group* object.

A many-to-many correspondence can also arise when two or more objects are very near each other: as a matter of fact, the similarity between the derived object and the derived blob can be only slightly higher than the similarity between the single objects and the single blobs (see Figure 3.18f). If it happens, each object is associated to the blob with the higher similarity.

During the second phase, only the one-to-one association is performed. For this reason, the algorithm follows a similar scheme, except that it considers only the objects in the *new* state, and does not create derived blobs and derived objects. Moreover, the similarity matrix is built using less features than in the first phase since we have experimentally verified that only the position information is sufficiently reliable for such objects. At the end of this phase, any remaining unassigned blobs are used to create new annotated object, initialized to the *new* state.

3.3.4 Similarity evaluation

As already mentioned, the similarity matrix is used to match one or more blobs with one or more objects.

In order to measure the similarity between an object o_i and a blob b_j, the tracking system uses an index based on three kinds of

```
procedure TrackingAlgorithm (annotated_objs, blobs)
 AssocStableObjs (annotated_objs, blobs)
 pending_blobs := SearchPendingBlobs(blobs)
 unassoc_objs := SearchUnassocObjs(annotated_objs)
 AssocInstableObjs (unassoc_objs, pending_blobs)
 unassoc_blobs := SearchPendingBlobs(pending_blobs)
 CreateObjFromPendingBoxes (annotated_objs,unassoc_blobs)
 UpdateObjectsState(annotated_objs)
end procedure

procedure AssocInstableObjs (annotated_objs, blobs)
 sim_mat := ObjInstableSimMatrix (annotated_objs, blobs)
 foreach obj in annotated_objs:
         (best_boxes, best_objs) := BestAssoc(sim_mat)
 end
end procedure

procedure AssocStableObjs (annotated_objs, blobs)
 derived_objs:=CreateDerivedFeasibleObjs(annotated_objs)
 derived_blobs := CreateDerivedFeasibleBlobs(blobs)
 all_objs := JoinObjs(annotated_objs, derived_objs)
 all_blobs := JoinBlobs(blobs, derived_blobs)
 sim_mat := ObjStableSimMatrix (all_objs, all_blobs)
 foreach obj in all_objs:
        (best_boxes, best_objs) := BestAssoc(sim_mat)
 end
end procedure
```

Figure 3.19 Structure of the algorithm for stable and unstable objects associations.

information: the position, the shape and the appearance:

$$s_{ij} = \sqrt{\frac{\alpha_p \cdot (s_{ij}^p)^2 + \alpha_s \cdot (s_{ij}^s)^2 + \alpha_a \cdot (s_{ij}^a)^2}{\alpha_p + \alpha_s + \alpha_a}} \tag{3.12}$$

As described below, s_{ij} values identify similarity metrics and α values are weights chosen according to the state of the object and the association management phase. In particular:

- s_{ij}^p is the position similarity index, computed as the distance between the estimated centroid of an object o_i and the centroid of a blob b_j;
- s_{ij}^s is the shape similarity index between an object o_i and a

blob b_j;

- s^a_{ij} is the appearance similarity index between an object o_i and a blob b_j, based on color histograms;
- α_p, α_s and α_a are the weights of position, shape and appearance similarity index respectively;

All α values have been chosen by experimentation over a training set. Namely, in the first phase, selected values for objects in the *to be classified* and *classified* state are $\alpha_p = \alpha_s = \alpha_a = 1$ while for objects in the *in group* state selected values are $\alpha_s = \alpha_a = 1$; $\alpha_p = 0$ since in this context shape and appearance similarity perform better than position one. Finally, in the second phase that evaluates *new* objects, we choose to consider the only reliable feature, namely the position. Thus selected α values are $\alpha_s = \alpha_a = 0$; $\alpha_p = 1$.

For the position, as already seen, the system uses a Kalman filter, based on a uniform velocity model, to predict the coordinates of the object centroid at the current frame. The predicted coordinates are compared with the blob centroid, using Euclidean distance, in order to obtain for each object o_i and each blob b_j the distance d_{ij}. The position similarity index is then computed as:

$$s^p_{ij} = 1 - d_{ij}/d_{\max} \tag{3.13}$$

where $d_{\max}$ is a normalization factor depending on the maximum velocity of objects representing the maximum displacement of an object between two frames.

For characterizing the shape similarity, the system uses the real height and the area of the blob and of the object model; in particular if we denote as Δh_{ij} the relative height difference between o_i and b_j, and as ΔA_{ij} the relative area difference, the considered shape similarity index is:

$$s^s_{ij} = 1 - \sqrt{\frac{|\Delta A_{ij}| + (\Delta h_{ij})^2}{2}} \tag{3.14}$$

Finally, as a representation of the appearance we have used the color histograms computed separately for the upper half and for the lower half of the object or blob (*Image Partitioning*). We have experimented with several criteria for comparing the histograms, and we have found that the most effective value is the χ^2 distance:

$$q_{ij} = \frac{1}{M} \sum_k \frac{\left(h_i^o(k) - h_j^b(k)\right)^2}{h_i^o(k) + h_j^b(k)} \tag{3.15}$$

where index k iterates over the bins of the histogram, h_i^o is the histogram of object o_i, h_j^b is the histogram of blob b_j, and M is the number of bins. The appearance similarity index is:

$$s_{ij}^a = 1 - \sqrt{\frac{\left(q_{ij}^{up}\right)^2 + \left(q_{ij}^{low}\right)^2}{2}}. \tag{3.16}$$

where q_{ij}^{up} is the value of q_{ij} computed using only the upper half of the object/blob, and q_{ij}^{low} is the value computed using only the lower half.

In Section 6.1 we will show the results obtained by the proposed people tracking algorithms.

Chapter 4

Fire Detection Algorithm

4.1 Rationale of the method

Up to now the efforts of the research community have been mainly devoted to the definition of a representation of both color and movement, so as to discriminate fire from non fire objects; this inevitably leads to high dimensional feature vectors.

How to manage high dimensional feature vectors is a well-known problem in the communities of machine learning and pattern recognition: in fact, as shown in [58], employing a high dimensional feature vector would imply a significant increase in the amount of data required to train any classifier in order to avoid overspecialization and to achieve good results.

Furthermore, independently of the particular features extracted, in most of the above mentioned methods the high variability of fires, as well as the large amount of noise in data acquired in fire environments, prevent the systems from the achievement of a high recognition rate. More generally, it has been shown [59] that increasing the performance of a system based on the traditional combination feature vector – classifier is often a very expensive operation: in fact, it may require to design a new set of features to represent the objects, to train again the classifier, or to select a different classifier if the performance are not sufficiently satisfactory. Furthermore, this effort could be payed back by only a slight improvement in the overall accuracy, so this approach may prove not very convenient.

In order to overcome the above mentioned limitations, one of the solutions coming from the literature [59] is to split the feature vector and consequently to adopt a set of classifiers, each tailored on a feature set and then trained to be an expert in a part of the feature space. The main idea of this kind of paradigm, usually referred to as Multi Expert System (MES), is to make the decision by combining the opinions of the different individual classifiers (hereinafter *experts*), so as to consistently outperform the single best classifier [60]. This latter paper explains on the basis of a theoretical framework why a MES can be expected to outperform a single, monolithic classifier. In fact, most classifiers, given an unlimited amount of training data, converge to an optimal classification decision (in a probabilistic sense); but on a finite training set, their output is affected by an error (additional with respect to the inherent error due to ambiguities in the input data), which is either due to overspecialization or to the choice of reducing the classifier complexity in order to avoid the loss of generalization ability. The author of [60] shows that, under some assumptions satisfied very often in practical cases, a suitably chosen *benevolent* combining function can make the overall output of the MES less sensitive to the errors of the individual classifiers.

MESs have been successfully applied in several application domains, ranging from biomedical images analysis [61][62] and face detection [63] to movie segmentation [64] and handwriting recognition [59]. In this algorithm we propose the employing of a MES for detecting the fire in both indoor and outdoor environments: three different experts, complementary in their nature and regarding their errors, are combined with relatively little effort so as to make possible the improvement of the overall performance. It is evident that the successful implementation of a MES requires both the adoption of complementary sets of features feeding the different experts and the choice of a reliable combination rule.

Regarding the first aspect, we considered three different experts able to analyze the same problem from different points of view, based on color, on movement and on shape variation respectively. The main advantage deriving from this choice lies in the fact that the experts are very simple to configure, so making the proposed system particularly suited for deployment in real environments.

As for the experts based on color and shape variation, two algorithms widely adopted by the scientific community and providing very promising results have been considered; they are based on a thresholding in the YUV space and on the variation of the shape in terms of minimum bounding box enclosing the detected moving object, respectively. In particular, the expert based on color aims to discriminate *red* from *non red* objects and is particularly suited for sterile environments, while the one based on shape variation is very effective for distinguishing fire, usually having a strongly variable shape, from rigid objects moving in the scene, such as vehicles.

Finally, the expert based on movement evaluation is based on the assumption that fire has a disordered movement, much more disordered if compared with any other object usually populating the scene. In order to exploit this property, a novel descriptor for representing the movement is proposed for this algorithm: the main idea is to adopt a bag of words approach for evaluating the direction of some salient points detected in the moving objects. The main advantage deriving from this choice is that the representation is very robust with respect to the noise introduced both during the extraction of the salient points and the evaluation of the direction of their motion.

Once obtained the decisions from the three different experts, the system needs to properly combine them: the idea is that each classifier should have different voting priorities, depending on its own learning space. For this reason, the combination rule adopted is based on weighted voting, where the weights depend on the prediction confidence of each class the system has to recognize [60].

The main original contributions are: 1) the proposition of a novel system for characterizing the movement of the flame; 2) the use of a multi expert approach based on three complementary experts; 3) a wide characterization of performance on a standard dataset of videos, made available at *http://mivia.unisa.it*.

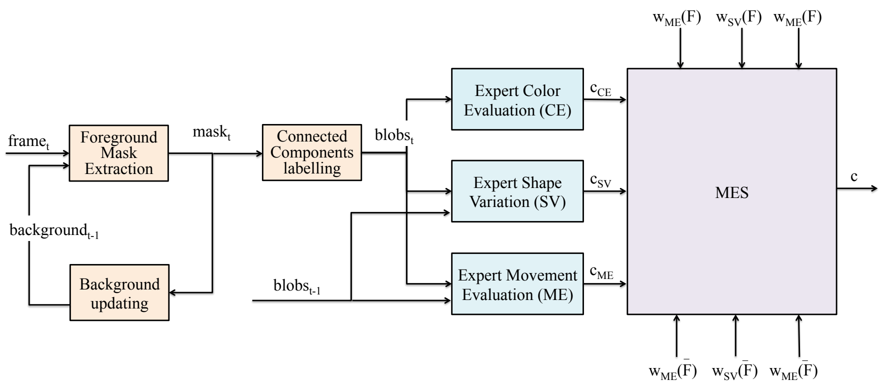
4.2 Proposed Architecture

An overview of the proposed approach is presented in Figure 4.1. Objects moving in the scene are first detected by using the algorithm we recently proposed in [65], which proved to be very effective both from a qualitative and a computational point of view: a model of the background (which represents the scene without any object moving inside) is maintained and properly updated (*Background updating*) so as to deal with the changes of the environments during the day; then, a background subtraction strategy is applied in order to obtain the foreground mask, encoding the objects moving in the scene (*Foreground mask extraction*). Finally, the *blobs*, each one being associated to an object, are obtained by a connected component labeling analysis [66] (*Connected component labeling*).

Three different experts have been introduced for evaluating the blobs: the first is based on color (*Color evaluation*, hereinafter *CE*); the second analyzes the shape of the blobs detected in the current frame with respect to the ones detected in previous frame (*Shape variation*, hereinafter *SV*); the third evaluates the movement of the blobs in two consecutive frames (*Movement evaluation*, hereinafter *ME*). The decisions taken by the experts are combined by a MES classifier based on a weighted voting rule, which finally assigns a class to each blob.

4.2.1 Multi expert evaluation

As mentioned before, one of the main choices determining the performance of a MES is the combination rule. Although several strategies have been proposed in the last years [67], it has been proved that one of the most robust to errors of the combined



frame_t
Foreground Mask Extraction
mask_t
Connected Components labelling
blobs_t
background_t-1
Background updating
blobs_t-1
Expert Color Evaluation (CE)
Expert Shape Variation (SV)
Expert Movement Evaluation (ME)
c_CE
c_SV
c_ME
MES
c
w_ME(F) w_SV(F) w_ME(F)
w_ME(F̄) w_SV(F̄) w_ME(F̄)

Figure 4.1 Overview of the proposed approach: the blobs are detected by a background subtraction algorithm; the decision is taken by combining information coming from three different experts, respectively based on color, shape variation and motion which analyze every input blob. Note that the last two experts need to be supplied with the blobs detected at the current frame and at the previous frame.

classifiers (both when combining classifiers based on the same features and when using different feature subsets in each classifier) is the weighted voting rule [60]. The main idea is that each expert can express its vote, which is weighted proportionally to the recognition rate it achieved for each class on the training set. For instance, let suppose that both CE and the ME classify the blob b as fire and that the percentage of fires correctly detected on the training set is 0.8 and 0.7 for the two experts respectively. Then, the two experts' votes for the class fire will be weighted 0.8 and 0.7.

In a more formal way, the generic k-th expert, being $k \in \{CE, SV, ME\}$, assigns to the input blob the class $c_k(b)$ chosen between the labels (F for *fire*, $\overline{F}$ for *non fire*); this can be formulated as a vote to the generic class i as follows:

$$\delta_{ik}(b) = \begin{cases} 1 & \text{if } c_k(b) \text{ gives the class } i \\ 0 & \text{otherwise} \end{cases} \tag{4.1}$$

In other words, if the output corresponds to the class, then the vote will be 1, otherwise it will be 0.

As suggested in [59], the weights $w_k(i)$ are dynamically evaluated by a Bayesian formulation in order to lead to the MES highest recognition rate. In particular, this formulation takes into account the performance of each expert on the training set of each class. More formally, given the classification matrix $C^{(k)}$ computed by the k-th expert on the training step, $w_k(i)$ can be determined by evaluating the probability that the blob b under test, belonging to the class i, is assigned to the right class c_k by the k-th expert:

$$w_k(i) = P(b \in i | c_k(b) = i) = \frac{C_{ii}^{(k)}}{\sum_{i=1}^{M} C_{ij}^{(k)}}, \tag{4.2}$$

being M the number of classes and $C_{(ij)}$ the value of the classification matrix in the position (i, j).

The final decision is taken by maximizing the reliability of the whole MES in recognizing that particular class.

In particular, the reliability $\psi(i)$ that the blob b belongs to the class i is computed by a weighted sum of the votes:

$$\psi(i) = \frac{\sum_{k \in \{CE,SV,ME\}} \delta_{ik}(b) \cdot w_k(i)}{\sum_{k \in \{CE,SV,ME\}} w_k(i)}. \tag{4.3}$$

The decision for the class c is finally taken by maximizing the reliability over the different classes:

$$c = \arg\max_i \psi(i). \tag{4.4}$$

4.2.2 CE: the expert based on color evaluation

This expert evaluates the color by analyzing its properties in the YUV color space; as already mentioned, YUV has been widely adopted in the literature since it separates the luminance from the chrominance and so is less sensitive to changes in brightness. In particular, as proposed on [28], this expert is based on the combination of six different rules, denoted as $r_1^c \dots r_6^c$, able to model the color of the flames.

In more details, as for r^c_1 and r^c_2, the idea is related to the experimental evidence that in most of flames the pixels exhibit a Red channel value greater than Green, as well as a Green greater than Blue [68]:

$$R(x, y) > G(x, y) > B(x, y). \tag{4.5}$$

Such conditions can be equivalently expressed in the YUV plane, by adopting the well known conversion rules [69], so that we obtain for the generic pixel (x,y) of the image:

$$r^c_1 : Y(x, y) > U(x, y); \tag{4.6}$$

$$r^c_2 : V(x, y) > U(x, y) \tag{4.7}$$

On the other side, r^c_3 and r^c_4 are based on the assumption that the Red component of fire pixels is higher than the mean Red component in the frame. Expressed in the YUV space, it implies that a fire pixel has the Y and V components higher than the mean Y and V value in the frame respectively, while the U component lower than the mean U value in the frame:

$$r^c_3 : Y(x, y) > \frac{1}{N} \cdot \sum_{k=1}^{N} Y(x_k, y_k) \tag{4.8}$$

$$r^c_4 : U(x, y) < \frac{1}{N} \cdot \sum_{k=1}^{N} U(x_k, y_k), \tag{4.9}$$

$$r^c_5 : V(x, y) > \frac{1}{N} \cdot \sum_{k=1}^{N} V(x_k, y_k), \tag{4.10}$$

being N the total number of pixels in the image.

Finally, in [28] it has been experimentally evaluated that fire pixels are characterized by a considerable difference between U and V components. Thus, the last rule can be defined as:

$$r_6^c : |V(x, y) - U(x, y)| \geq \tau_c. \tag{4.11}$$

In our experiments, τ_c has been set to 40, as suggested in [28].

The classifier decision c_{CE} is finally taken by evaluating the above mentioned rules. In particular, if all the conditions are verified, then the blob is assigned to the fire class:

$$c_{CE} = \begin{cases} F & \text{if } r_1^c \wedge r_2^c \wedge r_3^c \wedge r_4^c \wedge r_5^c \wedge r_6^c \\ \overline{F} & \text{otherwise} \end{cases} \tag{4.12}$$

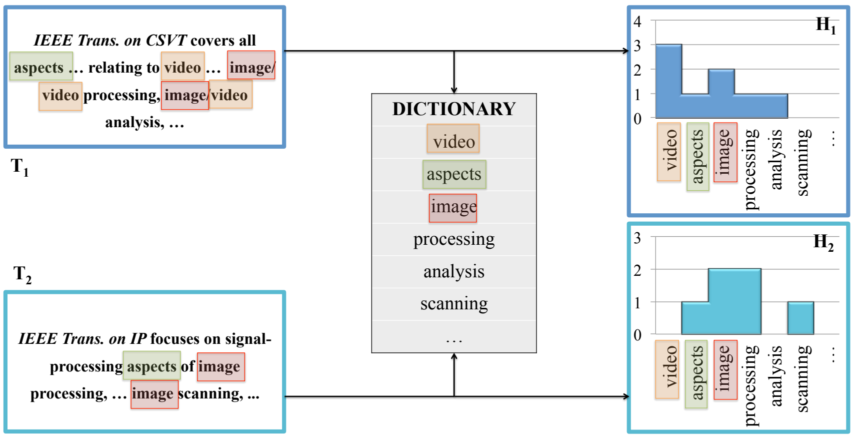

IEEE Trans. on CSVT covers all aspects ... relating to video ... image/ video processing, image/video analysis, ...
T1
T2
IEEE Trans. on IP focuses on signal-processing aspects of image processing, ... image scanning, ...
DICTIONARY
video
aspects
image
processing
analysis
scanning
...
H1
H2
4
3
2
1
0
video
aspects
image
processing
analysis
scanning
...

