

# Video Streaming for Mobile Video Surveillance

Giovanni Gualdi, Andrea Prati, *Member, IEEE*, and Rita Cucchiara, *Member, IEEE*

**Abstract**—Mobile video surveillance represents a new paradigm that encompasses, on the one side, ubiquitous video acquisition and, on the other side, ubiquitous video processing and viewing, addressing both computer-based and human-based surveillance. To this aim, systems must provide efficient video streaming with low latency and low frame skipping, even over limited bandwidth networks. This work presents *MoSES* (MOBILE Streaming for vidEo Surveillance), an effective system for mobile video surveillance for both PC and PDA clients; it relies over H.264/AVC video coding and GPRS/EDGE-GPRS network. Adaptive control algorithms are employed to achieve the best tradeoff between low latency and good video fluidity. *MoSES* provides a good-quality video streaming that is used as input to computer-based video surveillance applications for people segmentation and tracking. In this paper new and general-purpose methodologies for streaming performance evaluation are also proposed and used to compare *MoSES* with existing solutions in terms of different parameters (latency, image quality, video fluidity, and frame losses), as well as in terms of performance in people segmentation and tracking.

**Index Terms**—H.264 coding, mobile video surveillance, video streaming.

## I. INTRODUCTION

**T**HANKS to the spread of both mobile devices and wireless network accessibility, Ubiquitous Multimedia Access (UMA) has become a very common topic within the multimedia community during the last few years. Research centers and telecom providers address new, smart and efficient solutions for the ubiquitous access to multimedia data and in particular videos, from everywhere with mobile devices (laptops, PDAs or last generation cellular phones). Possible applications of such technology include consumer entertainment and digital TV broadcasting, video conferencing, telemedicine and telemanipulation, military applications, and remote video surveillance. All these applications share several technological challenges. On the one side, videos pose serious problems in terms of both amount of data transferred on the network and computational resources. On the other side, mobile devices and UMA scenario require accessibility through different and often limited wireless networks, either 802.11 WiFi, 3G networks such as HSPA (High Speed Packet Access) and UMTS (Universal Mobile Telecommunications Service), or even 2/2.5G networks

such as GPRS (General Packet Radio Service) or EDGE-GPRS (Enhanced Data rates for GSM Evolution). These conflictual requirements—high data volumes and limited resources—emphasize the need for efficient codecs for both downloading and streaming applications. In the case of video data, the goal is to allow UMA services to maintain a sufficient quality for human users. Nevertheless, in some emerging applications the data quality and the compression factors must be evaluated against the more restrictive requirements of “non-human users”, i.e., software that processes and interprets the received data.

This paper will focus on these applications and in particular on *mobile video surveillance*: with this term we refer to the broad class of emerging real-time video surveillance applications where the computational load is not completely in charge of a fixed platform directly connected to the camera. In mobile video surveillance all the issues related with video grabbing, processing, interpretation and dispatching of multimedia data become more challenging due to the presence of mobile platforms, either in the transmitting or the receiving side, wirelessly interconnected.

The meaning of the term “mobile” is quite hazy and might assume different meanings, depending on the context: for example it could be an installation not constrained to remain in a fixed location, a moving device, a portable device (such as handhelds and laptops), or finally a battery-powered device. However, in multimedia the term “mobile” is generally related to the connectivity. Accordingly, here we assume that the reference mobile video surveillance system is provided with an ubiquitous wireless connectivity (either on the server, on the client or on both). Conversely the term “fixed” will be used to consider systems with wired connectivity.

A typical example of architecture for mobile video surveillance is the three-layer architecture of Fig. 1. The first layer (transmitting side or *encoder*) is devoted to encode the video provided by either a live source (camera) or a stored repository of videos. The encoded video is sent to the second layer (receiving side or *decoder*) through a wireless radio channel that provides the widest possible coverage. The received video, once decoded, is processed by the third layer (*video surveillance system*); it can be a traditional *human-based* system consisting of human operators analyzing the videos or a *computer-based* system with automatic processing that extracts moving objects and recognizes interesting events.

This is not the only possible architecture for mobile video surveillance. The growth of smart cameras makes more feasible and interesting the shifting of some processing tasks on-board on the local encoder side. In general, part of the computer-based video surveillance algorithms could be implemented on the first layer with the twofold advantage of reducing the transmission bandwidth and working with uncompressed images. However, our focus will be kept on a first layer that performs video encoding and streaming only, demanding any surveillance tasks to

Manuscript received November 19, 2007; revised March 27, 2008. Current version published October 24, 2008. The work was supported in part by Project FREE SURF of the Italian MIUR Ministry (2007–2008). The associate editor coordinating the review of this manuscript and approving it for publication was Prof. Marco Rocchetti.

G. Gualdi and R. Cucchiara are with the Dipartimento di Ingegneria dell’Informazione, University of Modena and Reggio Emilia, 41100 Modena, Italy (e-mail: giovanni.gualdi@unimore.it; rita.cucchiara@unimore.it).

A. Prati is with the Dipartimento di Scienze e Metodi dell’Ingegneria, University of Modena and Reggio Emilia, 42100 Reggio Emilia, Italy (e-mail: andrea.prati@unimore.it).

Color versions of one or more of the figures in this paper are available online at <http://ieeexplore.ieee.org>.