Figure 4.2 The rationale of the Bag of words approach applied to text classification: the occurrences, in a document, of a predefined set of words included into a dictionary are used for building up an histogram of occurrences. See the histograms H_1 and H_2 associated to texts T_1 and T_2 respectively.

4.2.3 SV: the expert based on shape variation

This expert (hereinafter referred as SV) analyzes the variation of the blob shape across two consecutive frames in order to exploit the observation that the shape of flames changes very quickly. In particular, as in [36], the algorithm computes, for each blob, the perimeter P_t and the area A_t of the minimum bounding box enclosed it. Such values are then used to compute the perimeter-area ratio r_t, which is an indicator of shape complexity:

$$r_t = \frac{P_t}{A_t}. \tag{4.13}$$

The shape variation s_v^t is then evaluated by comparing the shape measure computed at the frame t with the one obtained by the nearest blob detected at the previous frame $(t-1)$:

$$s_v^t = \left| \frac{r_t - r_{t-1}}{r_t} \right|. \tag{4.14}$$

The score s_v^t is finally analyzed; if it is higher than a given threshold, then the class fire is assigned to the input blob:

$$c_{SV} = \begin{cases} F & \text{if } s_v^t > \tau_v \\ \overline{F} & \text{otherwise} \end{cases} \tag{4.15}$$

4.2.4 ME: the expert based on movement evaluation

ME is based on a novel descriptor which adopts a bag-of-words approach [70], introduced in order to characterize the cluttered movement of fire. The rationale of this expert is based on the empirical observation that the parts of a flame appear to move at the same time in several different directions in a rather chaotic and unpredictable way; by contrast the parts of a rigid or articulated object show at each instant a quite limited set of motion directions. For translating this observation into an effective description and classification system, we have chosen a bag-of-words approach.

Bag-of-words has been successfully applied in several application domains, ranging from text classification to audio event detection and action recognition. The underlying idea is that the pattern to be classified is represented by the occurrences of low-level features (words) belonging to a dictionary and such occur-

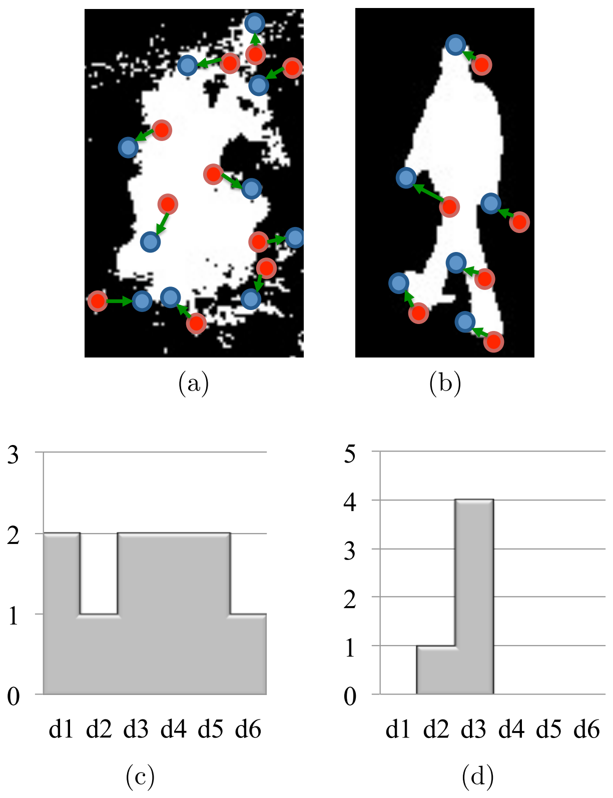


Figure 4.3 Low level representation (a,b) and high level representation (c,d) of a fire (a,c) and a non fire (b,d) blob. Red and blue circles in (a) and (b) represent the salient points extracted at the frame t and $t-1$ respectively.

rences are used to build a high-level vector; the generic component is associated to a word and its value is given counting to the occurrences of that word in the input pattern (see Figure 4.2 for an example).

In order to apply the bag of words strategy to our problem, the following main steps need to be dealt with: the extraction of the low-level representation, the definition of the dictionary that determines the construction of the high-level representation, and the paradigm adopted for the classification. An overview of the proposed approach is shown in Figure 4.4, while a more detailed explanation of the above mentioned phases will be provided in the following.

Low-level representation In order to capture the motion of the different parts of a foreground blob, salient points are extracted and matched across consecutive video frames.

The set of salient points is extracted by using the Scale Invariant Feature Transform (SIFT) [71] descriptors. At a given time instant t, the high-speed corner detection algorithm proposed in [72] is used to extract the set C_t of $|C_t|$ corners:

$$C_t = \{c_t^1, ..., c_t^{|C_t|}\}. \tag{4.16}$$

Each corner is then represented by measuring the local image gradients in the region around it, so obtaining the set of corresponding SIFT descriptors:

$$V_t = \{v_t^1, ..., v_t^{|V_t|}\}, \tag{4.17}$$

being $|V_t| = |C_t|$.

Given the corner points extracted in two consecutive frames (t and $t-1$) and the corresponding set of descriptors (V_t and V_{t-1}), the algorithm computes a set of matchings by pairing each point at time t with the one at time $t-1$ that is closest according to the Euclidean distance between SIFT descriptors:

$$M = \{m_1 \dots m_{|M|}\} \tag{4.18}$$

where:

$$m_j = (v_t^a, v_{t-1}^b) \tag{4.19}$$

such that:

$$b = \arg\min_i \|v_t^a - v_{t-1}^i\| \text{ and } \|v_t^a - v_{t-1}^b\| < \tau_M \tag{4.20}$$

For each matching m_j we consider the vector connecting the two corresponding corner points c_t^a and c_{t-1}^b, and extract the angle ϕ_j of this vector with respect to the x axis.

Figure 4.4 clarifies this concept: the corner points c_{t-1} and c_t, represented as red and blue circles respectively, are associated to their descriptors v_{t-1} and v_t. The matching m_j is represented by the green line connecting such points, while ϕ_j is the angle that m_j build with the x axis (black line).

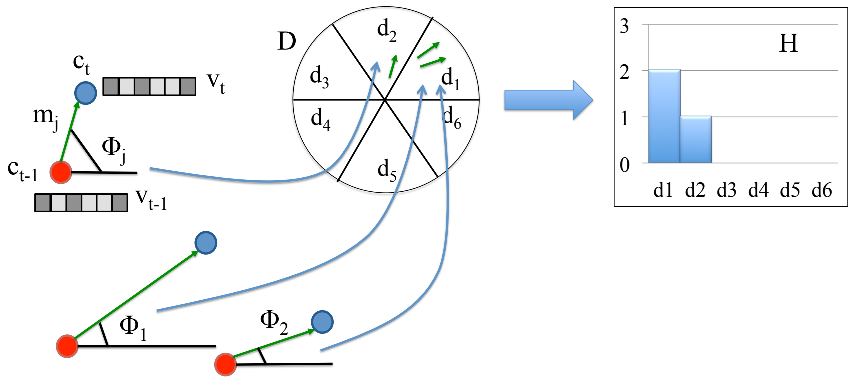


Figure 4.4 Given the corner points c_{t-1} and c_t (red and blue circles respectively), the matching m_j is obtained by minimizing the euclidean distance between the corresponding descriptors v_{t-1} and v_t. The direction of the motion, encoded by the angle ϕ_j, is evaluated according the dictionary D and the histogram of occurrences H is then built.

Dictionary According to the proposed low-level representation, the space of the possible words is the round angle ($0\,°-360\,°$). In order to obtain a adequately small finite set of words, we decide to uniformly partition the space into $|D|$ sectors d, so obtaining the dictionary D as follows:

$$D = \left\{ d_k \middle| \quad k = 1, ..., |D|; \quad d_k = \left] k \frac{2\pi}{|D|}, (k+1) \frac{2\pi}{|D|} \right] \right\}. \qquad (4.21)$$

$|D|$ has been experimentally set for this algorithm to 6: it implies that the resolution of the direction is $60\,^{\circ}$, which represents a good tradeoff between a suitable representation of the movement and the immunity to the noise.

High-level representation Given the dictionary D, for each blob the algorithm finds the words of D that occur in the blob, i.e. the intervals d_k that correspond to the motion direction of the salient points; then the blob can be represented by the histogram H of the occurrences of such words. An example is reported in Figure 4.3, where the low level representation (a,b) and the corresponding high level representation (c,d) for a fire (a,c) and a non fire object (b,d) is shown.

In a more formal way, the generic angle ϕ_j is associated to the index s_j, depending on the word d_k it belongs to:

$$s_j = |k : \phi_j \in d_k, k \in \{1, ..., |D|\}|. \qquad (4.22)$$

The set $S = \{s_1, ..., s_{|M|}\}$ associated to a generic blob b is then evaluated obtaining the histogram $H = \{h_1, ..., h_{|D|}\}$, whose generic element h_i can be computed as follows:

$$h_i = \sum_{l=1}^{|M|} \delta(s_l, i), \quad i = 1, ..., |D|, \qquad (4.23)$$

being $\delta(\cdot)$ the Kronecker delta.

Classification The main assumption used for designing the classifier is the evidence that the obtained feature vector is different for the two classes; in presence of fire the movement is disordered, determining occurrences of the words rather homogeneously distributed. Vice versa, when a rigid or articulated object moves in the scene, we mainly obtain values concentrated on a single or a few bins. See Figure 4.3 for an example.

For this reason, we introduce a measure of the homogeneity hm of the histogram:

$$hm = 1 - \frac{max(H)}{\sum_{k=1}^{|H|} h_k} \tag{4.24}$$

and consequently if it is higher than a given threshold, then the input is classified as fire, otherwise as not fire:

$$c_{ME} = \begin{cases} F & \text{if } hm > \tau_m \\ \overline{F} & \text{otherwise} \end{cases} \tag{4.25}$$

In Section 6.2 we will show the results obtained by the proposed video analitics algorithms.

Chapter 5

Data Flow Architecture for video surveillance platform

As introduced in Section 1.1, in this thesis we propose a novel and efficient architecture based on a data-flow paradigm, where the interfaces of the connected nodes are specified using a Node Definition Language (NDL), thus enabling the easy definition of new data types. The types of node inputs and outputs are statically defined, enabling compile-time checking within the implementation of each node, thus freeing the programmer from the burden of manual type checks. However, metadata about these types are retained in the nodes, allowing the platform to automatically ensure at run-time that only compatible nodes are connected to each other. The nodes are implemented as dynamic loading libraries, making it possible to extend the set of supported operations without recompiling the platform. The instantiation of the nodes and their interconnection are defined at initialization time by means of a configuration file; so it is easy to alter the processing network without recompiling the code. The execution of the nodes is directed by a scheduler that can allocate the work on the available processors or cores, exploiting parallelism even for a single data stream. Synchronization issues are dealt with by the middleware, which ensures proper locking of the involved data buffers before a node execution, and proper unlocking after its termination; locks are used according to the single writer/multiple readers pattern. Communication between nodes is based on shared memory, thus entailing a low overhead for medium-granularity nodes.

This architecture has been expressly designed to simplify some of the common issues we have encountered when designing video analytics/surveillance applications:

- the applications are highly customizable at configuration time; an installer can add or remove processing steps, or replace an implementation of a processing operation, with no need to access the application source code; this is important since even at a single deployment site, different cameras often need slightly different processing chains (e.g. not all the cameras may require shadow removal);

- the applications should use all the available processing power, by parallelizing as much as possible the work using all the available CPU cores; this is not trivial considering the previous point: during development it is not know exactly which processing steps will be enabled when the application will be deployed, since they can be easily changed by the installer; our platform provides an automatic distribution of the workload on the CPU cores, and also handles all the synchronization issues, thus avoiding a category of difficult to detect software bugs that could reduce the robustness of the application;

- memory leaks are a serious issue for video surveillance applications that need to run 24 hours a day, 7 days a week; while the use of a language with garbage collection (like Java) would solve this issue, video analytics still requires the performance of languages like C or C++ that lack this feature; our platform provides a memory management scheme which is less general than garbage collection, but this loss of generality is compensated by a much more efficient implementation, compatible with the performance requirements of video analytics;

- several video analytic functions could be applied over a single video stream (for instance in cases in which we could be interested in detecting fires, smokes and intrusions in sterile zones by using a single camera). Our platform gives the possibility to define nodes which are in common between the different video analytic algorithms and to execute them just a single time instead than one time for each algorithm;
- other features of the platform are aimed at solving general problems that are anyway commonly encountered in surveillance applications: for instance, the version management helps to check that there is no mismatch in the installed dynamic libraries (which could result in bugs difficult to be fixed by an installer), and the license management helps to ensure that the application is only used according to the conditions of the license (e.g. the number of video streams is restricted to the one specified by the license).

As the previous discussion of the literature has pointed out, each one of the features present in our platform has already been offered by some other architecture. However, to our knowledge, this is the first time that all these features are available in an integrated way in a single platform. The integration of all these features can determine some difficulties in their implementation, if one considers their mutual interactions: for instance, memory management would have been significantly simpler if it did not have to coexist with multi-threading, and the automatic parallelization of the workload could have been easier if the information on the processing steps that are to be performed would have been statically defined, and not decided when the application is started. However, the availability of all of them in a single platform can greatly simplify the task of a surveillance application developer, which require a unique combination of flexibility, efficiency and robustness.

In the following, we will first present, in Section 5.1, an overview of our architecture. Then we will discuss in Section 5.2 the development of an application using our platform, and in Section 5.3

the implementation of a single component. Section 5.4 will be dedicated to a detailed description of the services provided by the platform. Finally, in Section 6.3 we will describe our experience with the porting of some existing video-surveillance applications to our new platform, as a validation of its effectiveness.

5.1 The platform architecture

Fig. 5.1 shows an overview of the architecture of the proposed platform from the point of view of the code. As it can be seen, the platform is composed of a fixed middleware, and a set of plugins, that are isolated from each other, and only communicate through the middleware.

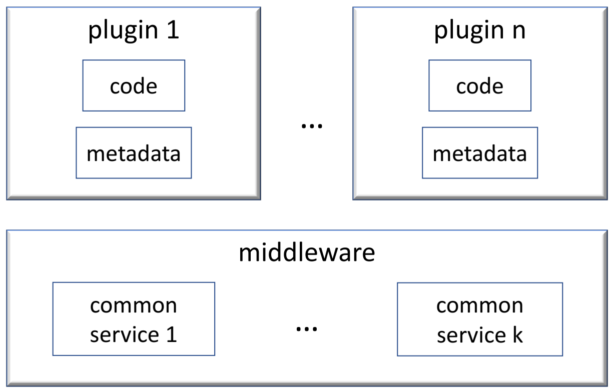


Figure 5.1 The code architecture of the proposed system. The application-specific code is distributed among dynamically loaded plugins, which also contain metadata about the implemented functionalities. The middleware uses these metadata to connect the plugins to each other. The middleware also provides common services that are independent of the application domain.

Plugins are dynamic-loading libraries that contain the implementation of one of more *nodes*, where a node corresponds to a processing activity needed by the application, and metadata describing these nodes. Typically, a node corresponds to a processing step clearly identifiable and with a well defined interface, that has a granularity sufficient to make it worth executing in parallel with other activities, and that is likely to be reused in a different application. For instance, the acquisition of a frame from a camera is an activity implemented as a node.

The middleware provides common services that are independent of the actual nature of the application:

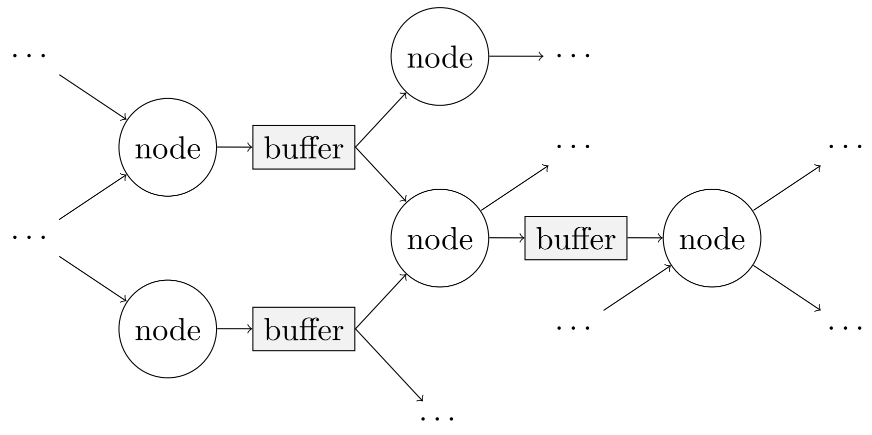


Figure 5.2 The execution architecture of the proposed system. Execution is based on the data-flow paradigm: a node is activated only when a datum is available in its input buffers, and a free space is available in its output buffers.

- *memory management*: the middleware takes care of allocation and deallocation of memory blocks for the input/output buffers of the nodes; it also provides functions for tracking the memory usage of each node;
- *scheduling/thread management*: a scheduler allocates the execution of each node on possibly several threads;
- *synchronization*: proper locking/unlocking is ensured for buffers shared between nodes;
- *configuration*: each node can have a set of configuration parameters; the middleware reads the parameter values from a common configuration file;
- *logging*: functions are provided to write messages on a log shared by all the nodes; messages can have different priority levels, and the log can be configured to write to file and/or display on the console a message on the basis of its level;

- *version management*: the middleware is able to keep different versions of the same plugin and to activate the appropriate one on the basis of the plugin metadata;
- *profiling*: the middleware can record the time spent in the execution of each node;
- *testing*: the middleware includes a utility to automate the unit testing of a node, providing the node with test inputs read from a file and checking the validity of the outputs either by comparing them with the expected outputs read from a file or by means of a user-defined validation function;
- *data file management*: a plugin can access any auxiliary data files it needs without the need to know where on the file system the application is currently installed;
- *license management*: the middleware includes components to generate and to verify software license keys, based on cryptography algorithms, to limit the use of an application to a specified machine and to a specified period of time.

These services will be described with more detail in Section 5.4.

The proposed architecture simplifies the development process of an application by promoting modularity, separation of concerns and reusability. In fact, the developer of a plugin is relieved both from the common tasks which the middleware provides for, and from the need of understanding the other part of the application, since a plugin only interacts with the middleware. Thus the developer can concentrate on the implementation issues of the specific task performed by the plugin. Furthermore, if an application is properly factored into plugins, it is very likely that some of the plugins can be reused for other applications (for instance, a camera acquisition plugin can be used for both a video-surveillance application and for a traffic monitoring system).

Besides the advantages during the development cycle, this architecture provides important benefits during the deployment and configuration of the application. Namely, since the plugins are dynamically loaded and their activation and configuration is controlled by means of a configuration file, it is easy to change on the fly an application setup, adding or removing processing steps, or replacing an implementation of a processing step with a different one, to adapt the application to the special requirements of a particular installation. Furthermore, the ability of the middleware to keep different versions of the same plugin eases the migration to a new release of the application.

5.2 Definition of an application using the platform

From the application designer point of view, a running instance of an application based on the proposed platform can be considered as a data-flow network, as illustrated in Fig. 5.2. The processing is performed by nodes, each taking possibly some data items as inputs, and producing possibly other data items as outputs. A node is executed as soon as its inputs are ready, and there is available buffer space for its outputs. Node inputs and outputs are statically typed, and the type information is contained in the metadata of the plugins that implement the nodes. The middleware uses this information to allocate and manage the buffers through which nodes exchange data items, and to check the compatibility between nodes connected to a same buffers. Each buffer can have only one producer, that is, a node that writes to the buffer; however, several consumers (i.e. nodes that read from the buffer) can be connected to it; the middleware takes care of ensuring the proper synchronization between the producer and the consumers.

The application designer specifies which nodes constitute the processing network and how they are to be connected by means of a Network Configuration File (NCF), that is a text file based on an ad-hoc description language. In particular, the NCF allows the designer to describe in a declarative way which plugins have to be used (including constraints on the plugin versions), which nodes have to be instantiated, how they have to be configured (by means

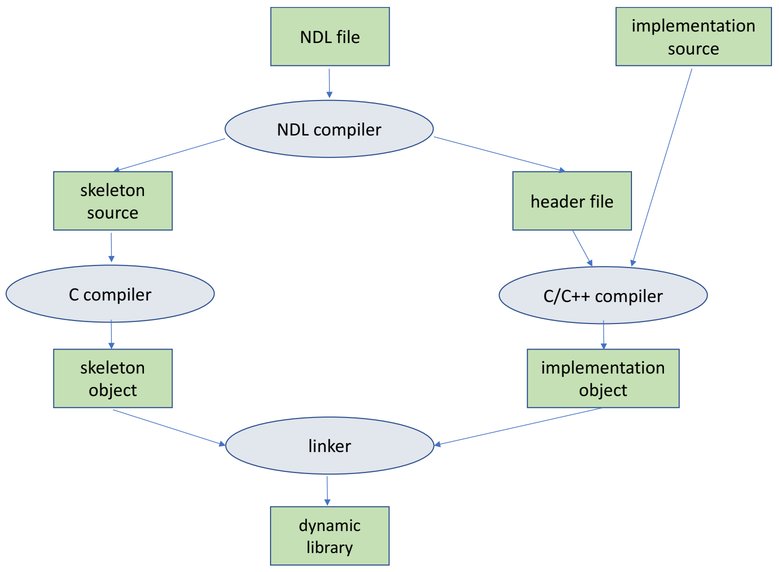


Figure 5.3 The compilation process for plugin implementing primitive nodes. Stating from a definition of the node interfaces in the Node Description Language (NDL), a NDL compiler produces a skeleton with the code for connecting to the middleware, that has to be linked together with the code implementing the node functionalities in order to obtain a dynamically loaded plugin.

of node-specific configuration parameters) and how they have to be interconnected. If needed, the design can provide additional information such as the size of each single buffer, to fine tune the operation of the application.

If new nodes are to be developed (which might not be the case, if the current application can be completely defined reusing plugins already available from previous applications), the designer can specify the interface of each node using a Node Description Language (NDL). The NDL files are then passed to the plugin developers in order to obtain an implementation for the nodes.

5.3 Node development

The development of a node implementation (a plugin) starts with a description of the node interface through the Node Description Language (NDL). A NDL file specify for each of the nodes implemented by a plugin which are the inputs, the outputs and the configurable properties. For these attributes of a node, the NDL assigns a name and a type; this latter can be either a built-in type or a user defined type. Fig. 5.4 shows an example NDL file for a face detection plugin.

Once the node interface is defined, the developer can choose the implementation technology to be used for the plugin. The current release of the middleware supports two kinds of implementation:

- *primitive nodes*, which are implemented by means of C/C++ functions compiled into a dynamic loading library;
- *composite nodes*, which are implemented by aggregating other nodes from existing plugins, specifying how they have to be assembled by means of a NCF file; once a composite node is defined, it can be used exactly as if it were a primitive one, thus allowing to raise the abstraction level of the plugins by introducing higher and higher level nodes built upon the foundations of the lower levels.

In future releases of the platform, other implementation technologies are expected to be introduced (for instance, implementations based on scripting languages); the middleware architecture is flexible enough to accomodate quite easily such extensions.

As regards the implementation of primitive nodes, Fig. 5.3 shows the steps and the intermediate files needed for building a primitive plugin. A NDL compiler is used to obtain both a C-language skeleton (containing metadata and automatically generated functions for interfacing the plugin with the middleware) and a header file, with the declaration of the data types and function prototypes to be used by the user-supplied implementation functions. The skeleton is compiled and linked together with the user-supplied implementation files, to generate a dynamic loading

```
library face_detection version 1.0.0
  # User-defined types
  type size=struct
              width, height: int32
            end
       rect=struct
              left, top, right, bottom: int32
            end

  # Node interface for the node type "detect"
  node detect
    inputs
       frame : image
    outputs
       faces : array of rect
    # Configurable parameters of the node
    properties
       min_size, max_size: size
       scale_factor: float64
       do_canny_pruning: bool
  end

end
```

Figure 5.4 An example NDL file defining the interface of a face detection plugin.

library. A make-file for compiling and linking the plugin can be automatically generated using a script provided with the platform.

5.3.1 The node library for video analytics

In the course of the adoption of the proposed platform for our video surveillance applications, we have developed a library of reusable nodes that implement common tasks found in this kind of systems. In Table 5.5, we highlight some of the nodes available. Notice that for some tasks there are several dedicated nodes, providing the same interface but differing in the implementation. Thus it is possible to change the algorithm used for some processing step by simply modifying the application NCF file.

`frame_acquisition`	Acquires the frames from a camera or a video file, using the functions provided by OpenCV.
`background_subtraction`	Given a frame and a reference background, compares the two using a dynamic threshold and produces a foreground mask.
`background_update`	Updates a background model represented by the mean color value for each background pixel.
`mog_background_update`	Updates a background model where each pixel is represented by a Mixture-of-Gaussians (MoG); has the same interface as `background_update`.
`connected_components`	Detects the connected components in the foreground mask.
`small_blobs_filter`	Removes from the foreground the “blobs” smaller than a threshold.
`shadow_removal`	Removes shadow pixels from the foreground.
`reflection_removal`	Given a set of detected foreground objects, removes reflection pixels (due to a reflective floor) from the bottom of each object.
`inverse_perspective_mapping`	Given a set of detected foreground objects, computes their position in 3D space by inverting the perspective transform.
`overlap_tracking`	Gives an identifier to detected objects trying to preserve the object identities across the video sequence; uses the measure of the overlap of foreground regions across adjacent frames to establish a correspondence.
`bgm_tracking`	Gives an identifier to detected objects trying to preserve the object identities across the video sequence; uses Bipartite Graph Matching, and a similarity measure that takes into account the position, the size and the color of the objects. Has the same interface as `overlap_tracking`.
`log_event_sink`	Registers detected events on a text-based log file.
`mysql_event_sink`	Registers detected events on a relational database using the MySQL DBMS. Has the same interface as `log_event_sink`.
`gui_event_sink`	Shows detected events to the user on a window. Has the same interface as `log_event_sink`.

Figure 5.5 A subset of the node library.

5.4 Common services provided by the platform

This section describes the services provided by the middleware that are independent of the exact nature of the application being developed. Some of these services are available by means of APIs to the node developers, while others are activated automatically at runtime, without requiring modifications of the node source code.

5.4.1 Memory management

Dynamically allocated data structures are the source of many subtle programming errors in languages like C and C++ that lack garbage collection. Often these errors, even when discovered, are difficult to debug because it is not easy to spot the portion of the code that was responsible for the error; this problem gets even more acute in programs based on a plugin architecture, where software components independently developed are dynamically added at run time.

Our middleware addresses this problem in two ways:

- by reducing the need to resort to dynamic memory allocation, taking complete responsibility for the allocation and deallocation of data structures used by the nodes to communicate with each other;
- by providing a set of allocation and deallocation functions that enable the middleware to track the memory operations and to check for common errors.

For the first point, all the data exchanged between nodes is constrained to pass through buffers, as illustrated in Section 5.2; the memory used for the buffers is completely taken care of by the middleware. Thus the node developer is not concerned with the deallocation of the input data received by the node, nor with the allocation of the output data produced by the node. Notice that the data structures may not have fixed length. This is obviously the case of images, which may have different dimensions; but the platform also supports variable-length arrays of other data types (including array of images, or arrays of arrays). In these cases, the node has to specify with an API call the desired dimension for the output data structures, in terms of their data type: for instance, for images the width and height are specified, not the number of bytes. Fixed-length data structures are allocated once at program startup; for variable-length data structures the platform attempts to avoid allocating memory at each processing cycle, in order to reduce the computational cost, by maintaining a cache of allocated memory areas.

Regarding the second point, the platform includes memory allocation functions, specialized for the different supported datatypes, that assist the developer in debugging memory errors. More specifically, in normal mode of operation, these functions work as their equivalent standard counterparts (except that they always check for memory exhaustion), and only a negligible runtime overhead is incurred. However the application can also be started in a memory debugging mode; in this case for each allocated memory block the platform keeps additional information, that is used to check if the block is correctly deallocated (signaling if a block is never deallocated or is deallocated twice) and to report the node that has performed the allocation/deallocation of the block. Furthermore, the platform can be instructed to periodically report in the log (described in Section 5.4.5) the total memory allocated by each node, so as to detect memory leaks in node implementations.

Notice that the memory management support provided by the platform is less extensive and general than a Garbage Collection mechanism, such as the ones included in several programming languages (e.g. Java). This fact does not limit significantly the usefulness of memory management for the video surveillance applications, and has made possible an effective implementation that has a far lighter impact on the processing speed than a full-fledged garbage collector would have had.

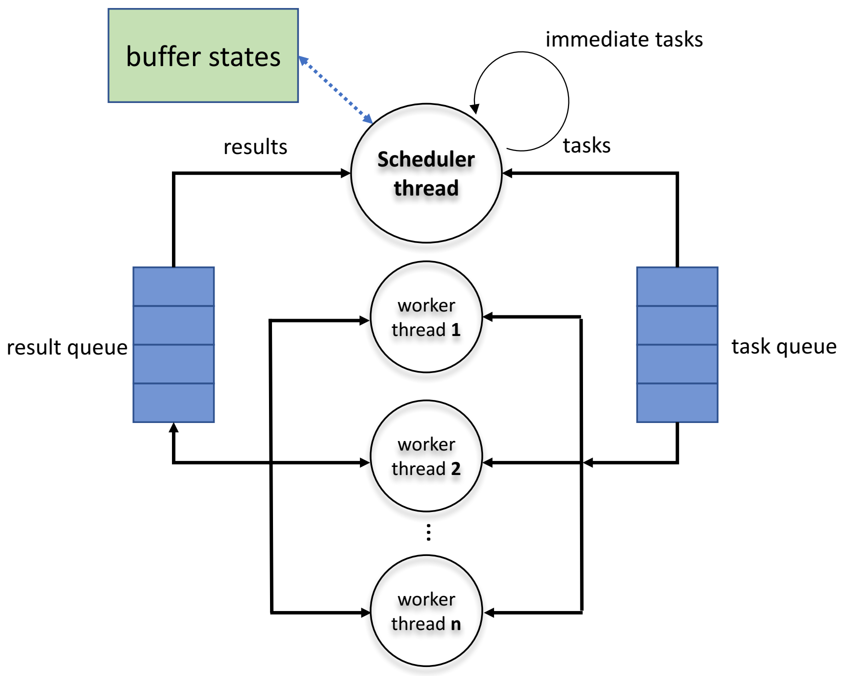


Figure 5.6 The multi-threaded scheduler. The main scheduler thread determines, on the basis of the buffer states, which nodes can be activated, and for them creates task objects in the task queue (*immediate* tasks, for trivial operations, are executed immediately by the scheduler thread without being put in the task queue). The worker threads in the thread pool pick tasks from the task queue and execute the code of the corresponding nodes, putting a result object in the result queue. Finally, the scheduler thread reads the result objects and updates the corresponding buffer states, possibly making more nodes ready to be activated.

5.4.2 The scheduler

The scheduler is the component of the middleware that manages the execution of the individual nodes of the application. Notice that the adopted scheduling algorithm is not an original contribution of this architecture; here we provided a detailed description of the scheduler only with the aim of making more clear how this component of the platform interacts with the rest of the proposed architecture. The current version of the platform actually includes two different scheduler implementations: the single-threaded scheduler and the multi-threaded scheduler.

As its name implies, the single-threaded scheduler uses a single thread to perform the execution of all the nodes. This scheduler is not able to activate more nodes in parallel, and so it is not recommended on multi-processor or multi-core systems. However, since it does not have to care for synchronization issues, its internal structure is simpler, and it has a lower computational overhead. Essentially this scheduler manages a queue of *tasks* to be performed, using auxiliary data structures to check when a task is ready to be fired. For primitive nodes, there is a one-to-one correspondence between a node and a task; on the other hand, composite nodes are usually associated to more tasks.

The multi-threaded scheduler divides the execution of the nodes among multiple threads, as shown in Fig. 5.6. On multi-processor or multi-core machines this implies that multiple nodes can be run in parallel; even on low-end single-core machines, technologies like hyper-threading allow the application to achieve a speedup (although not as conspicuous as on multi-core machines) with respect to the single-threaded scheduler. The multi-threaded scheduler uses data structures that are similar to the ones of the single-threaded scheduler, only they use the proper locking discipline when accessing shared data. To avoid the large overhead incurred by thread creation, the scheduler uses a thread pool: a fixed number of worker threads is created at startup, and they cyclically pick a task from the ready queue, execute it, and notify the scheduler of its completion. So a thread services numerous tasks, possibly related to different nodes, during its lifetime. This pooling discipline, which is widely adopted also in other domains (e.g. web servers) is essential for video analytics applications whose workload is characterized by very short tasks that must be repeated for each frame of each camera.

In order to avoid the synchronization overhead for trivial tasks, nodes can be flagged as *immediate* in their plugin. Tasks corresponding to immediate nodes are executed directly by the scheduler thread, and not passed to worker threads.

Notice that the choice of the scheduler to be used is not wired in the application, but is made at startup time, thus giving the user the possibility to decide for each single computer whether to use or not multi-threading, and how many threads must be created. This can be very important for cases where the video analytics application must coexist on the same computer with other programs; a common occurrence is when, on sites with a very small number of cameras, the same server must be shared by the video management software and the video analytics software.

5.4.3 Synchronization

When the multi-threaded scheduler is used, the platform must properly synchronize the access to shared data by different threads. Traditionally, the application developer is in charge to ensure proper synchronization in the code he/she writes. In the context of a traditional video surveillance application, two approaches are commonly used: the first is to avoid the problem by avoiding running in parallel two processing steps for the same video stream; in this way, the only parallelism is between different video streams, which could not allow to exploit all the available computational power. The other approach is that the developer must remember to use the appropriate synchronization mechanism each and every time the code makes use of a potentially shared data structure.

Since synchronization is error prone, and synchronization bugs are usually very hard to solve, because they manifest themselves in a non deterministic way, it is important to relieve the node developers from this task. Thus the platform ensures automatically the synchronization for all the data structures that are accessed by a node, making the resulting application inherently more robust.

Each node is allowed to access only two kinds of memory areas: the input and output buffers to which it is connected, and its internal node state, that is allocated at node initialization time. Since the node states are not shared, they pose no synchronization problems; the access to the input and output buffers, instead, needs to be performed by one thread at a time. This is ensured by the platform, which activates a node only when it is safe to do so, that is when its input data are ready and there is space available in its input buffers. For the outputs of a node, the platform ensures that other nodes waiting for these outputs are not activated until the producing node has terminated its processing. For the inputs of a node, the platform ensures that the producer of the datum cannot access the same memory buffer until the first node has terminated; however other consumer nodes can access the same memory areas.

The synchronization is realized using thread-safe queues as the basic mechanism: the scheduler uses such a queue to pass to the worker threads the tasks to be performed and the references to the buffers to be used, while the worker threads use another queue to pass back to the scheduler information on the completion of the task. The platform includes two different implementations of thread-safe queues: the first one is based on Pthreads synchronization primitives (locks and condition variables), while the second one is based on FastFlow [73], a low-level programming framework for implementing lock-free queues. This latter has the advantage that a thread accessing a queue does not incur in the (small) overhead associated with the operation of a lock; however each thread remains in the running state even when it has to wait for the queue (busy wait), and so it may leave less CPU time available to other applications on the same computer.

```
a_color = [0xFF 0x80 0x20] # a shade of orange
list_of_rectangles = [ {x=10 y=7 width=100 height=80}
                       {x=40 y=128 width=38 height=122} ]
background_image = image "background.jpg"
a_solid_blue_image = image 800*600 of [0 0 255]
```

Figure 5.7 An excerpt from a configuration file, showing some examples of the supported syntax.

5.4.4 Configuration

Node implementations may need a set of parameters to tune their algorithms. It would be inefficient to let each module to define its own format for the configuration file: first, there would be a duplication of code for parsing the configuration files; second, and most important, the system integrator deploying an application would have to struggle with several configuration files written using different syntaxes.

The platform provides a common configuration framework, in which the configuration for each node instance can be either centralized in the NCF file or moved to a different file included by the NCF. The common syntax for the configuration parameters does not only make provisions for the built-in basic data types, but also allows user-defined structured data types. Among the supported types for a parameter there is the image type; a value for an image parameter can be constructed either by specifying the numerical values of the pixels in text format, or by using an image file in one of the most commonly used formats. Fig. 5.7 presents an excerpt from a configuration file, showing some examples of the supported syntax.