Digital Object Identifier 10.1109/TMM.2008.2001378

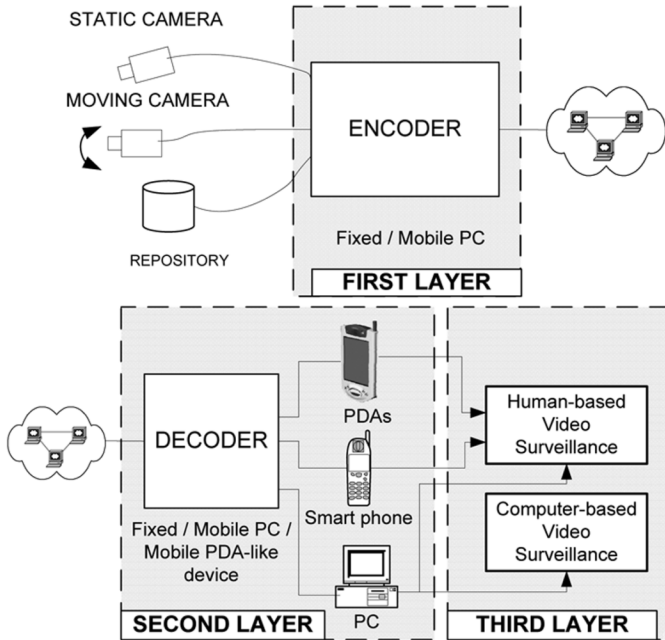


Fig. 1. Generic architecture for mobile video surveillance.

the further steps: this solution remains the most flexible since no specific assumptions are made on the surveillance applications implemented on the third layer. In our work the first layer will be embodied by a standard PC architecture, being aware that standard video encoders have often embedded hardware implementation.

Despite the general architecture, we will focus on the most challenging case in which the video source is provided by a live camera and acquired on demand. There are many examples of interesting applications; in the case of mobile-to-fixed scenarios: a video surveillance system with PTZ moving network cameras over wireless communication, a camera mounted on a police car patrolling a city area, a robot equipped with a camera for monitoring disaster areas, a security camera used in a construction site that moves over on daily basis; any of these might stream the live videos to a control center; vice versa the case of fixed-to-mobile could be a police officer viewing on his PDA the video collected by a moving/static camera installed inside a building (or by a camera mounted on a moving vehicle for the mobile-to-mobile case).

In this work we propose a streaming system, called *MoSES* (MOBILE Streaming for vidEo Surveillance), that effectively implements the described general purpose architecture for mobile video surveillance. MoSES supports video streaming in different conditions, aiming at low-latency transmission over limited-bandwidth network. Additionally, the video stream is provided with a sufficient quality to be correctly analyzed by both human-based or computer-based video surveillance layers. To this aim we propose an optimization of the streaming process with an adaptive control of the streaming parameters.

MoSES is built upon open-source software components. The reason is twofold. First, the availability of the complete source code permits modifications and optimization at any level. Second, most of open-source software or components

TABLE I  
REQUIREMENTS FOR THE SYSTEM. “√” AND “-” INDICATE “REQUIRED” AND “NOT REQUIRED,” RESPECTIVELY

#	Requirement	Video Surveillance	
		Human-based	Computer-based
1	Ubiquitous accessibility	√	√
2	Low latency	√	√
3	High image quality	preferable	√
4	Video fluidity	preferable	-
5	No frame skipping / loss	-	√
6	High frame rate	-	preferable

can be cross-compiled on different architectures and/or operating systems with just specific adjustments: cross-architecture software is required since the client side of MoSES is meant to work not only on PCs but also on PDAs.

Evaluating real-time video streaming is not simple neither clearly defined. In this paper we propose a new methodology with an effective image analysis step to provide comparative performance evaluation over four key parameters, namely latency, image quality, video fluidity and frame losses. Eventually, we compare results achieved with MoSES and other streaming systems over such parameters and also over moving object segmentation and tracking for mobile video surveillance.

The rest of the paper is structured as follows. In the next section we define the system requirements for effective human-based and computer-based video surveillance. Then, in Section III, we review some commercial and scientific approaches to video streaming and related works on mobile video surveillance. Section IV presents the full details of MoSES. Experimental results (Section VI), were gathered using the methodologies described in Section VII. Conclusions are drawn in Section VII.

## II. SYSTEM REQUIREMENTS

Mobile video surveillance calls for live video streaming in order to dispatch an online view of the controlled scene. Beyond this basic feature, there are other requirements (listed in Table I) that need to be considered in order to define a successful mobile video surveillance system.

Given the requirement #1 the video coding must be sufficiently adaptable to different wireless network supports and not only to the ones with large bandwidth (such as WiFi or UMTS/HSPA), that do not still offer ubiquitous coverage. In this work GPRS/EDGE-GPRS network has been selected as reference mobile data service, since it provides wide coverage over the European territory. This is merely an implementation choice that does not compromise the generality of the system which can provide video streaming over any IP-based network. Given the effective bandwidth available for EDGE-GPRS and GPRS connections, MoSES will be tested on 80 and 20 kbps, respectively. Moreover, in mobile video surveillance a multiple delivery of video sources could be requested, therefore tests on 20 and 5 kbps will be performed to simulate a four cameras video delivery. The requirement #2 of *low latency* is necessary because surveillance systems should exhibit high reactivity to changes in the scene. Moreover, mobile video surveillance needs *high image quality* (requirement #3) and *good video fluidity* (requirement #4). Both are preferable for a satisfactory human-based surveillance and the first is mandatory in

computer-based surveillance to allow a correct video analysis and scene recognition. This process is very sensitive to noise and changing conditions and it is, thus, greatly influenced by the coding artifacts or the quantization introduced by the video compression. Consequently, the lower the bandwidth available, the greater the video compression, the more the noise affecting the correct video analysis. The most basic steps of video processing within the considered surveillance context are [1]: segmentation of moving objects, object tracking and object analysis. While the last step is largely dependent on the goal of the application, segmentation and tracking tasks are very general: a performance degradation in segmentation and tracking compromises the whole automatic surveillance system. In the case of a static camera, the segmentation step is often based on background suppression techniques [2], which compare the actual pixel values with the corresponding values in the (statistical) model of the static scene, i.e., the background model. It is evident that the frame compression can significantly affect this step by changing pixel values and making a sophisticated background model useless. For this reason requirement #3 is crucial. The tracking step is typically based on object-to-track association on a frame basis and tracking algorithms usually assume a fixed frame rate for effective status predictions. Therefore the requirement #5 (*no frame skipping*) becomes necessary. Finally, the search area for a tracked object in a new frame is generally proportional to the displacement that the object can have between two consecutive frames: thus requirement #6 (*high frame rate*) prevents an excessively-enlarged search area.

Most of the listed requirements are necessary for the majority of UMA services and the last-generation commercial streaming systems typically fulfill them just in part. Our system MoSES, on the other hand, is specifically designed to fulfill them in full. It is based on H.264/AVC (MPEG-4 part 10) [3]–[5], suitably devoted to work on low-capacity networks by means of several improvements to make the streaming adaptive. H.264/AVC guarantees a better tradeoff between image quality and bandwidth occupation with respect to MPEG-2 and MPEG-4 part 2 [5].

### III. RELATED WORKS

#### A. Video Streaming Solutions and Components

Several video streaming solutions handle all the steps of the process, namely video grabbing, encoding, network streaming and video playback. Two examples of popular off-the-shelf suites are Microsoft Windows Media<sup>®</sup> and Real Networks<sup>®</sup>, based on Helix technology [6]. The encoding layers of such systems are intended to provide streaming server capabilities, i.e., to handle intensive video broadcasts. They require strong processing platforms and/or server-oriented operating systems. For example, Windows Media Streaming Server runs on Windows Server OS only. The main goal of these proprietary solutions is to provide massive access to both live and stored video resources for entertainment purposes, rather than ubiquitous, uni-cast video streaming as required for surveillance purposes. For this reason their latency is usually rather high (as shown in Section VI), being in conflict with requirement #2. Moreover, since the typical users are non-technical practitioners, their settings are often pretty limited mainly in terms

of video coding and network streaming. Regardless of such considerations, their performance in mobile video surveillance working conditions are not negligible. We measured the overall streaming performances of both Windows Media and Real Networks, since they both provide a video player for PC and PDA. Numerical results will be discussed in Section VI.

Skype<sup>®</sup> probably represents the most popular freeware tool for live multimedia streaming, addressing audio and video. The overall performance of this system is very interesting, though it has some limitations to be applied for mobile video surveillance. In particular, the settings flexibility is even coarser than the aforementioned tools since almost everything is automatically handled: for instance, grabbing source, frame size and rate and video bitrate are not adjustable. This approach makes the system very easy to be used for audio/video calls but also very rigid and certainly not flexible enough to be used for our goals. Moreover, according to the methodologies that will be presented in Section V, analytical latency measurement becomes unfeasible in such conditions. Skype is meant for audio streaming, that cannot be disabled in favor of video, producing a bandwidth waste that becomes a critical issue on radio mobile connections. Finally, the video player is currently implemented in the PC version only.

On the side of open-source software, VideoLan Client (VLC) is probably the most renowned tool available. Differently from all previous examples, VLC is designed for research or free use and not for commercial purposes; it provides a very flexible and refined setup, that reaches the lowest level of details for video grabbing, coding and network streaming. It supports many video compression standards (including H.264) and streaming technologies. Anyway the system shows strong limitations that conflict with many of the requirements of our project: as reported in Section IV-B, video latency (requirement #2) can be kept pretty low only at the cost of strong video quality degradation. Regarding bandwidth usage, the H.264 video bitrate control is neither very strict nor optimized. For example, H.264 streaming is allowed only on MPEG TS (Transport Stream) encapsulation: the work in [7] demonstrates that this is a drawback, since using the stack MPEG-TS/UDP/IP introduces more than double the overhead than using the stack RTP/UDP/IP. Finally, the VLC parameter control of the H.264 video coding is not precise since it gives access to the full palette of parameters but cannot handle them correctly: artifacts and strong image quality degradation are introduced as soon as the setup deviates from the standard conditions in order to be optimized for low bandwidths.

An alternative solution is to design and develop an optimized system that specifically targets mobile video surveillance, using existing components where suitable.

The most complex blocks in the video streaming pipeline are video encoding and decoding. Given the choice for H.264 codec, we compared existing encoding engines according to their computational performances and parameters flexibility. Performance is required since the encoding must be performed in real time and conversely H.264 can be computationally very demanding if configured on high encoding profiles; flexibility is needed to tune the encoder for addressing different needs, such as high encoding speed or high image quality.

JM [8] is the H.264 reference software: it is open source, completely flexible and modifiable, but has very limited computational performance. Conversely, Intel Integrated Performance

Primitives (Intel IPP) [9] is a suite of libraries that offers broad digital signal processing algorithms, including also video coding and specifically H.264. Such libraries are optimized but not open source and can be executed on Intel processors only. X.264 [10] is today one of the open source H.264 encoder with best performance and highest completeness over the H.264 standard and for these reasons it was chosen as the foundation block in our first layer of Fig. 1.

Regarding the choice for H.264 decoder, the performance issue becomes very restrictive since we want to address not only standard x86 architectures, but also CPUs with limited computational power, e.g., ARM architectures of PDAs. The same considerations mentioned on the JM and Intel IPP H.264 encoders are valid for their decoder tools. >From the wide variety of other decoders, many being open source, the choice is limited if we consider only those that can be compiled on both PC and PDA. Our selected decoder is based on FFMPEG [11], because of performance and ability to handle most of the H.264 coding profiles; it does not provide a ready-to-use PDA version, but since it is open source it can be appropriately modified for that. This library has the additional advantage to offer a network down-streaming layer implemented for many different protocols.