5.4.5 Logging

The ability to write log messages can be useful for at least two purposes: saving information that can may help debugging the application, or collecting statistics that can be used for performance tuning. In the traditional approach, since the application is divided into modules that are independently developed, either each module developer would devise an ad hoc, possibly different, solution for logging, or would use some system-wide logging tool such as syslog.

The platform provides logging functions that ensure that all the node implementations use a common log format, thus easing log post processing; also the log functions make the access to the log file properly synchronized in case a multi-threaded scheduler is used by the application. Furthermore, since logging can be both a time and space consuming activity (if detailed information is written on the log), the platform allows the user to specify at run time the desired level of logging detail: each logging call specifies a priority level, and the user can choose a range of levels that are to be saved on the log file and the range of levels that are to be directly written on the system console.

An important advantage of the integration of logging in the platform is that the logging subsystem is aware of the runtime architecture of the processing network, and inserts in the log information on which node instance has generated each message; this would have been very difficult for the developer to do with a non-integrated logging facility, since during the development it is not known where and how many times each node will be instantiated in an application.

5.4.6 Version management

In a software architecture based on dynamic linking a very common problem is the mismatch between the versions of different modules that have to work together. This often leads to the so-called *DLL hell* on the Windows platform, where updating a component of the system may cause other, apparently unrelated components to stop working. To avoid this problem, our platform supports a versioning of the modules. Each module has a version number, and each time a node is instantiated (explicitly or implicitly, as it happens when the node is part of the implementation of a composite node), it is possible to specify a constraint on the required version. An important feature of the implemented version management is that different versions of the same module can coexist (as long as the have different version numbers); thus if a module is updated, the new version can be introduced gradually in the applications based on the platform.

This feature is extremely useful in an application domain like surveillance systems, where often the applications have a large degree of customization, and must be upgraded without loosing the previous configuration and customization work, and thus without cleaning out the modules of the previous releases.

5.4.7 Profiling

In a real-time video analysis application, performance is often a critical aspect. In order to optimize such an application, it is necessary to be able to spot the bottlenecks that limit the overall throughput. In order to help the developers obtaining this information, our platform supports an execution mode, selectable at startup time, in which the scheduler collects statistics about the execution times and the number of activations of each node. These statistics are periodically saved on a text file.

Performance optimization of a parallel application gets more complicated because of the need to avoid conditions that limit the parallelism, thus attaining a speed-up that is not proportional to the number of available processors. The profiling subsystem of our platform specifically addresses this aspect, being able to analyze the dependency graph of the application and to highlight, whenever the number of active nodes if less than the number of worker threads, which part of the network is not balanced.

For video analytics applications, the availability of this information impacts both developer, that can use it to optimize the code, and installers or system integrators, that can exploit it to fine tune the performance of the system.

5.4.8 Testing

In recent years, automated unit testing has emerged as a practice recommended by several software engineering methodologies to improve the quality and the reliability of software systems. A unit test verifies the behavior of a software component in isolation; the automation of such tests ensures that they can be easily executed whenever the software is modified, so as to immediately discover if a bug has been introduced.

The architecture of our platform, in which an application is composed by loosely coupled nodes that interact only through their inputs and outputs, makes easy to test each node in isolation. Furthermore, the platform includes a tool that, using the information contained in the NDL files, automatically generates a C++ file with a main function that loads the plugin, instantiates the node to be tested, and feeds it with input values from a test configuration file; the developer can add to this file a function to verify the correctness of the node outputs, or can provide in the test configuration file the desired output values. Thus, it is possible to create with little effort a suite of tests for each plugin; these tests can be executed automatically by means of a script.

The support for automated unit testing helps improving the correctness and the robustness, which are important qualities for security-related applications such as video surveillance.

5.4.9 Data file management

A plugin implementation may need to access some data files containing information needed for its operation. For example, we have a face detection node that is a wrapper around the Viola-Jones object detector included in OpenCV [74]. This detector requires an external file containing the cascade classifier parameters in order to perform its operation.

While a standard application can access an external file using a fixed path, a plugin is not allowed to impose any constraint on the position of its files in the file system, since this would complicate the deployment of the final application, which includes several independently developed plugins. Thus the platform includes an API by which a plugin can access its data files using a search path that can be defined at run time in a way that is transparent to the plugin developer; this mechanism also allows the installer to provide alternate versions for some of the files that override the ones bundled with the plugin, without altering the plugin directory structure, by simply redirecting some of the access requests.

5.4.10 License management

A commercial video analysis application often needs a mechanism for enforcing the licensing conditions, thus preventing an unauthorized use of the software. Usually the license constrains the number of computers on which the program can be used and the number of video streams that can be processed on each computer.

Our platform include a license management service, which requires the generation of a software key, protected by means of a cryptographic algorithm, for each machine on which the software is installed. The key contains information identifying the machine, so that it cannot be reused on a different computer, information about the expiration of the license, the number of authorized video streams and the possibility to include other application specific fields. By centralizing the license management within the middleware, two benefits are obtained: first, the application developers are relieved from the task of devising a license verification mechanism and of checking the license within the code of the application specific modules; second, the license checking can be integrated with the scheduler, thus ensuring that it will be performed for each activated node.

Chapter 6

Experimental Results

6.1 Tracking Algorithm Results

In this section we will show the results of the tracking method, detailed in Chapter 3. In particular, in Subsection 6.1.1 we introduce the standard datasets used for the experimentation. In Subsection 6.1.2 the parameters selected for the experimentation are justified, while in Subsections 6.1.3 and 6.1.4 a quantitative and qualitative evaluation is respectively provided.

View	**Camera Model**	**Resolution**	**Frame Rate**
1	Axis 223M	768x576	7
3	Axis 233D	768x576	7
4	Axis 233D	768x576	7
5	Axis 223M	720x576	7
6	Axis 223M	720x576	7
7	Canon MV1	720x576	7
8	Canon MV1	720x576	7

Figure 6.1 PETS Dataset: Camera Specification [75].

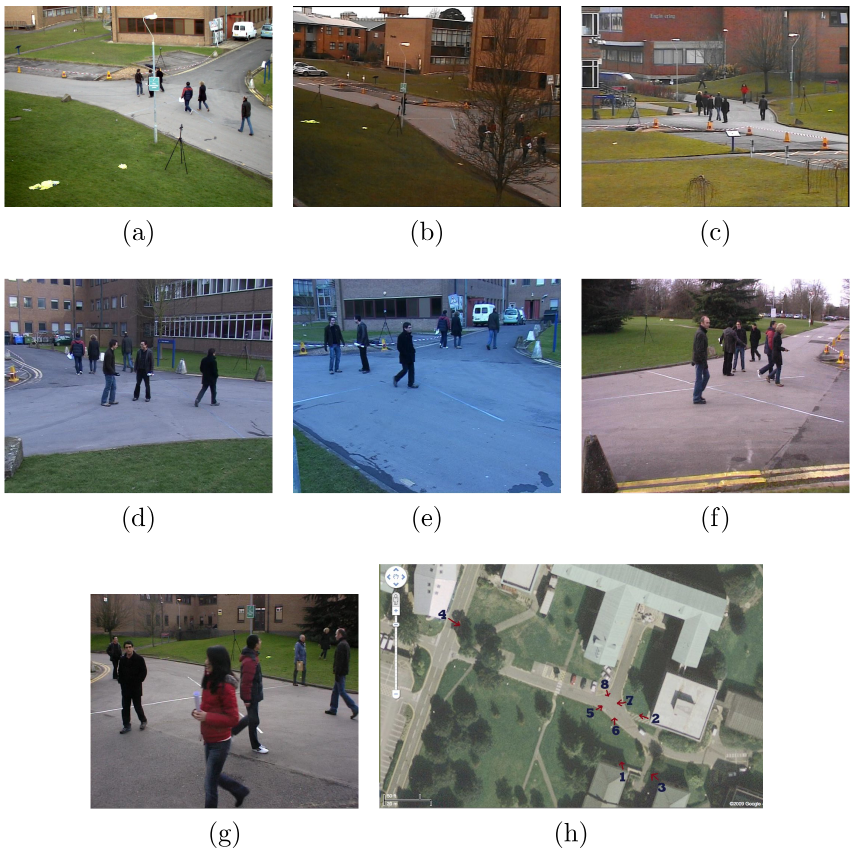


Figure 6.2 Camera views used in the PETS2010 database. We can note that each view emphasizes one or more problem. For example, the first one (a) causes occlusions between persons and the pole while the second one (b) is characterized by several occlusions between persons and the tree. In (h) the position of the cameras on the plane is shown.

6.1.1 Datasets

In order to assess the performance of the method with respect to the state of the art, we have used the publicly available PETS 2010 dataset [75], currently used by many research papers.

PETS 2010 Dataset: it has been recorded at Whiteknights Campus, University of Reading, UK in 2009. It is composed by three datasets: S1 concerns person count and density estimation, S2 addresses people tracking and S3 involves flow analysis and event recognition. In this thesis we focus on the dataset S2 (hereinafter PETS 2010 Dataset), made of seven videos, containing several occlusions between a person and an object, two persons or among several persons. Figure 6.2 shows an example for each considered view of the PETS 2010 database, while more information are provided in Table 6.1.

6.1.2 Parameters setup

In the proposed tracking method the only parameter that needs to be properly set up is the d_{max} parameter of equation 3.13. In order to evaluate it, we have computed in each view the maximum speed of the objects, from which we have derived the following values: $d_{max} = 100$ for views 1, 3 and 4 and $d_{max} = 150$ for view 5, 6, 7 and 8 of the PETS dataset.

6.1.3 Quantitative Evaluation

In this section a quantitative evaluation of the proposed method over both the datasets is performed.

Experimentation 1

The first quantitative evaluation of the method has been carried out using the performance indexing proposed in [76]. In particular, we have used the following indices, especially suited for tracking algorithms: the *Average Tracking Accuracy* (ATA), the *Multiple Object Tracking Accuracy* (MOTA) and the *Multiple Object Tracking Precision* (MOTP). In the following we introduce some notations useful to formally define them.

Let G_i and D_i be the ith ground truth object and the ith detected one at the sequence level, respectively; $G_i^{(t)}$ and $D_i^{(t)}$ denote the ith ground truth object and the detected one in frame t; $N_G^{(t)}$ and $N_D^{(t)}$ denote the number of ground truth objects and detected ones in frame t, respectively, while N_G and N_D denote the number of ground truth objects and unique detected ones in the given sequences. N_{frames} is the number of frames in the sequences. Finally, N_{mapped} refers to the mapped system output objects over an entire reference track, taking into account splits and merges and $N_{mapped}^{(t)}$ refers to the number of mapped objects in the frame t.

ATA is a spatiotemporal measure that penalizes fragmentations in spatiotemporal dimensions while accounting for the number of objects detected and tracked, missed objects, and false positives. ATA is defined in terms of Sequence Track Detection Accuracy (STDA):

$$STDA = \sum_{i=1}^{N_{mapped}} \frac{\sum_{t=1}^{N_{frames}} \frac{|G_i^{(t)} \cap D_i^{(t)}|}{|G_i^{(t)} \cup D_i^{(t)}|}}{N_{G_i \cup D_i \neq 0}}. \tag{6.1}$$

The latter measures the overlap in the spatiotemporal dimensions of the detected object over the ground truth, taking a maximum value of N_G. The ATA is defined as the STDA per object:

$$ATA = \frac{STDA}{\left[\frac{N_G + N_D}{2}\right]}. \tag{6.2}$$

As already mentioned, the MOTA is an accuracy score that computes the number of missed detections, false positives and switches in the system output track for a given reference ground truth track. It is defined as:

$$MOTA = 1 - \frac{\sum_{t=1}^{N_{frames}} \left(c_m \cdot m_t + c_f \cdot fp_t + c_s(is_t)\right)}{\sum_{t=1}^{N_{frames}} N_G^{(t)}}, \tag{6.3}$$

where m_t is the number of misses, fp_t is the number of false positives, and is_t is the number of ID mismatches in frame t considering the mapping in frame $(t-1)$; c values are weights chosen as follows:

$$c_m = c_f = 1; c_s = log_{10}(\cdot).$$

Finally, the MOTP is a precision score that calculates the spatiotemporal overlap between the reference tracks and the system output tracks:

$$MOTP = \frac{\sum_{i=1}^{N_{mapped}} \sum_{t=1}^{N_{frames}^{(t)}} \frac{|G_i^{(t)} \cap D_i^{(t)}|}{|G_i^{(t)} \cup D_i^{(t)}|}}{\sum_{t=1}^{N_{frames}} N_{mapped}^{(t)}}. \tag{6.4}$$

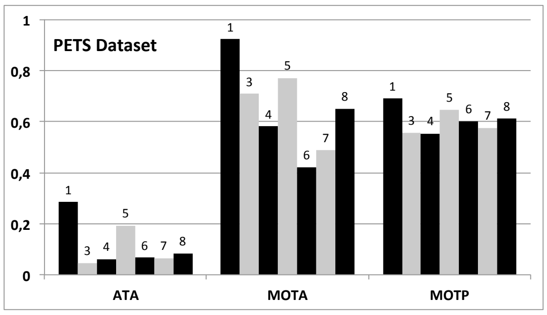


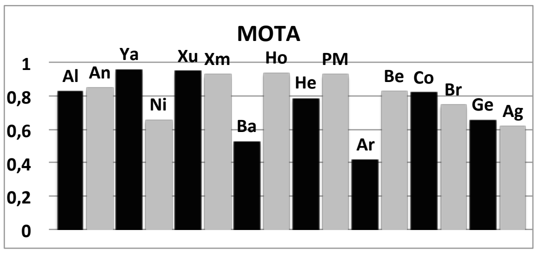
Figure 6.3 Performance of the proposed method for the considered views of the PETS 2010 dataset. The numbers on the bars correspond to the views number.

The obtained results are shown in Figure 6.3 that shows the results of the proposed method over the PETS 2010 dataset, related to the individual sequences. We can note that the performance is strongly influenced by the complexity of the single sequence. This complexity is not determined only by the typology of the single view (*i.e.*, the presence of the pole in the first view or the presence of the tree in the third one, which covers one-third of the scene taken by the camera), but also by the interactions among the tracked objects.

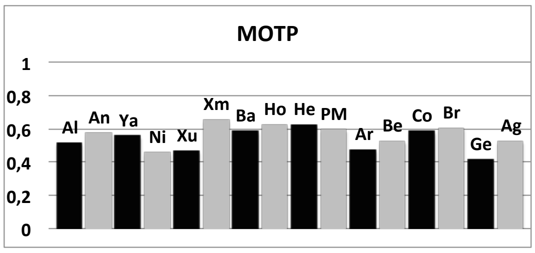
At this point we can examine in detail the considered views, analyzing their performance in relation to the complexity of the scene. First view presents interactions among two or three objects; the only difficulty is due to the presence of the pole and of the sign hanged on it, which causes a lot of splits. Note that the proposed method proves to be particularly robust with respect to the split situations on this view. Views 3 and 4 are the most complex, as shown by the results displayed in Figure 6.3. Indeed, as already mentioned, view 3 is characterized by the presence of a large tree (about one-third of the scene), occluding a lot of individual or group objects. The situation is further complicated by the complexity of interactions among the objects, which involves in the average $2-5$ objects for view 3 and $2-6$ for view 4. Another problem in view 4 is the presence of a white-orange ribbon, continuously moving because of the wind. Such situation causes a lot of problems also in the detection phase. The problem of the moving ribbon is also present in views 5, 6, 7 and 8, even if it is less visible. We can note that the performance obtained in views 6 and 7 is generally lower than that obtained on other sequences; this is related to more complex interactions between the tracked objects, having a very high number of occlusions associated to objects that are entering the scene (unstable objects). It is worth noting that the method, during an occlusion, does not attempt to find the exact position of an object inside a group; it continues to track the group as a whole, using the Kalman filter for obtaining a prevision of the position of each object inside the group itself; this choice obviously causes a degradation of the performance if it is measured using indices defined assuming that objects are always tracked individually.

Comparison: PETS Contest (Performance Evaluation of Tracking and Surveillance) is a competition organized by the University of Reading, which allows to compare all the state of the art tracking algorithms: each participant has to submit the output of the proposed method over a standard dataset; the output is processed by the organizers and the results are finally disclosed during the contest session.

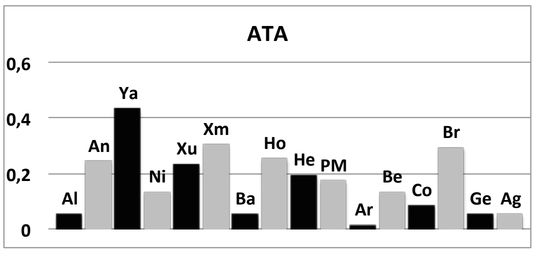
The proposed algorithm participated to the last PETS 2013 contest [77][78] and, as highlighted by the organizers, it ranked first in terms of MOTA and in the first positions in terms of ATA and MOTP. Although the results have not been made available, in Figure 6.4 we report the ones computed over View 1, declared during the final contest session. Our method strongly outperforms all the other ones working in real time and based on a single cam-



(a)



(c)



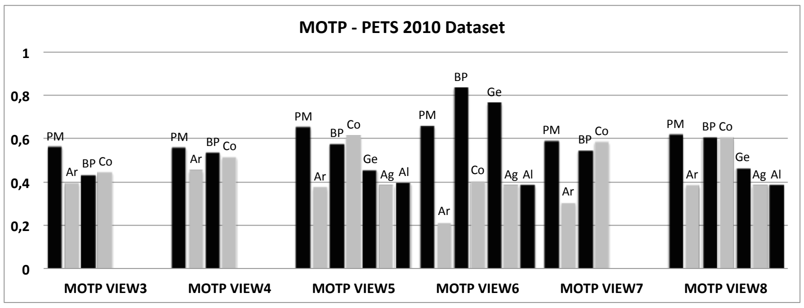
(d)

Method	Abbrev.
Alahi Olasso	Al
Anon	An
Yang	Ya
Nie	Ni
Xu	Xu
Xu multiview	Xm
Badie	Ba
Hoffman	Ho
Heili	He
Proposed Method	PM
Arsic	Ar
Berclaz	Be
Conte	Co
Breitenstein	Br
Ge	Ge
Alahi Ogreedy	Ag

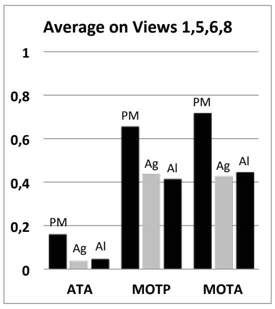
(b)

Figure 6.4 Comparison of the proposed method with the participant to the last PETS 2014 competition, in terms of MOTA (a), MOTP (c) and ATA (d). In (b) the associations between methods and abbreviations is summarized.

era view: in fact, it is worth to point out that Breitenstein et al. (Br) [79] and Xu et al. (Xu) [13] use a multi-camera approach, while Badie et al. (Ba) [4] and Hoffman et al. (Ho) [5] consider a post-processing of the trajectories in order to link all the extracted tracklets. It should be clear that our method extracts the objects' positions at each frame, without applying any kind of post-processing aimed at linking the tracklets. It is a very important and not negligible feature in the field of behavioral analysis,



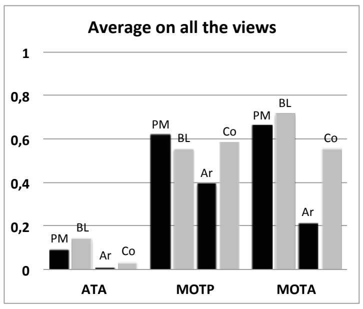
(a)



(b)

Method	Abbrev.
Proposed Method	PM
Breitenstein	Br
Leykin	Le
Sharma	Sh
Yang	Ya
Berclazdp	BD
Berclazlp	BL
Arsic Winter	AW
Conte	Co
Ge	Ge
Alahi Ogreedy	Ag
Alahi Olasso	Al
Arsic	Ar

(c)



(d)

Figure 6.5 Performance of the proposed method compared with the PETS 2010 contest participants on MOTP index (a), on Views 1, 5, 6 and 8 (b) and on all the views (e). In (c) the associations between methods and abbreviations is summarized.

since we are interested in detecting abnormal behaviors in real time, when the objects are still inside the scene.