### B. Streaming for Mobile Video Surveillance

Video streaming has reached its peak of interest in the scientific community in the last ten years. A good survey paper on this topic has been written by Lu in 2000 [12]. It reports and discusses the relevant signal processing issues and proposes possible solutions. Regarding video streaming, researchers have addressed the problem from several different perspectives. For example, one of them addresses the allocation of computational resources for a server with multiple requests [13]. Another set of papers focuses on the system architecture, in terms of both models of communication (as in [14], where the analysis is based on Real Networks products, but without considering low-capacity networks) and data synchronization (as in the case of [15] where an inter-vehicle communication for live video streaming is considered, even though based on 802.11 WiFi networks and thus not suitable for general mobile video surveillance). Some works proposed systems for video streaming over low-capacity networks, such as GPRS. For instance, Lim *et al.* in [16] introduced a PDA-based live video streaming system on GPRS network. The system is based on MPEG-4 compression and contains several computational optimizations for working on a PDA. It can achieve a frame rate of 21 fps at the encoder side and 29 fps at the decoder side for transmitting a QCIF (176 × 144) video at 128 kbps. However, their system drops to 2–3 fps when transmission is over GPRS. Moreover, no information on the latency of the system is provided. The work in [17], instead, solves the problem of transmitting real-time videos over GPRS by using frame skipping. Chuang *et al.* in [18] deals with adaptive playout for video streaming over simulated 3G networks: a statistical model on both departure and arrival processes is built in order to avoid buffer underflows and preserve playout smoothness. Even if the work does not deal clearly with latency measurements and supposes an additional payload for timing data exchange, the idea of adapting video playout to optimize the buffer management will be used also in our work.

Mobile video surveillance has been envisioned in the literature as either classical video streaming with an extension over wireless networks, with no processing at remote side but only remote control by a human operator [19]–[21], or as a special case of distributed wireless sensor networks in which one type of sensors corresponds to video sensors [22]. Moreover, most of these works do not address low-capacity networks. A very seminal work has been published in 1999 by P. Mahonen [23]. This work analyzes the key issues of mobile video surveillance, such as networking requirements and digital image transmission. Moreover, similarly to what has been done by us, the author evaluated the effect of error-prone transmission and coding artifacts on the final video surveillance applications (specifically, intruder alarming and object recognition). However, mainly due to the immaturity of the technology in 1999, no effective proposal to video streaming for mobile video surveillance over low-capacity network is really proposed in the paper. Lam *et al.* presented a very interesting work [1] with a final objective similar to the one of this work. However, in their case frame skipping is functionally employed to fit in the low-bandwidth requirement with an intelligent filtering of frames—bandwidths close to 5 kbps are sustainable only through a very aggressive frame skipping. As aforementioned, this complicates the tracking task; moreover, it requires to move part of the computational load on the local side of the camera.

## IV. SYSTEM PROPOSAL

The basic architecture of MoSES follows the scheme of Fig. 1. In this section each layer will be detailed. Section IV-A describes the video encoder layer (developed for PC-based hardware only). The decoding layer has been designed in two different versions, depending on the computational resources of the client, namely PC-based (Section IV-B) and PDA-based (Section IV-C). The techniques used for computer-based video surveillance are based on SAKBOT (Statistical And Knowledge-Based Object Tracker) [24], which is a system for moving object detection and tracking.

### A. Video Encoder Layer

The typical encoder layers of streaming systems are made of three basic blocks (video grabbing, encoding and network streaming) plus further controlling steps. Our encoder layer aims to provide high flexibility in the control of video source and compression and to keep the latency and the frame-loss rate at lowest levels (Fig. 2). The following peculiar aspects of the architecture were specifically designed to attain such objectives:

- *multithreaded processing and pipeline*: video grabbing, encoding and network streaming are handled by dedicated threads decoupled through circular buffers (Fig. 2). Having asynchronous threads optimizes the processing since the execution of each one is basically independent of the others; this allows the implementation of a pipeline that reduces latency compared to simple serial processing. The original X.264 source code was modified for this purpose;
- *low buffer occupancy*: buffering is necessary to avoid video data losses due to thread decoupling; as drawback, it introduces some latency and for this reason the application

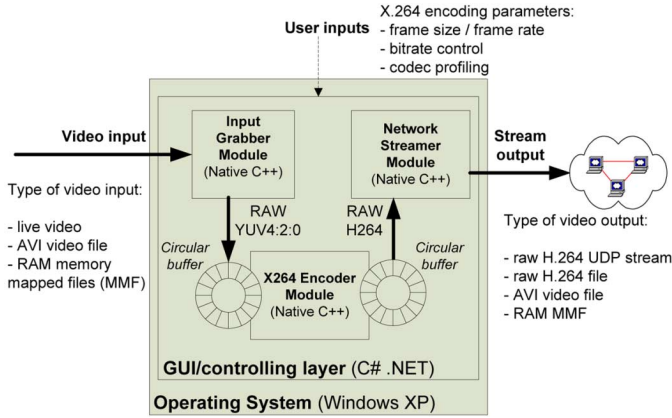


Fig. 2. Functional scheme of the video encoder layer.

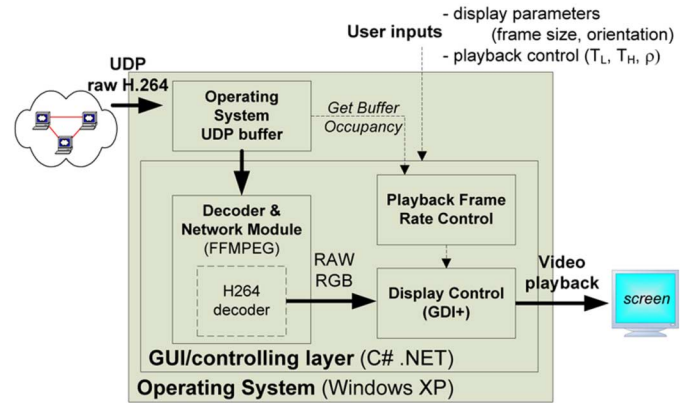


Fig. 3. Functional scheme of the PC video decoder layer.

is tuned to keep buffers at the lowest occupancy. The best way to achieve this is to set the grabbing frame rate equal to (or slightly lower than) the average encoding frame rate; the buffering between encoder and network streamer is not crucial since the second task, being light weighted, manages to keep the buffer always close to zero occupancy;

- *UDP streaming*: raw H.264 encoded stream is seamlessly forwarded to the network streamer that packetizes it into UDP datagrams of fixed byte size. UDP is preferable with respect to TCP due to its prioritized dispatch over the network in case of congestion, but this comes at the cost of possible datagram loss. Nevertheless, thanks also to the Automatic Repeat-reQuest (ARQ) mechanism implemented on the Radio Link Control (RLC) [25], our experiments reported in Section VI-A will show that the transmission over EDGE-GPRS or even GPRS is very robust since the rate of lost datagrams is extremely low and none is received out of order. For this reason, and for the fact that MoSES aims to deliver only video streams with no additional audio or text to be synchronized with, we decided to use raw UDP instead of RTP/UDP.