Furthermore, a deeper comparison has been performed with the methods participating to the previous PETS 2010 contest, whose results are available in [80]. Figure 6.5 summarizes the obtained results. In particular, Figure 6.5a gives an overview of the precision index (MOTP) over all the views. It is evident that the proposed method outperforms all the other methods on six out of seven considered views in terms of precision. Figure 6.5b provides the average performance, in terms of ATA, MOTA and MOTP obtained on Views 1, 5, 6 and 8, the only ones taken into account by Alahi et al. [81].

Finally, Figure 6.5d shows the average results over all the views: our method is the most precise over the entire dataset and it is outperformed, in terms of accuracy, only by the method proposed by Berclaz *et al*. [16], which takes advantage of the use of a multi-camera approach and thus it is not directly comparable with our approach.

It is worth highlighting that ATA, MOTA and MOTP do not perfectly fit, for their nature, the proposed method. It is due to the fact that our approach, during an occlusion, does not attempt to find the exact position of an object inside a group; it continues to track the group as a whole, using the Kalman filter for obtaining a prevision of the position of each object inside the group itself; this choice obviously causes a degradation of the performance if measured using indices assuming that objects are always tracked individually.

In general, our method confirms a very high accuracy and precision; this result is mainly a direct consequence of the fact that it solves many of the errors usually occurring in tracking algorithms that do not distinguish between single and multiple objects.

Experimentation 2

In this section the performance of the proposed method in terms of resolution percentage of split and occlusion patterns is analyzed. The results are summarized in Table 6.6, where a comparison with our previous method [1] is performed over the PETS dataset. Note that [1] is more robust with respect to the occlusion occurrences,

View	*Split Resolution Percentage*		*Occlusion Resolution Percentage*	
	[1]	**Proposed**	[1]	**Proposed**
1	45%	73%	75%	95%
3	36%	51%	61%	76%
4	35%	74%	45%	74%
5	15%	51%	56%	67%
6	12%	56%	51%	62%
7	16%	61%	52%	63%
8	20%	51%	42%	59%

Figure 6.6 Comparison between the proposed method and [1] in terms of split and occlusion patterns resolution over the PETS dataset.

rather than to the split ones. It is mainly due to the association manager module, which uses a greedy strategy to solve split and occlusion patterns. The main novelty of the proposed approach lies in the introduction of a graph based approach, which proves to significantly improve the performance with respect to [1], both in terms of split and occlusion patterns.

Experimentation 3

A further experimentation, shown in Table 6.7, presents some more general evaluation criteria, which reflect the possibility to correctly follow a trajectory, assigning it one or more id [82]. In particular:

- *TP* (True Positive) refers to the number of trajectories followed for more than the 75% of their life, also with different identifiers. An example of *TP* is shown in Figure 6.8a;

- *PTP* (Perfect True Positive) refers to the number of trajectories followed for the 100% of their life with the same identifier. An example of *PTP* is shown in Figure 6.8b;

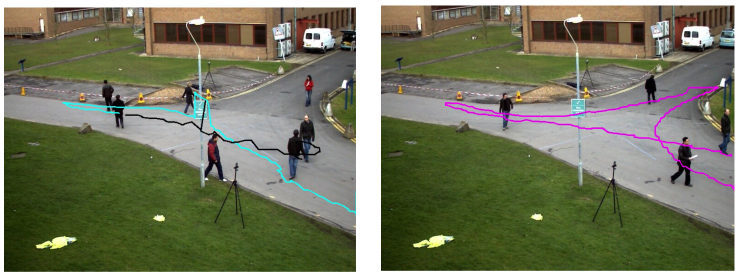
- *FN* (False Negative) refers to the number of trajectories followed for less than the 75% of their life;

PETS Dataset										
V	**GT**	**TP**	**PTP**	**FP**	**FN**	**IDS**	**Av**	**Min**	**Max**	**AvI**
1	21	21	4	2	0	43	232,6	86	575	2
3	21	15	0	7	6	82	306,4	72	792	3
4	23	18	0	19 (13)	5	89	264,5	23	792	3
5	28	28	1	4	0	81	97,3	22	293	2
6	33	32	1	28(20)	1	142	95,3	7	320	4
7	31	28	0	17	3	123	148,1	32	320	3
8	30	25	1	6	5	95	143,8	13	417	3

Figure 6.7 Tracking results on PETS Dataset. (V: View; GT: Ground Truth trajectories; TP: True Positives, at least 75% of the track without id-switches; PTP: Perfect True Positives, 100% of the track without id-switches; FN: False Negatives; FP: False Positives; IDS: Id Switches; Av: Average trajectory length; Min: Minimum trajectory length; Max: Maximum trajectory length; AvI: Average number of id-switches).

- *FP* (False Positive) refers to the number of spurious trajectories followed for more than two seconds;
- *IDS* (ID-Switch) refers to the number of times an object changes its identifier.
- *Av* refers to the average trajectories length;
- *Min* refers to the minimum trajectories length;
- *Max* refers to the maximum trajectories length;
- *AvI* refers to the average number of id-switches for each trajectory;

It is worth noting that the high number of false positive trajectories, especially in Views 4 and 6 of the PETS dataset, is caused by very frequent detection errors: in particular, as already mentioned, in View 4 it is caused by the presence of the white-orange moving wire, while in View 6 it is related both to the presence of the wire and to the white car, wrongly identified as an object by the detection phase during all the sequence. All these kinds of detection errors, identified in Table 6.7 by the numbers in brackets, could be reduced in a very simple way, by applying a filter to the detection phase, taking into account the particular shape and



(a) (b)

Figure 6.8 Examples of true positive trajectory (a) and perfect true positive trajectory(b).

appearance of the spurious objects. Since we are only interested in the proposed tracking method, we do not investigate into the performing of the detection phase.

6.1.4 Qualitative Evaluation

A qualitative evaluation has been finally performed in order to confirm the efficiency of the proposed approach. In particular, we show how the proposed algorithm deals with splits and occlusions respectively in Figure 6.9 and 6.10: in particular, Figure 6.9 reports an example of split caused by an error during the detection phase and properly adjusted. On the other hand, Figures 6.10 shows some excerpts of the video sequences, with two complex occlusion patterns among three and two persons; as it can be seen, the system preserves the object identities across the occlusions.

Finally, we show in Figure 6.11 the trajectories obtained by applying the proposed algorithm over the PETS 2010 Dataset. Despite the complexity of each view, the trajectories are generally stable and reliable, so confirming the goodness of the proposed approach.

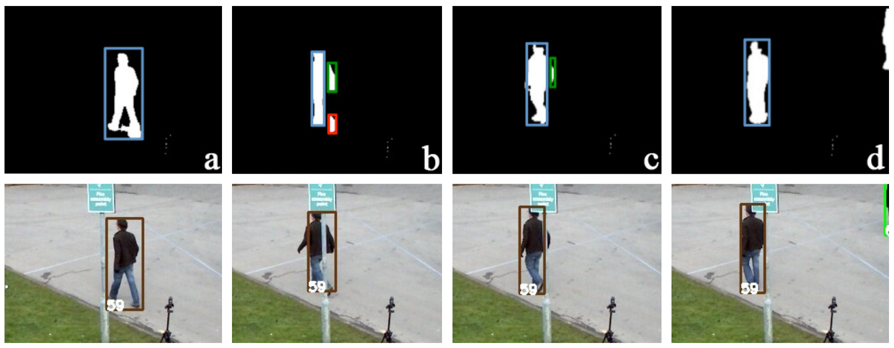


Figure 6.9 The output of the people tracking method containing a split; the first and the second row respectively represent the input and the output of the method.

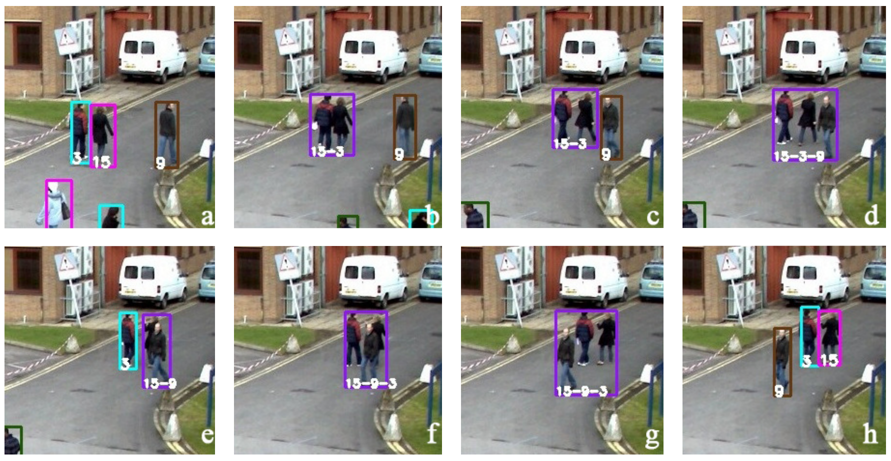


Figure 6.10 The output of the people tracking method containing an occlusion. Note how the object 9 is correctly tracked inside the different groups although it quickly changes its direction in the frame (c).

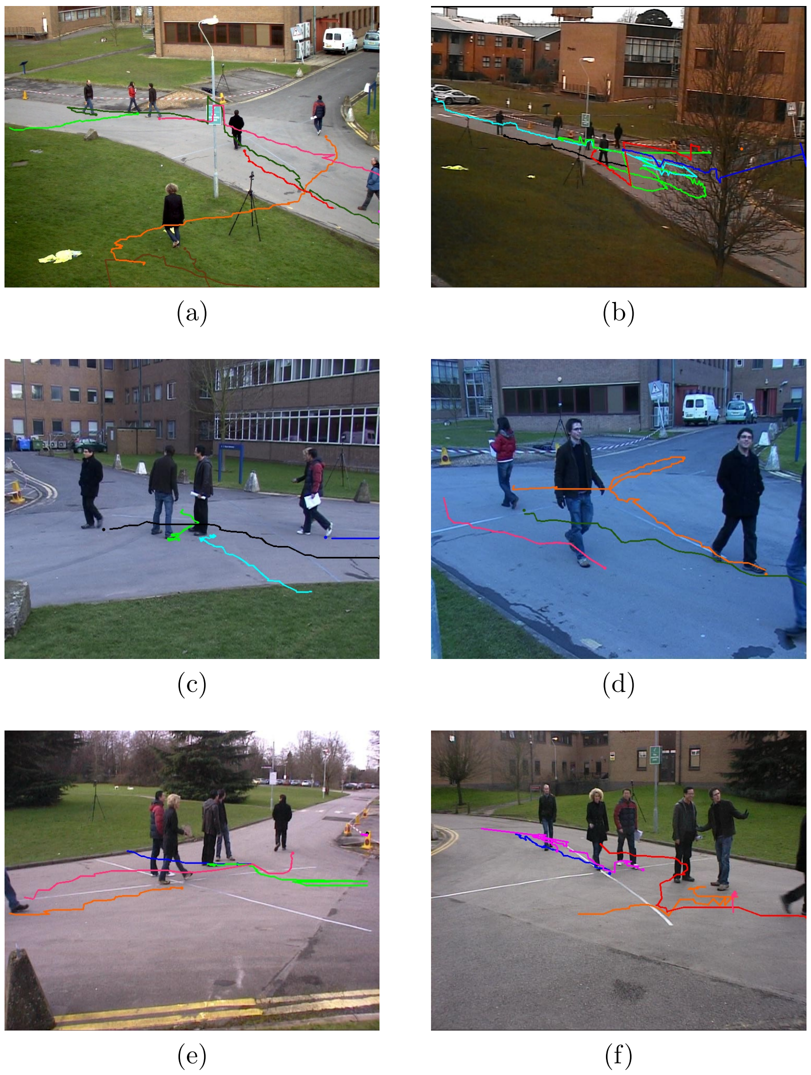


Figure 6.11 Output of the proposed algorithm for Views 1 (a), 3 (b), 5 (c), 6 (d), 7 (e) and 8 (f) of the PETS dataset.

6.2 Fire Detection Results

In this section we will show the results of the fire detection method, detailed in Chapter 4. In particular, in Subsection 6.2.1 we introduce the standard datasets used for the experimentation. In Subsection 6.2.2 and 6.2.3 a quantitative and a computational cost evaluation is respectively provided.

6.2.1 Datasets

Most of the the methods in the literature (especially the ones based on the color evaluation) are tested using still images instead of videos. Furthermore, no standard datasets for benchmarking purposes have been made available up to now. One of the biggest collection of videos for fire and smoke detection has been made available by the research group of Cetin [83][34]. Starting from this collection, composed by approximatively 31.308 frames, we added several long videos acquired in both indoor and outdoor situations so resulting in a new dataset composed by 62.748 frames and more than one hour of recording.

More information about the different videos are reported in Table 6.12, while some visual examples are shown in Figure 6.13[1].

Note that the dataset can be seen as composed by two main parts: the first 14 videos characterized by the presence of fire and the last 17 videos which do not contain fires; in particular, this second part is characterized by objects or situations which can be wrongly classified as containing fire: a scene containing red objects may be misclassified by color based approaches, while a mountain with smoke, fog or clouds may be misclassified by motion based approaches.

Such composition allows us to stress the system and to test it in several conditions which may happen in real environments.

The dataset has been partitioned into two parts: 80% has been

[1]The whole dataset can be downloaded from our website: *http://mivia.unisa.it/datasets/video-analysis-datasets/fire-detection-dataset/*.

Video	Resolution	Frame Rate	Frames	Fire	Notes
Fire1	320x240	15	705	yes	A fire generated into a bucket and a person walking near it. Vidmakeo downloaded from [83].
Fire2	320x240	29	116	yes	A fire very far from the camera generated into a bucket. The video has been downloaded from [83].
Fire3	400x256	15	255	yes	A big fire in a forest. The video has been acquired by [84] and downloaded from [83].
Fire4	400x256	15	240	yes	See the notes of the video *Fire3*.
Fire5	400x256	15	195	yes	See the notes of the video *Fire3*.
Fire6	320x240	10	1200	yes	A fire generated in a red ground. Video downloaded from [83].
Fire7	400x256	15	195	yes	See the notes of the video *Fire3*.
Fire8	400x256	15	240	yes	See the notes of the video *Fire3*.
Fire9	400x256	15	240	yes	See the notes of the video *Fire3*.
Fire10	400x256	15	210	yes	See the notes of the video *Fire3*.
Fire11	400x256	15	210	yes	See the notes of the video *Fire3*.
Fire12	400x256	15	210	yes	See the notes of the video *Fire3*.
Fire13	320x240	25	1650	yes	A fire in a bucket in indoor environment. Video downloaded from [83].
Fire14	320x240	15	5535	yes	Fire generated by a paper box. The video has been acquired by the authors near a street.
Fire15	320x240	15	240	no	Some smoke seen from a closed window. A red reflection of the sun appears on the glass. Video downloaded from [83].
Fire16	320x240	10	900	no	Some smoke pot near a red dust bin. Video downloaded from [83].
Fire17	320x240	25	1725	no	Some smoke on the ground near a moving vehicle and moving trees. Video downloaded from [83].
Fire18	352x288	10	600	no	Some far smoke on a hill. Video downloaded from [83].
Fire19	320x240	10	630	no	Some smoke on a red ground. Video downloaded from [83].
Fire20	320x240	9	5958	no	Some smoke on a hill with red buildings. Video downloaded from [83].
Fire21	720x480	10	80	no	Some smoke far from the camera behind some moving trees. Video downloaded from [83].
Fire22	480x272	25	22500	no	Some smoke behind a mountain in front of the University of Salerno. The video has been acquired by the authors.
Fire23	720x576	7	6097	no	Some smoke above a mountain. The video has been downloaded from [83].
Fire24	720x576	10	400	no	Some smoke on a mountain. Video downloaded from [83].
Fire25	352x288	10	140	no	Some smoke far from the camera in a city. Video downloaded from [83].
Fire26	720x576	7	847	no	See the notes of the video *Fire24*.
Fire27	320x240	10	1400	no	See the notes of the video *Fire19*.
Fire28	352x288	25	6025	no	See the notes of the video *Fire18*.
Fire29	720x576	10	600	no	Some smoke in a city covering red buildings. Video downloaded from [83].
Fire30	800x600	15	1920	no	A person moving in a lab holding a red ball. The video has been acquired by the authors.
Fire31	800x600	15	1485	no	A person moving in a labwith a red notebook. The video has been acquired by the authors.

Figure 6.12 The dataset used for the experimentation.

used to test the proposed approach while 20% for training the system by determining the weights of the MES.