As shown in Fig. 2, the development was partly based on C#/.NET Framework and partly on native C/C++ modules. In particular, the use of the .NET Framework is devoted only to non-intensive tasks (GUI and controlling layer).

### B. PC Video Decoder

In case of PC-based client, the decoder layer has the functional scheme reported in Fig. 3. The computational demand of H.264 for decoding is definitely lower than encoding, therefore the most critical issue for this PC-based layer is not really the computation optimization rather the efficient network buffering and the data flow management; in fact, even if video grabbing frame rate (encoder side) and playback frame rate (decoder side) are set to the same values, the datagram generation rate (encoder side) and the datagram extraction rate (decoder side) might differ from time to time for several reasons, such as wireless network instability, varying CPU load (either on encoder or decoder side) due to operating system tasks, video coding (changing video scene complexity), and so on. Specifically the following procedures were adopted to minimize latency and preserve the best of the video quality obtained from the network down-streaming:

- *dynamic buffer sizing*: if the datagram generation rate remains higher than the datagram extraction rate for a sufficiently long time, the buffer might fill up and every datagram received afterward would be lost (*buffer overflow*). We propose a simple algorithm that dynamically adapts the buffer size: it either doubles the buffer size when this gets filled up beyond a value  $\chi\%$ , or halves it when its level decreases below  $(100 - \chi)\%$ .  $\chi$  is computed empirically and depends on the network's conditions, buffer initial size and video bitrate;
- *adaptive playback frame rate*: even if the latency time is, for obvious reasons, directly related to the occupancy of the network buffer, tuning the system to keep it as empty as possible would result in uneven trend of the playback frame rate, due to *buffer underflow*. Therefore, to achieve the best tradeoff between low latency and good fluidity, the playback frame rate control (see Fig. 3) implements a set of rules to keep the buffer occupancy between two values  $T_L$  and  $T_H$  (typical values are  $T_L = 5\%$  and  $T_H = 15\%$  of the buffer size). In general, the playback frame rate  $FR_{\text{playback}}^t$  at time  $t$  is function of two values: the buffer occupancy  $B_{\text{occ}}^t$  (that needs to be kept between  $T_L$  and  $T_H$ ) and the discrete derivative of the buffer occupancy  $\Delta B_{\text{occ}}^t$ . The adaptive control can be summarized as follows:

$$FR_{\text{playback}}^t = \begin{cases} FR_{\text{playback}}^{t-1} \cdot (1 + \rho), & \text{if } \Delta B_{\text{occ}}^t > W \\ FR_{\text{playback}}^{t-1} \cdot (1 - \rho), & \text{if } \Delta B_{\text{occ}}^t < -W \\ FR_{\text{playback}}^{t-1}, & \text{if } (B_{\text{occ}}^t, \Delta B_{\text{occ}}^t) \text{ optimal} \\ FR_{\text{playback}}^{t-1} \cdot (1 + \frac{\rho}{W} \Delta B_{\text{occ}}^t), & \text{otherwise} \end{cases} \quad (1)$$

where  $W$  defines a window of  $\Delta B_{\text{occ}}^t$  (typical value is a few thousandths), which represents the limits for a sustainable variation of the buffer occupancy; if the  $|\Delta B_{\text{occ}}^t|$  consistently exceeds  $W$ , the system will end up in buffer overflow or underflow in a short time.  $\rho$  is the reactivity of the adaptive control ( $0 < \rho < 1$  but typical value is approximately a few hundredths). The closer  $\rho$  gets to 0, the weaker the adaptive control is; eventually the control would be disabled with  $\rho = 0$ . The reactivity increases together with  $\rho$ , eventually ending up in an unstable system.

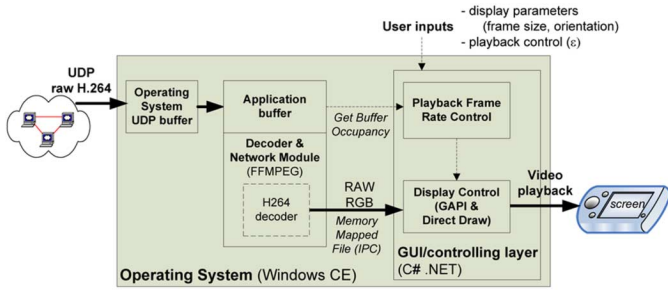


Fig. 4. Functional scheme of the PDA video decoder layer.

The pair of values  $(B_{occ}^t, \Delta B_{occ}^t)$  is considered optimal in the following cases:

$$(\Delta B_{occ}^t, B_{occ}^t) \text{ optimal if : } \begin{cases} T_L < B_{occ}^t < T_H & \wedge & \Delta B_{occ}^t \approx 0 \\ B_{occ}^t > T_H & \wedge & -W \leq \Delta B_{occ}^t \leq 0 \\ B_{occ}^t < T_L & \wedge & 0 \leq \Delta B_{occ}^t \leq W \end{cases} \quad (2)$$

In other words, the control reacts increasing (decreasing) the playback frame rate when the buffer occupancy increases (decreases) too rapidly ( $|\Delta B_{occ}^t| > W$ ). When the conditions are optimal the playback frame rate is kept constant. In all the other cases, the frame rate is slightly adjusted with a factor proportional to the slope of variation of the buffer occupancy;

- *decoder-display coupling*: In the absence of synchronization between the decoder and display threads, it may happen that a frame is correctly decompressed but not displayed, therefore lost, because it is overtaken by a decoded frame which follows (*frame overwriting effect*). Buffering the frames flow would solve the problem but it would also introduce some latency. A different approach, that completely avoids buffering, is to introduce a simple synchronization decoder-display that, just before a frame gets overwritten, delays the decoder (up to a maximum of  $1/2FR_{playback}^t$ ) until the frame is effectively displayed. As shown in Section VI-C, this solution massively reduces frame losses and, even better, has a positive effect on the latency.