In order to further confirm the effectiveness of the proposed

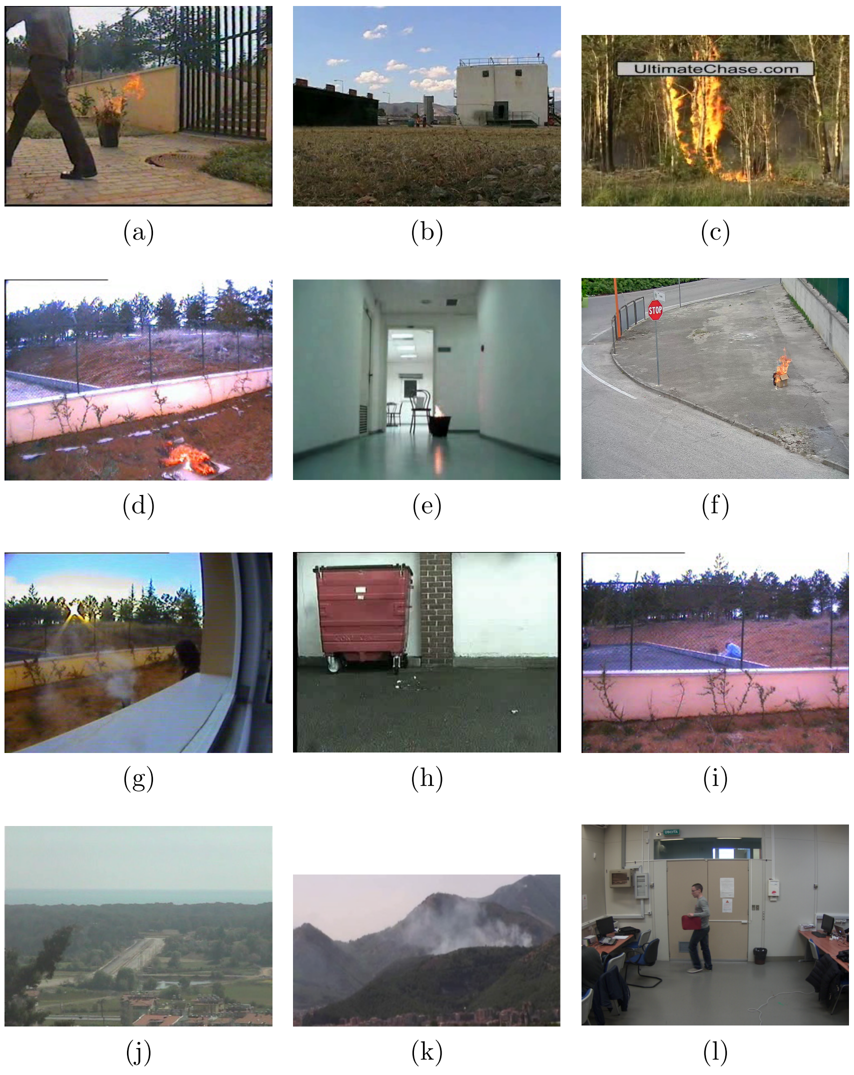


Figure 6.13 Examples of images extracted from the videos used for testing the method: (a) *fire1*, (b) *fire2*, (c) *fire4*, (d) *fire6*, (e) *fire13*, (f) *fire14*, (g) *fire15*, (h) *fire16*, (i) *fire19*, (j) *fire20*, (k) *fire22*, (l) *fire31*.

approach, especially with respect to the false positive rate, we also evaluated it over a second freely available dataset (hereinafter

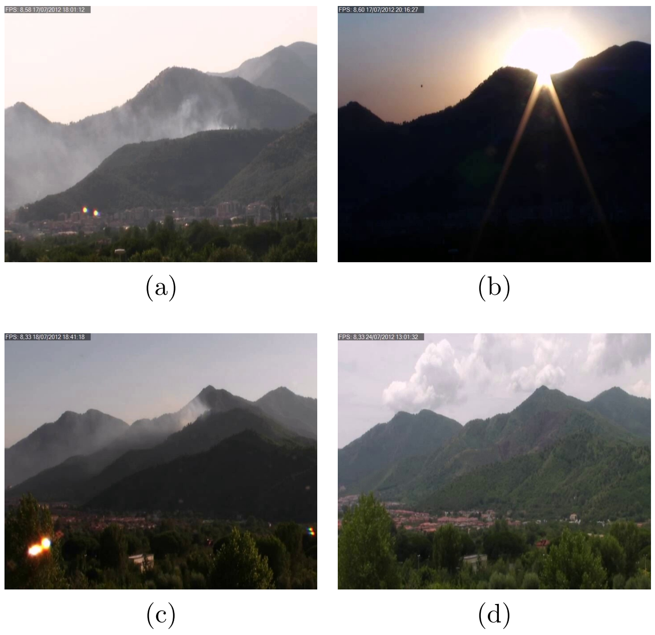

(a) (b) (c) (d)

Figure 6.14 Some examples of the Dataset D2, showing red houses in the wide valley, the mountain at sunset and some lens flares.

D2)[2]. It is composed by 149 videos, each lasting approximatively 15 minutes, so resulting in more than 35 hours of recording; D2 contains very challenging situations, often recovered as fire by traditional color based approaches: red houses in a wide valley (see Figures 6.14a and 6.14d), a mountain at sunset (see Figure 6.14b) and lens flares (bright spots due to reflections of the sunlight on lens surfaces, see Figures 6.14a and 6.14c).

6.2.2 Quantitative Evaluation

In this section a quantitative evaluation of the proposed method over both the datasets is performed.

[2]The whole dataset can be downloaded from our website: *http://mivia.unisa.it/datasets/video-analysis-datasets/smoke-detection-dataset/*.

An overview of the performance achieved on the test set, both in terms of accuracy and false positives, is summarized in Table 6.18.

Among the three experts considered is Section 4.1 (CE, ME and SV), the best one is the CE, which achieves on the considered dataset a very promising performance (accuracy = 83.87% and false positives = 29.41%). Note that such performance is comparable with the one reached by the authors in [28], where over a different dataset the number of false positives is about 31%.

On the other hand, we can also note that the expert ME, introduced for the first time in Section 4.2.4 for identifying the disordered movement of fire, reveals to be very effective. In fact, we obtain a 71.43% accuracy and 53.33% false positives. It is worth pointing out that the considered dataset is very challenging for this expert: in fact, the disordered movement of smoke as well as of trees moving in the forests can be easily confused with the disordered movement of the fire. This consideration explains the high number of false positives introduced by using only ME.

As expected, the best results are achieved by the proposed MES, which outperforms all the other methods, both in terms of accuracy (93.55%) and false positives (11.76%). The very low false positive rate, if compared with state of the art methods, is mainly due to the fact that ME and SV act, in a sense, as a filter with respect to CE. In other words, ME and SV are able to reduce the number of false positives introduced by CE without paying in terms of accuracy: this consideration is confirmed by the results shown in Figure 6.16, where the percentage of the number of experts which simultaneously take the correct decision is reported. In particular, Figure 6.16a details the percentage of the number of experts correctly assigning the class fire: we can note that all the experts correctly recognize the fire in most of the situations (69%), while two experts assign the class fire in the remaining 31%.

The advantage in using a MES is much more evident in Figure 6.16b, which refers to non fire videos. In this case, only 17% of videos are correctly classified by all the experts. On the other hand, most of the videos (61%) are assigned to the correct class by two experts, so confirming the successful combination obtained thanks to the proposed approach.

In order to better appreciate the behavior described above, a few examples are shown in Figure 6.15; in Figure 6.15a the fire is correctly recognized by all the experts: the color respects all the rules, the shape variation in consecutive frames is consistent and the movement of the corner points detected is very disordered. A different situation happens in Figure 6.15b, where the only classifier detecting the fire is the one based on the color: in this case, the uniform movement of the salient points associated to the ball as well as its constant shape allow the MES to avoid a false positive introduced by the use of a single expert. In Figures 6.15c and 6.15d other two examples are shown: in particular, in the former a small fire with a variable shape has both a uniform color and a uniform movement of the salient points. The combination of color and shape variation experts helps the proposed system to correctly detect the fire. The last example shows a very big but settled fire, whose shape is stable and so is not useful to correctly assign the class fire. In this situation, the combination between the experts based on color and motion respectively allows the MES to take the correct decision about the presence of fire.

Although the situations are very challenging, no false positives are detected by our MES. The result is very encouraging, especially if compared with CE, achieving on the same dataset 12% of false positives. It is worth pointing out that such errors are localized in approximatively 7 hours, mainly at sunset, and are due to lens flares. Such typology of errors is completely solved by the proposed approach, able to take advantage of the disordered movement of the flames.

6.2.3 Computational cost

Finally, we have also evaluated the computational cost of the proposed approach over two very different platforms: the former is a traditional low-cost computer, equipped with an Intel dual core T7300 processor and with a RAM of 4GB. The latter is a Rasp-

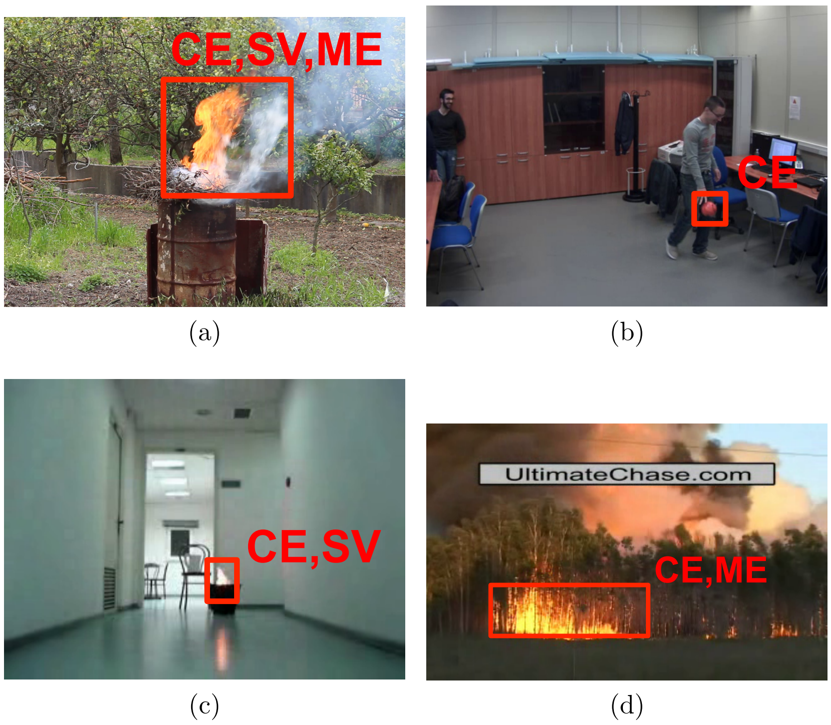
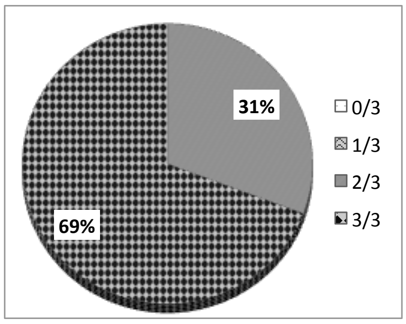
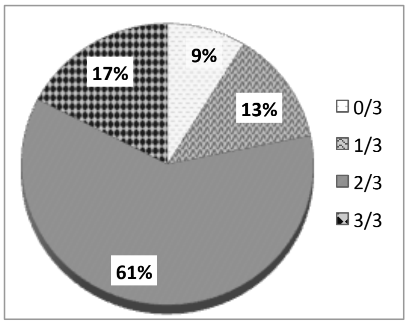


Figure 6.15 The three experts in action; the red box indicates the position of the fire, while the letter on it refers to the expert recognizing the presence of the fire.



(a)



(b)

Figure 6.16 Number of experts simultaneously taking the correct decision in fire (a) and non fire (b) videos. For instance, 31% of situations are correctly assigned to the class fire by two experts over three while in the remaining 69% all the three experts correctly recognize the fire.

berry B, a Broadcom BCM2835 System-on-a-chip (SoC), equipped with an ARM processor running at 700 MHz and with a RAM of 512 Mb. The main advantage in using such device lies in its affordable cost, around 35 dollars.

The proposed method is able to work, considering 1CIF videos, with an average frame rate of 60 fps and 3 fps respectively over the above mentioned platforms. Note that 60 fps is significantly higher than the traditional 25 - 30 fps that a traditional camera can reach during the acquisition. It implies that the proposed approach can be easily and very effectively used on existing intelligent video surveillance systems without requiring additional costs for the hardware needed for the images processing.

In order to better characterize the performance of the proposed approach, we also evaluated the time required by the different modules, namely the three experts (CE, ME and SV) and the module in charge of updating the background, extracting the foreground mask and labeling the connected components (FM). The contribution of each module is highlighted in Figure 6.17: the average time required to process the single frame has been computed and the percentage of each module with respect to the total time is reported. We can note that SV only marginally impacts on the execution time; this is due to the fact that the search of the minimum bounding boxes enclosing the blobs and of its properties (in terms of perimeter and area) is a very low-cost operation. Although the introduction of SV only slightly increases the performance of the MES (from 92.86% to 93.55% in terms of accuracy), the small additional effort strongly justifies its introduction in the proposed MES.

On the other side, the higher impacts are due to ME and CE: as for the former (85%), it is evident that the computation of the salient points, as well as their matching, is a very onerous operation. As for the latter, it may appear surprising the big effort required by the CE with respect to FM (CE: 11%, FM: 2%). It is worth pointing out that FM's operations (such as background updating and connected component labeling) are very common in computer vision, and thus very optimized versions have been proposed in standard libraries such as OpenCV.

Finally, it is worth pointing out that the computation time is strongly dependent on the particular image the algorithm is processing. In fact, it is evident that pixel-based modules (such as FM and CE) need to process the whole image independently of the objects moving inside. On the other hand, it is evident that the more are the objects moving inside the scene, the higher is the effort required by FM for detecting and analyzing the salient points. It implies that the variance with respect to the overall time required for the computation is about 51% of the overall time. Note that the final combination of the decisions taken by the three experts has not been considered, since the time required is very small with respect to the other modules.

In conclusion, the obtained results, both from a quantitative and a computational point of views, are very encouraging since they allow the proposed approach to be profitably used in real environments.

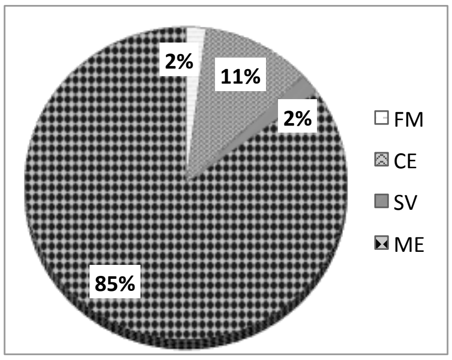


Figure 6.17 The average time needed to execute the proposed algorithm, expressed in terms of percentage of the different modules with respect to the total time.

Typology	Method		Accuracy	False Positive
Single Expert	**CE**	[28]	83.87 %	29.41 %
	ME	Proposed	71.43 %	53.33 %
	SV		53.57 %	66.67 %
MES	**CE + SV**		88.29 %	13.33 %
	CE + ME	[2]	92.86 %	13.33 %
	CE + ME + SV	Proposed	**93.55 %**	**11.76 %**

Figure 6.18 Comparison of the fire detection algorithm approach with state of the art methodologies in terms of Accuracy and False Positive.

6.3 DataFlow Results

In order to validate the proposed software architecture, in terms of both performance and services provided by the middleware, we have undertaken the porting to the new middleware of the two algorithms explained in the previous chapters, by implementing all the nodes required.

The first application we have ported is the people tracking algorithm [1] explained in Chapter 3. The trajectories extracted by the tracked objects are used to detect events of interest, like entering a forbidden area, transiting on a one-way passage in the wrong direction, abandoning objects and so on.

Fig. 6.19a shows how the application has been decomposed into nodes. Notice that there is a feedback loop needed for the background update; the platform can support cyclic networks like this through the specification (in the NCF file) of one or more initial values to be inserted in the feedback loop.

The second application we have integrated in our platform is a fire detection algorithm [2] shown in Chapter 4. Considering that this algorithm is based on the same background subtraction and updating algorithm of the tracking module, we reuse the same nodes implemented in the previously described tracking algorithm. Fig. 6.19b shows how this algorithm has been decomposed into nodes, while the common nodes shared between tracking and fire detection algorithms have been highlighted in Fig. 6.19c.

We can also note that the parallelism can be achieved even in a single algorithm and this is a single chain (see Fig. 6.19b), as in cases of color, shape and movement analysis that can be parallelized.

Fig. 6.20 shows the NCF file that assembles the blocks into an application for the people tracking algorithm.

Notice that the acquisition block is instantiated using a node that reads frames from a video file, in order to make repeatable experiments; however, by just changing a single line of the NCF, a different source (e.g. a USB camera, or a video stream coming from a network connection) can be used.

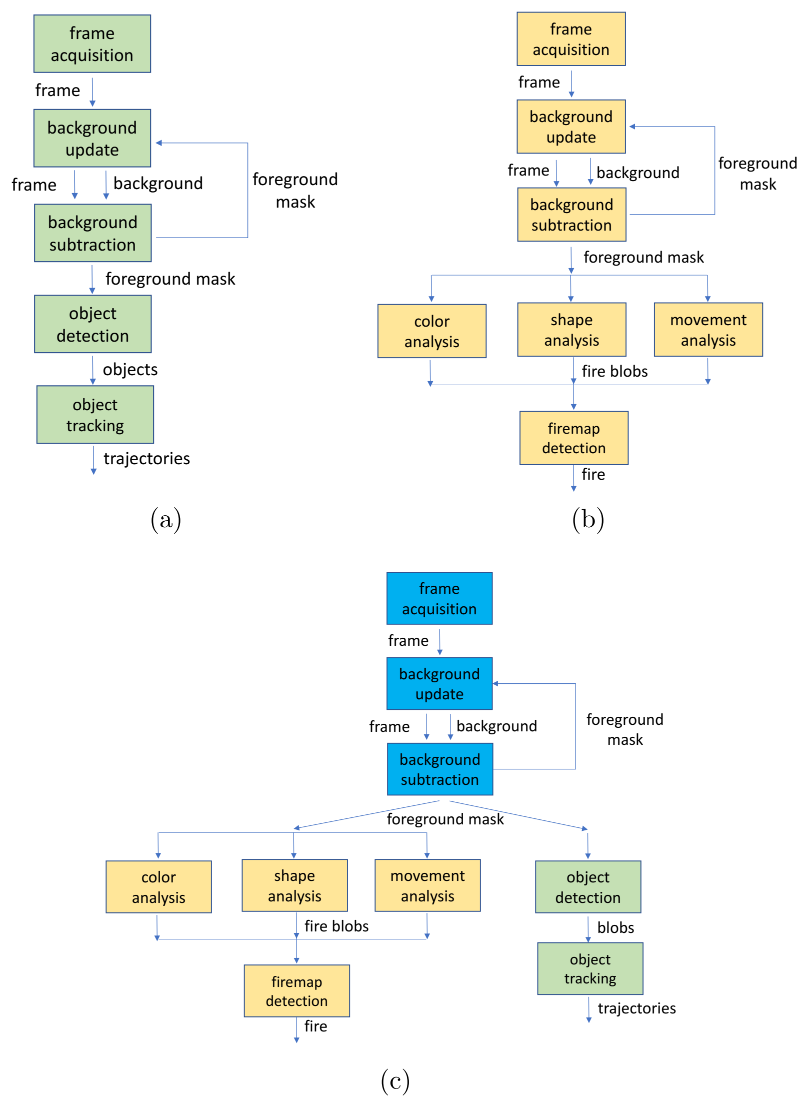


Figure 6.19 Structure of the video-surveillance application used to test the proposed platform. (a) for Tracking Algorithm, (b) for Fire Detection Algorithm and (c) an example of both Fire and Tracking Algorithm on the same NCF file. Notice the is a feedback loop between the blocks *background subtraction* and *background update*; our platform does not require the networks to be acyclic.

```
# Define short names for node types
alias frame_acquisition = acquisition.from_video_file version 1.0.0
      bg_update = background_update.update version 1.0.0
      bg_subtraction = background_subtraction.subtract version 1.0.0
      obj_detection = object_detection.detect version 1.0.0
      obj_tracking = object_tracking.track version 1.0.0
      evt_detection = event_detection.detect version 1.0.0

# Create and configure node instances
node fr_acq : frame_acquisition
end

node bg_update : bg_update
   alpha=0.01  # Blending coefficient
end

node bg_subtraction : bg_subtraction
   threshold = 40  # difference threshold
end

node obj_detection : obj_detection
end

node obj_tracking : obj_tracking
end

node evt_detection: evt_detection
end

# Connect node instances
connect fr_acq.frame to
      bg_update.frame, bg_subtraction.frame
connect bg_subtraction.foreground_mask to
      obj_detection.foreground_mask
connect obj_detection.objects to
      bg_update.objects, obj_tracking.objects
connect obj_tracking.trajectories to
      evt_detection.trajectories

# To handle the feedback loop, the following connection specifies
# an initial value (a black image) to be put in the buffer before
# the execution starts
connect bg_update.background to bg_subtraction.background
        values image of [0 0 0] end
```

Figure 6.20 The NCF file for the video-surveillance application.