### C. PDA Video Decoder

Fig. 4 shows the scheme of the PDA decoder. Differently from the PC version, the successful implementation of the PDA-based decoder requires peculiar optimizations, given the limited computational power of these devices. Specifically, the most critical issues are, together with video decoding and network buffering, video data exchange between processes and video display. The most suitable operating system to rely on in such tight conditions would be Linux for embedded systems, given the important advantage of being open source, thus enabling low-level control on memory, network and management of processes and services. Unfortunately, the limited support for most of the devices and peripherals (such as GSM/GPRS modems) prevented us from adopting this solution, opening the way to Windows CE. Each module and the most important functionalities designed for a successful PDA-based decoder follow:

- *optimized display control*: in the case of PDA-based solution, writing video data directly on the graphic card memory is computationally very convenient rather than relying on the standard functions of the operating system. For this reason the display control is based on GAPI (Game API)<sup>1</sup> and Direct Draw<sup>2</sup> instead of GDI+ functions. For a further speed up of this control, we made use of pre-calculated look-up tables for image rescaling and 90-degrees flipping, used to fit the desired playback frame size and orientation;
- *inter-process communication*: the decoding and the GUI modules are kept completely detached in their scheduling (i.e., they run on two separated processes), so that each processing flow will not interfere with the other (for example, when the GUI module calls the garbage collector). Since a QQVGA ( $160 \times 120$ ) video, 24 bits RGB colors at 10 fps, would generate a 4.6 Mbps bandwidth communication, it is evident that the Inter-Process Communication (IPC) must be extremely efficient to avoid frame skipping and preserve video fluidity. Data exchange through either a UDP local loop-back or file system is not feasible since both these methods could not sustain such a high data transfer rate without compromising the overall performance of the system. Moreover, since file systems are actually based on flash memories which have a finite number of erase-write cycles, IPC based on file system would deteriorate the support in a short time. The most efficient IPC method to be used is then memory-mapped files (MMFs), i.e., a virtual file on RAM memory. This approach allows the achievement of performance comparable to shared memory between threads;
- *adaptive playback frame rate*: the implementation of the adaptive control described in the PC-based decoder needs to be revisited to be successful on a PDA. Since Windows CE does not allow querying of the occupancy of the UDP buffer, an application buffer on top of the UDP operating system buffer must be added (see Fig. 4). In addition, the algorithm presented in Section IV-B can not be successfully implemented on a PDA, because the computational resources can not sustain the required frame rate in case it needs to be firmly increased (usually when  $\Delta B_{occ}^t > W$ ). For this reason the PDA decoder layer employs a light-weighted control based on the key task of keeping the UDP buffer occupancy as low as possible (but greater than zero) in order to minimize the playback interruptions due to buffer underflows. Given a value  $\epsilon$  that is a few thousandths above 1, the control acts as follows:

$$FR_{playback}^t = \begin{cases} FR_{playback}^{t-1} / \epsilon, & \text{if } \Delta B_{occ}^t = 0 \\ FR_{playback}^{t-1} \cdot \epsilon^2, & \text{if } \Delta B_{occ}^t > 0. \end{cases} \quad (3)$$

Being  $\epsilon$  slightly above one, the reaction in the case of buffer occupancy greater than 0 is stronger than in the other case. Analytical results will be given in Section VI-C, where  $\epsilon = 1.002$  was used. This control does not claim to be an optimal algorithm for playback frame rate adaptation (for a deep analysis of adaptive ployout, refer to [18]) but our goal is to verify that even on

<sup>1</sup><http://msdn2.microsoft.com/en-us/library/ms879884.aspx>

<sup>2</sup><http://msdn2.microsoft.com/en-us/library/ms879875.aspx>

reduced-power processors a simple adaptive control can significantly increase the fluidity of the video playback without the need of further buffering.

Regarding the *dynamic buffer sizing* and *decoder-display coupling*, the PDA decoder implements the same procedures described for the PC decoder.

## V. METHODOLOGY OF PERFORMANCE EVALUATION

Evaluating the performance of a live video streaming system is not an easy task, since it is mostly based on the perceived quality of live (not stored) videos by means of the human receiver. We propose a methodology to extract quantitative measures on four aspects in accordance with our requirements: *image quality*, *video latency*, *frame loss* and *video fluidity*. In addition, measures for assessing the performance of the mobile video surveillance system are proposed, as follows.

1) *Image Quality*: It can be easily measured with PSNR (Peak Signal-to-Noise Ratio) that gives an idea of the distortion introduced by the coding and transmission process. Even though this is not completely corresponding to the way our human visual system evaluates the quality, it is an easy and well-known method to measure image quality.

2) *Video Latency*: There are not commonly accepted methods for measuring the *latency* in an analytical way. An approximate measurement could be obtained by synchronizing the encoding and the decoding unit on the same time server, and modifying the encoder so that it dispatches, together with the encoded frames, also the timestamp of their grabbing. Then, the decoding unit detects the time of display of a decompressed frame and deducts the latency by time differencing [26]. This procedure has several drawbacks: on one side the synchronization with the time server should be frequent, in order to have precise time gap measurements; this would be a waste of both CPU cycles and bandwidth; moreover embedding a timestamp for each frame would result in further bandwidth waste. In addition this kind of measurement requires modifications to core functions of the video grabbing, networking and displaying, therefore it is only feasible on open source code and cannot be employed on closed systems such as Windows Media and Real Networks, that we want to compare with.