Figure 6.21 Examples of images extracted from the videos in PETS 2010 Dataset (a), (b) and MiviaFire Dataset (c), used to test the system.

6.3.1 Datasets

In order to evaluate the porting of the algorithms in the proposed architecture, the creation of the nodes and of the applications via NFC, the outputs of the algorithms with and without the new platform were compared over two standard datasets, the PETS 2010 dataset [75] described in Section 6.1.1 for the tracking algorithm (Fig. 6.21a and 6.21b) and the Mivia Fire dataset [2] for the fire detection algorithm (Fig. 6.21c) described in Section 6.2.1. This analysis confirms that the output of the original algorithms and of the ones ported in the middleware are exactly the same.

6.3.2 Experiments

Once the application has been completed, it has been tested to assess the speedup due to the parallelization, the overhead introduced by the platform and the processing time. Testing has been performed on a system with 4 Xeon cores (and 8 threads) with a 2.13 GHz clock, running a 64 bit Linux kernel. The experimentation has been conducted by varying:

- the number of threads in the set $\{1, 2, 4, 8, 12, 16\}$;
- the number of video streams in the set $\{1, 2, 4, 8, 12, 16\}$;
- the image resolution with a full resolution (4CIF), 1/2, 1/4 and 1/8 of the full resolution.

In order to confirm the effectiveness of the proposed architecture, four experiments showing the relationship between the above parameters have been carried out. All the values reported in the following have been obtained by computing the average times over all the frames of the considered datasets.

Relationship between threads and streams

The relationship between the number of streams and the number of threads is reported in Fig. 6.22; in general, increasing the number of streams, independently on the number of threads, implies as expected an increasing in the processing time required by the platform. However, it is interesting to note that when the number of threads set in the Thread Management module is equal to the number of threads of the hardware (8 in our experiments), the proposed architecture is able to achieve the best possible performance, since each thread of the architecture is allocated on a single thread of the operating system.

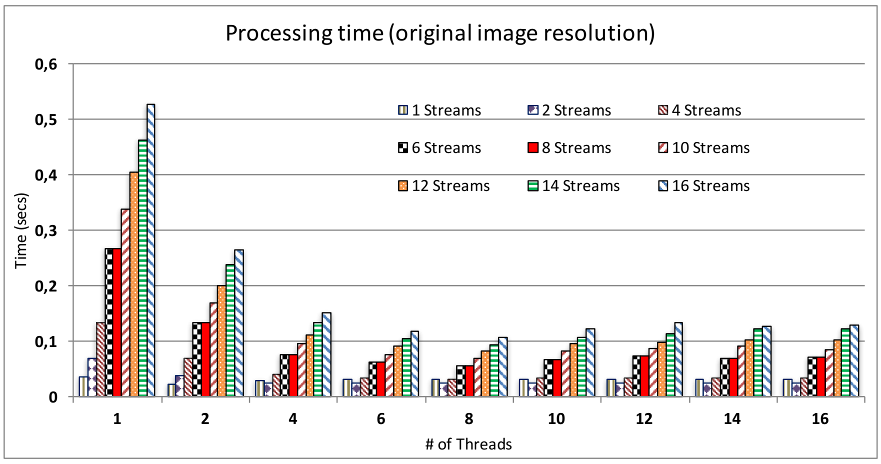


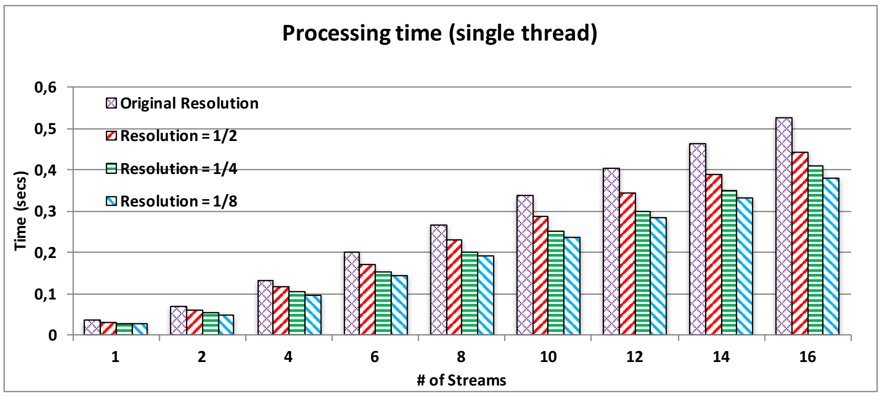
Figure 6.22 Relationship between the number of streams and the number of threads by varying the number of threads in the range 1-16 and the number of video streams in the range 1-16 for fixed video resolution (4CIF).

Processing time with different frame resolutions

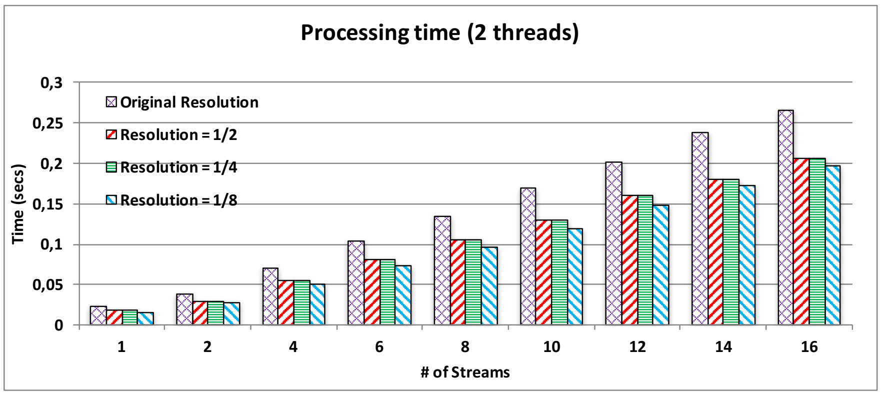
We have computed the time needed to process a single frame by considering different images resolutions and different number of video streams. The obtained results are summarized in Fig. 6.23a for a single thread configured and Fig. 6.23b for two threads: we can note that the time required to process on average a single frame linearly increases by increasing the number of streams (1, 2, 4, 6, 8, 10, 12, 14, 16) as well as by increasing the image resolution (1/8, 1/4, 1/2, 1).

Overhead

We evaluate the computational overhead introduced by the middleware, as the difference between the total usage time of a node and the processing time used by each node just to process its inputs and generate outputs. The results, in terms of time required for elaborating a single frame, are shown in Fig. 6.24: for each image, the number of threads has been fixed in the Thread Management module, while we varied the number of streams. For instance, Fig. 6.24a shows how the time for processing in the average a frame, by using a single thread, linearly increases with respect to the number of streams. In red, the overhead time introduced by the platform is reported. We can note that it is just a very small percentage of the execution time of the algorithms itself, ranging from about 1% for a single thread up to just 5% for 16 threads. It is also interesting to note the results reported in Fig. 6.25, where the absolute overhead introduced by the platform is reported by varying the number of thread from 1 up to 32. As we can see from the figure, this time is not constant but instead it grows due to the presence of the Scheduler Module, which has a growing effort by increasing the number of threads to be managed. However, this time is always below 4 ms per frame even in presence of 32 threads.

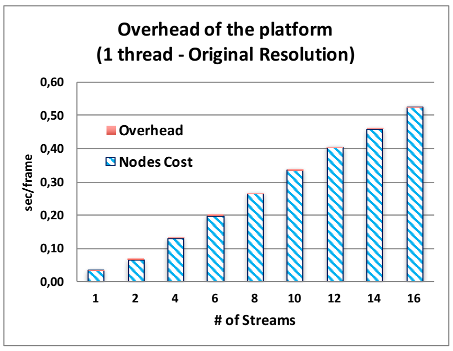


(a)

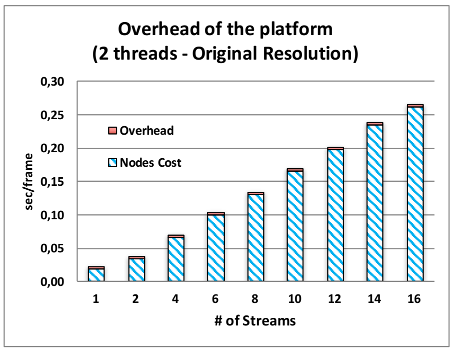


(b)

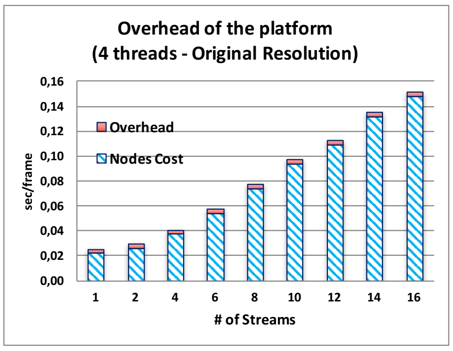
Figure 6.23 Processing time by varying the resolution and the number of video streams for (a) 1 thread and for (b) 2 threads.



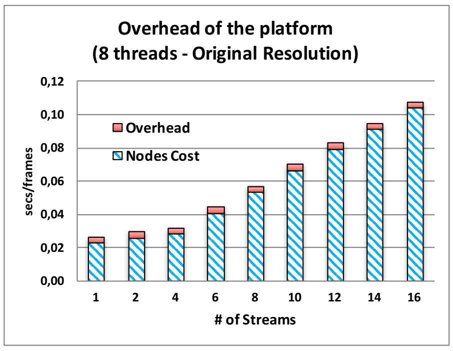
(a)



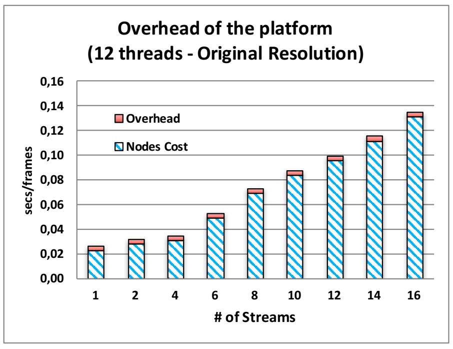
(b)



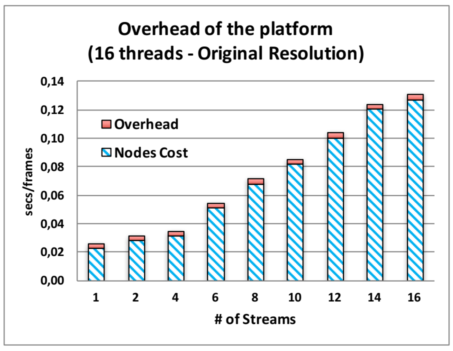
(c)



(d)



(e)



(f)

Figure 6.24 System overhead time and nodes cost time to vary the number of streams for (a) 1 thread, (b) 2 threads, (c) 4 threads, (d) 8 threads, (e) 12 threads, (f) 16 threads, used by the Middleware configured in Thread Management module.

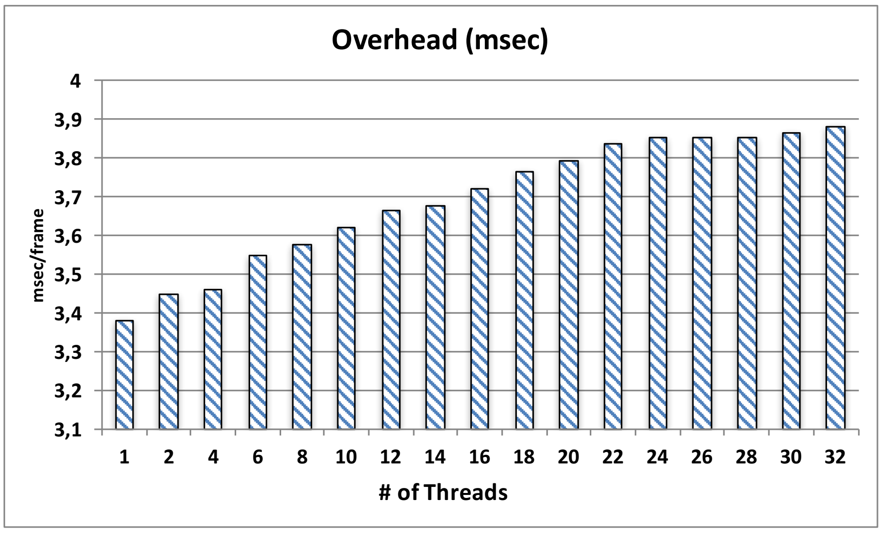


Figure 6.25 Computational overhead measured for a fixed threads number, averaged over all frames calculated from 1 to 16 streams in parallel.

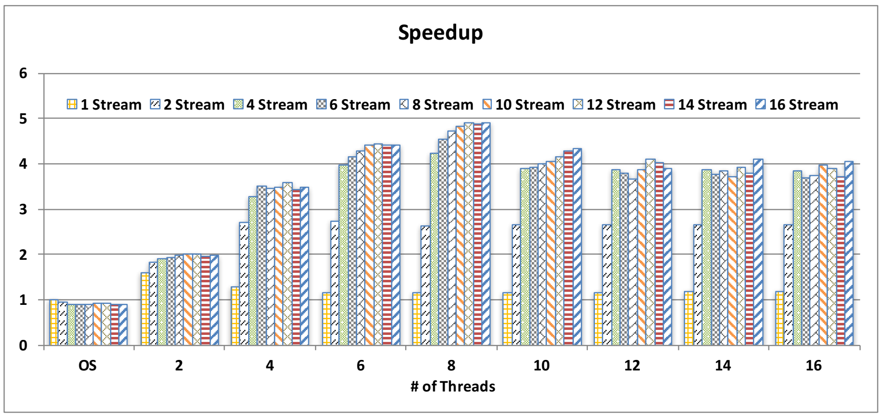


Figure 6.26 Speedup introduced by the architecture.

Speedup

The speedup introduced by the proposed architecture with respect to the time required for the elaboration of the same number of streams by only using the multi-threading approach of the operating system has been reported in Fig. 6.26. Let's focus on the results up to 8 threads, which is the number of threads managed by the used machine. As expected, the speedup linearly incresing with the number of threads, with a maximum incresing exactly with 8 threads. Indeed, using 8 threads, independently on the number of streams, would allow to speedup the proposed architecture of more than 5 times. Furthermore, from the same figure we can also note that the improvement slightly decreases by further increasing the number of threads. However, this improvement is higher than 4 times, thus confirming the effectiveness of the proposed architecture.

Chapter 7

Conclusion

In this thesis we have presented a middleware platform for the development of real-time video-processing applications, and we have described its experimental verification using two different and innovative surveillance applications as a test bed: a people tracking algorithm and a fire detection algorithm.

The novel tracking algorithm used in this thesis is able to overcome many of the problems induced by the object detection phase, as well as to deal with total or partial occlusions. In order to confirm these characteristics, the algorithm has been tested over a standard dataset. The obtained results confirm the effectiveness of the proposed approach in very different environments. Furthermore its low computational cost makes this algorithm well suited for real-time applications for behavior analysis.

The fire detection algorithm we propose in this thesis uses an ensemble of experts based on information about color, shape and flame movements. The approach has been tested on a wide database with the aim of assessing its performance both in terms of sensitivity and specificity. Experimentation confirmed the effectiveness of the approach, which achieves better performance in terms of true positive rate with respect to any of its composing experts. Particularly significant by the applicative point of view is its drastic reduction of false positives, from 29.41% to 11.76%. Even though the system is made of three experts working simultaneously, its overall computational load is compatible with real time video surveillance application.

Once the application porting has been completed, it has been tested to assess the speedup due to the parallelization, the overhead introduced by the platform and the processing time.

To confirm the effectiveness of the proposed data-flow architecture, four experiments showing the relationship between the above parameters have been carried out. The results underline that the middleware is flexible and effective, improving both the development process and the performance of the resulting application.

Bibliography

[1] R. Di Lascio, P. Foggia, G. Percannella, A. Saggese, and M. Vento, “A real time algorithm for people tracking using contextual reasoning,” *Computer Vision and Image Understanding*, 2013. [Online]. Available: http://dx.doi.org/10.1016/j.cviu.2013.04.004

[2] A. Greco, R. D. Lascio, A. Saggese, and M. Vento, “Improving fire detection reliability by a combination of videoanalytics,” in *ICIAR*, 2014.

[3] W. Nie, A. Liu, and Y. Su, “Multiple person tracking by spatiotemporal tracklet association,” in *Proceedings of the 9th AVSS Conference*. Beijing, China: IEEE, Septemer 18-21 2012.