Consequently, an alternative way to measure the latency must be used. Schmidt in [27] presents an interesting approach to measure the synchronization of synthetic multimedia data (video, audio and text) through external observation of media players using several sensors: we modified and extended the approach for real video data. The frame number is superimposed on each frame of a recorded video. We then play the video both on the encoding unit, and, after having it passed through compression and streaming, also on the decoding unit.

Let us call  $\bar{t}$  a given time instant,  $\text{FN}_{\text{enc}}(\bar{t})$  and  $\text{FN}_{\text{dec}}(\bar{t})$  the frame number shown on the encoder and decoder respectively. Let us define  $\bar{t}^*$  as the time such that  $\text{FN}_{\text{enc}}(\bar{t}^*) = \text{FN}_{\text{dec}}(\bar{t})$ , i.e., the time such that the same frame number is visible on both sides. It holds that  $\bar{t}^* < \bar{t}$ . Let us then call  $\Delta t_{\text{enc}}$  the time gap between two grabbed frames on the encoder, which is constant since the grabbing frame rate is constant; in such conditions a generic time  $t$  can be approximated with  $t = \Delta t_{\text{enc}} \cdot \text{FN}_{\text{enc}}(t)$

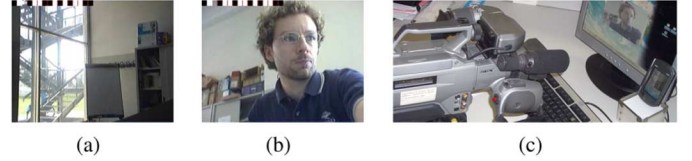


Fig. 5. Experimental methodology used to measure latency. (a) and (b) are two frames of the same video, respectively frame number 1702 and 1718. During a streaming session (see figure (c)) it happens that, due to latency, the encoder side is showing frame (b) while the decoder the frame (a).

and exploiting the definition of latency as  $L(\bar{t}) \equiv \bar{t} - \bar{t}^*$ , we can write it as:

$$\begin{aligned} L(\bar{t}) \equiv \bar{t} - \bar{t}^* &= \Delta t_{\text{enc}} \cdot (\text{FN}_{\text{enc}}(\bar{t}) - \text{FN}_{\text{enc}}(\bar{t}^*)) \\ &= \Delta t_{\text{enc}} \cdot (\text{FN}_{\text{enc}}(\bar{t}) - \text{FN}_{\text{dec}}(\bar{t})). \end{aligned} \quad (4)$$

This procedure needs to embed frame numbers directly on video frames and it could be made automatic with a tool for recognizing the numbers on the images: using plain numbers can be problematic due to the distortion introduced by strong image compression, which could make OCR task unreliable. For this reason, we prefer to adopt a code-based number representation. The binary coded frame number is superimposed on a small portion of each image. More specifically, blocks of white and black color were used to code 1s and 0s, respectively. Fig. 5 shows some snapshots of the methodology. The leftmost-upper two blocks are static and are used for calibration purposes. A video streaming session gets started and the video is played on the screens of both sides (encoder and decoder), which are physically placed one close by the other. At the same time, an external high-frame-rate camera acquires and stores a video of the evolution of the streaming process on both screen (Fig. 5(c)). The resulting video is processed with simple image processing algorithms to automatically compute  $\text{FN}_{\text{enc}}$  and  $\text{FN}_{\text{dec}}$  by recognizing black and white blocks and deduct latency using (4). The tool is very flexible as it can measure the latency also on non open-source systems, e.g., Windows Media and Real Networks.

3) *Frame Loss*: The following equation quantifies the *lost frames*  $\text{LF}(\bar{t})$  exploiting the same frame number coding methodology: given  $\Delta \text{FN}_{\text{dec}}(j) = \text{FN}_{\text{dec}}(j) - \text{FN}_{\text{dec}}(j - \Delta t_{\text{acq}})$ , where  $\Delta t_{\text{acq}}$  is the discrete sampling period of the external camera and  $j$  is a generic frame of its recorded video:

$$\begin{aligned} \text{LF}(\bar{t}) &= \sum_{j=0}^{\bar{t}} \varphi(j) \text{ where} \\ \varphi(j) &= \begin{cases} 0, & \text{if } \Delta \text{FN}_{\text{dec}}(j) \leq 1 \\ \Delta \text{FN}_{\text{dec}}(j) - 1, & \text{otherwise.} \end{cases} \end{aligned} \quad (5)$$

In other words, given that the encoding frame rate is constant and that the frame rate of the external camera is much higher than the encoding and decoding frame rates, if two successive frames grabbed with the acquisition camera contain the same number or successive numbers, no frames were lost.

Frame losses can be due to either network datagram losses (this event would most likely produce a loss of several consecutive frames, since with high compression rates and limited frame size, a datagram usually contains several frames)

or due to compression skipping or decoder frame overwriting. Given the aforementioned high degree of reliability of GPRS and EDGE-GPRS, in the next section we will give evidence of frame losses due to compression skipping and to decoder frame overwriting only, which are definitely predominant on the effects due to network failures.

4) *Video Fluidity*: It can be measured as the trend of the  $\Delta t_{\text{dec}}$ , the time gap between two frames played by the decoder. This information can be computed again exploiting the frame number coding. In the best case,  $\Delta t_{\text{dec}}$  is constant and equal to  $\Delta t_{\text{enc}}$ ; conversely the more the  $\Delta t_{\text{dec}}$  is scattered, the more the video playback loses in fluidity.