[4] J. Badie, S. Bak, S. Serban, , and F. Bremond, “Recovering people tracking errors using enhanced covariance-based signatures,” in *Proceedings of the 9th AVSS Conference*. Beijing, China: IEEE, Septemer 18-21 2012.

[5] M. Hofmann, M. Haag, and G. Rigoll, “Unified hierarchical multi-object tracking using global data association,” in *Performance Evaluation of Tracking and Surveillance (PETS), 2013 IEEE International Workshop on*, 2013, pp. 22–28.

[6] I. Haritaoglu, D. Harwood, and L. S. David, “W4: Real-time surveillance of people and their activities,” *IEEE Trans. on Pattern Analysis and Machine Intelligence*, vol. 22, no. 8, pp. 809–830, August 2000.

[7] D. Conte, P. Foggia, G. Percannella, and M. Vento, "Performance evaluation of a people tracking system on pets2009 database," in *Proceedings of the 7th IEEE International Conference on AVSS*, 2010, pp. 119–126.

[8] Z. Chen, T. Ellis, and S. A. Velastin, "Vehicle detection, tracking and classification in urban traffic," in *Proceedings of the 15th International IEEE Conference on Intelligent Transportation Systems.* IEEE, 2012.

[9] Z. Jiang, D. Q. Huynh, W. Moran, and S. Challa, "Tracking pedestrians using smoothed colour histograms in an interacting multiple model framework." in *ICIP*, B. Macq and P. Schelkens, Eds. IEEE, 2011, pp. 2313–2316.

[10] C. Dai, Y. Zheng, and X. Li, "Pedestrian detection and tracking in infrared imagery using shape and appearance," *Computer Vision and Image Understanding*, vol. 106, no. 23, pp. 288 – 299, 2007. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1077314206001925

[11] T.-L. L. Hwann-Tzong Chen, Horng-Horng Lin, "Multi-object tracking using dynamical graph matching," in *Proc. of the IEEE Conf. on Computer Vision and Pattern Recognition*, vol. 2, 2001, pp. 210–217.

[12] J. Zhang, L. L. Presti, and S. Sclaroff, "Online multi-person tracking by tracker hierarchy," in *Proceedings of the 9th AVSS Conference.* Beijing, China: IEEE, Septemer 18-21 2012.

[13] T. Xu, P. Peng, X. Fang, C. Su, Y. Wang, Y. Tian, W. Zeng, and T. Huang, "Single and multiple view detection, tracking and video analysis in crowded environments," in *Proceedings of the 9th AVSS Conference.* Beijing, China: IEEE, Septemer 18-21 2012.

[14] S. Pellegrini, A. Ess, K. Schindler, and L. van Gool, "You'll never walk alone: Modeling social behavior for multi-target

tracking," in *Computer Vision, 2009 IEEE 12th International Conference on*, 29 2009-oct. 2 2009, pp. 261 –268.

[15] Q. Delamarre and O. Faugeras, "3d articulated models and multiview tracking with physical forces," *Computer Vision and Image Understanding*, vol. 81, pp. 328–357, March 2001. [Online]. Available: http://dl.acm.org/citation.cfm?id=376890.376922

[16] J. Berclaz, F. Fleuret, E. Turetken, and P. Fua, "Multiple object tracking using k-shortest paths optimization," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 33, no. 9, pp. 1806 –1819, sept. 2011.

[17] D.-T. Lin and K.-Y. Huang, "Collaborative pedestrian tracking and data fusion with multiple cameras," *Information Forensics and Security, IEEE Transactions on*, vol. 6, no. 4, pp. 1432 –1444, dec. 2011.

[18] K. Bhuvaneswari and H. Abdul Rauf, "Edgelet based human detection and tracking by combined segmentation and soft decision," in *Control, Automation, Communication and Energy Conservation, 2009. INCACEC 2009. 2009 International Conference on*, june 2009, pp. 1 –6.

[19] Z. Han, Q. Ye, and J. Jiao, "Combined feature evaluation for adaptive visual object tracking," *Computer Vision and Image Understanding*, vol. 115, no. 1, pp. 69 – 80, 2011.

[20] X. Song, J. Cui, H. Zha, and H. Zhao, "Vision-based multiple interacting targets tracking via on-line supervised learning," in *Proceedings of the 10th European Conference on Computer Vision: Part III*, ser. ECCV '08. Berlin, Heidelberg: Springer-Verlag, 2008, pp. 642–655.

[21] M. Wang, H. Qiao, and B. Zhang, "A new algorithm for robust pedestrian tracking based on manifold learning and feature selection," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 12, no. 4, pp. 1195–1208, 2011.

[22] A. E. Cetin, K. Dimitropoulos, B. Gouverneur, N. Grammalidis, O. Gunay, Y. H. Habiboglu, B. U. Toreyin, and S. Verstockt, "Video fire detection: a review," *Digital Signal Processing*, vol. 23, no. 6, pp. 1827 – 1843, 2013.

[23] Z. Xiong, R. Caballero, H. Wang, A. Finn, and P.-y. Peng, "Video fire detection: Techniques and applications in the fire industry," in *Multimedia Content Analysis*, ser. Signals and Communication Technology, A. Divakaran, Ed. Springer US, 2009, pp. 1–13.

[24] T. Celik, H. Demirel, H. Ozkaramanli, and M. Uyguroglu, "Fire detection using statistical color model in video sequences," *J. Vis. Comun. Image Represent.*, vol. 18, no. 2, pp. 176–185, Apr. 2007.

[25] H.-Y. J. Yoon-Ho Kim, Alla Kim, "Rgb color model based the fire detection algorithm in video sequences on wireless sensor network," *International Journal of Distributed Sensor Networks*, 2014.

[26] C. Yu, Z. Mei, and X. Zhang, "A real-time video fire flame and smoke detection algorithm," *Procedia Engineering*, vol. 62, no. 0, pp. 891 – 898, 2013, 9th Asia-Oceania Symposium on Fire Science and Technology.

[27] X. Qi and J. Ebert, "A computer vision-based method for fire detection in color videos," *International Journal of Imaging*, vol. 2, no. 9 S, pp. 22–34, 2009.

[28] T. Celik and H. Demirel, "Fire detection in video sequences using a generic color model," *Fire Safety Journal*, vol. 44, no. 2, pp. 147–158, 2009.

[29] T. Celik, H. Ozkaramanli, and H. Demirel, "Fire pixel classification using fuzzy logic and statistical color model," in *ICASSP*, vol. 1, April 2007, pp. I–1205–I–1208.

[30] B. C. Ko, K.-H. Cheong, and J.-Y. Nam, “Fire detection based on vision sensor and support vector machines,” *Fire Safety Journal*, vol. 44, no. 3, pp. 322 – 329, 2009.

[31] M. Mueller, P. Karasev, I. Kolesov, and A. Tannenbaum, “Optical flow estimation for flame detection in videos,” *IEEE Trans. Image Process*, vol. 22, no. 7, pp. 2786–2797, July 2013.

[32] A. Rahman and M. Murshed, “Detection of multiple dynamic textures using feature space mapping,” *IEEE Trans. Circuits Syst. Video Technol.*, vol. 19, no. 5, pp. 766–771, May 2009.

[33] B. C. Ko, S. J. Ham, and J.-Y. Nam, “Modeling and formalization of fuzzy finite automata for detection of irregular fire flames,” *IEEE Trans. Circuits Syst. Video Technol.*, vol. 21, no. 12, pp. 1903–1912, Dec 2011.

[34] B. U. Töreyin, Y. Dedeoğlu, U. Güdükbay, and A. E. Çetin, “Computer vision based method for real-time fire and flame detection,” *Pattern Recogn. Lett.*, vol. 27, no. 1, pp. 49–58, Jan. 2006.

[35] P. Borges and E. Izquierdo, “A probabilistic approach for vision-based fire detection in videos,” *IEEE Trans. Circuits Syst. Video Technol.*, vol. 20, no. 5, pp. 721–731, May 2010.

[36] A. Rafiee, R. Tavakoli, R. Dianat, S. Abbaspour, and M. Jamshidi, “Fire and smoke detection using wavelet analysis and disorder characteristics,” in *ICCRD*, vol. 3, March 2011, pp. 262–265.

[37] Y. Habiboglu, O. Gunay, and A. Cetin, “Covariance matrix-based fire and flame detection method in video,” *Mach. Vision Appl.*, vol. 23, no. 6, pp. 1103–1113, November 2012.

[38] O. Gunay, K. Tasdemir, B. Ugur Toreyin, and A. Cetin, “Fire detection in video using lms based active learning,” *Fire Technology*, vol. 46, no. 3, pp. 551–577, 2010.

[39] C. Lindblad and D. Tennenhouse, “The VuSystem: a programming system for compute-intensive multimedia,” *IEEE Journal on Selected Areas in Communications*, vol. 14, no. 7, pp. 1298–1313, 1996.

[40] A. François and G. Medioni, “A modular software architecture for real-time video processing,” in *Computer Vision Systems*, ser. Lecture Notes in Computer Science, B. Schiele and G. Sagerer, Eds. Springer Berlin / Heidelberg, 2001, vol. 2095, pp. 35–49.

[41] A. R. J. François, “Software architecture for computer vision: Beyond pipes and filters,” in *Emerging Topics in Computer Vision.* Prentice Hall, 2003, pp. 585–654.

[42] ——, “A hybrid architectural style for distributed parallel processing of generic data streams,” in *Proceedings of the 26th International Conference on Software Engineering*, ser. ICSE ’04. Washington, DC, USA: IEEE Computer Society, 2004, pp. 367–376.

[43] A. R. François, “An architectural framework for the design, analysis and implementation of interactive systems,” *The Computer Journal*, vol. 54, no. 7, pp. 1188–1204, 2011.

[44] O.-C. Granmo, “Parallel hypothesis driven video content analysis,” in *Proceedings of the 2004 ACM symposium on Applied computing*, ser. SAC ’04. New York, NY, USA: ACM, 2004, pp. 642–648.

[45] A. Avanzi, F. Brémond, C. Tornieri, and M. Thonnat, “Design and assessment of an intelligent activity monitoring platform,” *EURASIP J. Appl. Signal Process.*, vol. 2005, pp. 2359–2374, January 2005.

[46] B. Georis, F. Brémond, and M. Thonnat, “Real-time control of video surveillance systems with program supervision techniques,” *Mach. Vision Appl.*, vol. 18, pp. 189–205, May 2007.

[47] X. Desurmont, A. Bastide, C. Chaudy, C. Parisot, J. Delaigle, and B. Macq, "Image analysis architectures and techniques for intelligent surveillance systems," *IEE Proc. Vision, Image and Signal Processing*, vol. 152, no. 2, pp. 224–231, 2005.

[48] P. Thoren, *Phission: A concurrent vision processing system software development kit for mobile robots.* University of Massachusetts Lowell, 2007.

[49] T. Räty, "High-level architecture for a single location surveillance point," in *Wireless and Mobile Communications, 2007. ICWMC '07. Third International Conference on*, march 2007, p. 82.

[50] X. Amatriain, "A domain-specific metamodel for multimedia processing systems," *Multimedia, IEEE Transactions on*, vol. 9, no. 6, pp. 1284 –1298, oct. 2007.

[51] X. Amatriain, P. Arumi, and D. Garcia, "A framework for efficient and rapid development of cross-platform audio applications," *Multimedia Systems*, vol. 14, pp. 15–32, 2008.

[52] H. Detmold, A. van den Hengel, A. Dick, K. Falkner, D. S. Munro, and R. Morrison, "Middleware for distributed video surveillance," *IEEE Distributed Systems Online*, vol. 9, no. 2, 2008.

[53] M. Saini, M. Kankanhalli, and R. Jain, "A flexible surveillance system architecture," in *Advanced Video and Signal Based Surveillance, 2009. AVSS '09. Sixth IEEE International Conference on*, sept. 2009, pp. 571 –576.

[54] M. Valera, S. Velastin, A. Ellis, and J. Ferryman, "Communication mechanisms and middleware for distributed video surveillance," *Circuits and Systems for Video Technology, IEEE Transactions on*, vol. 21, no. 12, pp. 1795 –1809, dec. 2011.

[55] M. Michalska, E. Bezati, S. Casale-Brunet, and M. Mattavelli, "A partition scheduler model for dynamic dataflow programs," *ICCS 2016. The International Conference on Computational Science*, vol. 80, pp. 2287–2291, 2016.

[56] N. Dalal and B. Triggs, "Histograms of oriented gradients for human detection," in *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on*, vol. 1, june 2005, pp. 886 –893 vol. 1.

[57] M. Fredman and R. Tarjan, "Fibonacci heaps and their uses in improved network optimization algorithms," *Foundations of Computer Science, IEEE Annual Symposium on*, vol. 0, pp. 338–346, 1984.

[58] V. N. Vapnik, "An overview of statistical learning theory," *Trans. Neur. Netw.*, vol. 10, no. 5, pp. 988–999, Sep. 1999.

[59] L. Lam and C. Y. Suen, "Optimal combinations of pattern classifiers," *Pattern Recogn. Letters*, vol. 16, no. 9, pp. 945 – 954, 1995.

[60] J. Kittler, "Combining classifiers: A theoretical framework," *Pattern Analysis and Applications*, vol. 1, no. 1, pp. 18–27, 1998.

[61] P. Soda, G. Iannello, and M. Vento, "A multiple expert system for classifying fluorescent intensity in antinuclear autoantibodies analysis," *Pattern Anal. Appl.*, vol. 12, no. 3, pp. 215–226, Sep. 2009.

[62] M. D. Santo, M. Molinara, F. Tortorella, and M. Vento, "Automatic classification of clustered microcalcifications by a multiple expert system," *Pattern Recogn*, vol. 36, no. 7, pp. 1467 – 1477, 2003.

[63] L.-L. Huang and A. Shimizu, "A multi-expert approach for robust face detection," *Pattern Recogn*, vol. 39, no. 9, pp. 1695 – 1703, 2006.

[64] L. P. Cordella, M. D. Santo, G. Percannella, C. Sansone, and M. Vento, "A multi-expert system for movie segmentation," in *MCS*. London, UK, UK: Springer-Verlag, 2002, pp. 304–313.

[65] D. Conte, P. Foggia, M. Petretta, F. Tufano, and M. Vento, "Meeting the application requirements of intelligent video surveillance systems in moving object detection," in *Pattern Recognition and Image Analysis*, ser. LNCS. Springer Berlin Heidelberg, 2005, vol. 3687, pp. 653–662.

[66] L. He, Y. Chao, and K. Suzuki, "A run-based two-scan labeling algorithm," *IEEE Trans. Image Process*, vol. 17, no. 5, pp. 749–756, May 2008.

[67] S. Tulyakov, S. Jaeger, V. Govindaraju, and D. S. Doermann, "Review of classifier combination methods." in *Machine Learning in Document Analysis and Recognition*, ser. Studies in Computational Intelligence. Springer, 2008, vol. 90, pp. 361–386.

[68] T.-H. Chen, P.-H. Wu, and Y.-C. Chiou, "An early fire-detection method based on image processing," in *ICIP*, vol. 3, Oct 2004, pp. 1707–1710 Vol. 3.

[69] C. Poynton, *Digital Video and HDTV Algorithms and Interfaces*, 1st ed. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2003.

[70] T. Joachims, "Text categorization with suport vector machines: Learning with many relevant features," in *ECML*. Springer-Verlag, 1998, pp. 137–142.

[71] D. G. Lowe, "Distinctive image features from scale-invariant keypoints," *Int. J. Comput. Vision*, vol. 60, no. 2, pp. 91–110, Nov. 2004.

[72] E. Rosten and T. Drummond, "Machine learning for high-speed corner detection," in *ECCV*, 2006, pp. 430–443.

[73] M. Aldinucci, M. Torquati, and M. Meneghin, “Fastflow: Efficient parallel streaming applications on multi-core,” Università di Pisa – Dip. di Informatica, Tech. Rep. TR-09-012, 2009. [Online]. Available: http://arxiv.org/abs/0909.1187

[74] G. Bradski, “The OpenCV Library,” *Dr. Dobb’s Journal of Software Tools*, 2000. [Online]. Available: http://opencv.willowgarage.com

[75] J. Ferryman and A. Ellis, “Pets2010: Dataset and challenge,” in *Advanced Video and Signal Based Surveillance (AVSS), 2010 Seventh IEEE International Conference on*, 2010, pp. 143–150.

[76] R. Kasturi, D. Goldgof, P. Soundararajan, V. Manohar, J. Garofolo, R. Bowers, M. Boonstra, V. Korzhova, and J. Zhang, “Framework for performance evaluation of face, text, and vehicle detection and tracking in video: Data, metrics, and protocol,” *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 31, no. 2, pp. 319–336, 2009.

[77] P. Foggia, G. Percannella, A. Saggese, and M. Vento, “Real-time tracking of single people and groups simultaneously by contextual graph-based reasoning dealing complex occlusions,” in *Proceedings of the IEEE International Workshop on Performance Evaluation of Tracking and Surveillance (PETS)*. IEEE, 2013.

[78] J. Ferryman and A.-L. Ellis, “Performance evaluation of crowd image analysis using the {PETS2009} dataset,” *Pattern Recognition Letters*, no. 0, pp. –, 2014. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0167865514000191

[79] M. Breitenstein, F. Reichlin, B. Leibe, E. Koller-Meier, and L. Van Gool, “Online multiperson tracking-by-detection from a single, uncalibrated camera,” *Pattern Analysis and Machine*

Intelligence, IEEE Transactions on, vol. 33, no. 9, pp. 1820 –1833, sept. 2011.

[80] A. Ellis and J. Ferryman, "PETS2010 and PETS2009 evaluation of results using individual ground truthed single views," in *IEEE Int. Conf. on Advanced Video and Signal Based Surveillance*, 2010, pp. 135–142.

[81] A. Alahi, L. Jacques, Y. Boursier, and P. Vandergheynst, "Sparsity-driven people localization algorithm: Evaluation in crowded scenes environments," in *Performance Evaluation of Tracking and Surveillance (PETS-Winter), 2009 Twelfth IEEE International Workshop on*, dec. 2009, pp. 1 –8.

[82] R. Eshel and Y. Moses, "Homography based multiple camera detection and tracking of people in a dense crowd," in *Computer Vision and Pattern Recognition, 2008. CVPR 2008. IEEE Conference on*, june 2008, pp. 1 –8.

[83] E. Cetin. (2014, July) Computer vision based fire detection dataset. [Online]. Available: http://signal.ee.bilkent.edu.tr/VisiFire/

[84] (2014, July) Ultimate chase. [Online]. Available: http://ultimatechase.com/