5) *Object Detection and Tracking*: The performance of the computer-based video surveillance system will be evaluated in terms of both pixel-level segmentation and object-level tracking performance. Pixel-level segmentation can be evaluated considering the possible loss in segmentation accuracy due to video compression and streaming. As ground truth we used the segmentation generated on the original uncompressed video. In fact, our scope is to evaluate the loss in performance only due to the video compression and streaming and not the generic performance of the segmentation algorithm. By comparison with the ground truth, recall and precision are computed for each frame. Similarly, the object-level tracking evaluation is provided by comparing the tracking results on original and compressed videos. Specifically, the evaluation of the loss in tracking accuracy due only to video compression and streaming is performed as follows: let us call TR a generic track of an object and  $\text{Length}(\text{TR})$  its length in time. In compressed videos, a track can be often split in more sub-tracks with different identifiers. Defining  $\text{TR}_{\text{orig}}^i$  as the  $i$ th track on the original video, and  $\text{TR}_{\text{cmpr}}^{i,j}$  (with  $j = 1, \dots, N$ ) the  $N$  distinct tracks on the compressed video in which the  $\text{TR}_{\text{orig}}^i$  was (mistakenly) split; defining a  $\hat{j}$  for each  $\text{TR}_{\text{orig}}^i$  as:

$$\hat{j} = \arg \max_j \text{Length}(\text{TR}_{\text{cmpr}}^{i,j}) \quad (6)$$

then the accuracy ACC can be measured as follows:

$$\text{ACC} = \frac{\sum_{\forall i} \text{TR}_{\text{cmpr}}^{i,\hat{j}}}{\sum_{\forall i} \text{TR}_{\text{orig}}^i}. \quad (7)$$

## VI. EXPERIMENTAL RESULTS





### A. Test Bed and Operational Conditions

The architecture of Fig. 1 can be divided in two distinctive blocks that can be evaluated in a separate manner: the *video streaming block* (first and second layer), and the *video surveillance block* (third layer), specifically dealing with the computer-based part.

The tests on the *video streaming block* are designed to verify the degree of fulfillment for the requirements listed in Table I. The results are discussed on the basis of two different platforms: the former is PC-to-PC, the latter is PC-to-PDA.

For the PC-to-PC platform, we prepared a mobile-to-fixed system: the mobile site was mounted on a car (camera-car setup) and equipped with a standard x86 laptop (Intel Pentium Centrino 1.7 GHz), connected to either a USB Camera (Logitech Quickcam Pro 4000) or an IP camera (Axis 2420 plus

TABLE II  
STORED VIDEOS USED FOR EVALUATION OF VIDEO STREAMING  
AND COMPUTER-BASED VIDEO SURVEILLANCE BLOCKS

	VLAB	VCAR
		
<b>Evaluation of</b>	video streaming	video streaming
<b>Scenario</b>	Indoor (webcam inside laboratory)	Outdoor (camera-car in urban traffic)
<b>Frame Rate</b>	10 fps	10 fps
<b>Resolution</b>	QVGA (PC-to-PC) QQVGA (PC-to-PDA)	QVGA (PC-to-PC) QQVGA (PC-to-PDA)
<b>Length</b>	≈ 120 s	≈ 600 s
	VHALL	VPARK
		
<b>Evaluation of</b>	video surveillance	video surveillance
<b>Scenario</b>	Indoor (static camera at building hall)	Outdoor (static camera at public park)
<b>Frame Rate</b>	10 fps	10 fps
<b>Resolution</b>	QVGA	QVGA
<b>Length</b>	≈ 420 s	≈ 420 s

infra-red illuminator for night vision), and an EDGE-GPRS modem used for uplink video transmission. For the PC-to-PDA platform, we tested a fixed-to-mobile scenario; some tests have been performed also in a mobile-to-mobile scenario (from our camera-car setup to the PDA). Two different PDAs have been used for the tests (i-Mate JasJar, WM 5.0, CPU: Intel Bulverde, 520 MHz; i-Mate PDA2k, WM 2003, CPU: Intel PXA263, 400 MHz); the results did not show relevant differences by using different PDAs. The radio mobile connectivity was always GPRS-based.

In both hardware configurations, the following operational conditions are used:

- *video encoding*: the H.264 codec has been tested in two different profiles: a *baseline* (to achieve low latency at the cost of low quality) and a *high* profile (for best video quality at the cost of higher latency). The high profile contains several enhancements including the use of CABAC [3] encoding, wider reference window for B frames, deeper analysis for macroblock motion estimation, finer sub-pixel motion estimation and better rate distortion algorithms;
- *video sources*: the design and development of the system was always tailored for live video sources, but the performance measurements were gathered using two stored videos (VLAB, VCAR) in order to replicate the experiments on the same data. Table II shows the main properties of the test videos. Specifically, the video VLAB contains scenes with three different types of motion: reduced (moving people but static camera), medium (freely moving camera) and extreme (shaking camera); the video VCAR is taken from the camera-car setup while driving in a busy urban area.
- *network and bitrates*: when relying on EDGE-GPRS, video bitrate was set to 80 and 20 kbps. A few tests



TABLE III  
LATENCY OF WINDOWS MEDIA, REAL MEDIA, VLC, MOSES. PC-TO-PC  
SCENARIO, VLAB AT QVGA. \*DECODER-DISPLAY COUPLING

	Tool	Profile	Bit rate (kbps)	D-D Cpl*	Playb. frame rate	Avg lat. (sec)	Std dev
1	MOSES	baseline	80	no	fastest	1.26	0.28
2	MOSES	baseline	80	yes	fastest	1.21	0.28
3	MOSES	baseline	80	yes	adaptive	1.55	0.27
4	MOSES	high pr.	80	yes	fastest	1.65	0.38
5	MOSES	baseline	20	yes	fastest	1.41	0.33
6	Win. M.		80			4.76	0.36
7	Real M.		80			3.15	0.07
8	VLC	baseline	80			2.21	0.27

were made saturating the effective EDGE-GPRS bandwidth, at 120 kbps. When using GPRS, video bitrate was set to 20 and 5 kbps, and a few tests were made with 10 kbps. As mentioned in the introduction, 20 kbps on EDGE-GPRS and 5 kbps on GPRS are meant to generate four simultaneous and independent (not spatially multiplexed) video streams. We have experimented the wireless network transmission in several conditions: half tests were performed with the encoder as mobile site, the other half being the decoder side. Half the cases inside a building and in the other half on our camera-car setup (we drove for more than 80 km, at urban and freeway speeds  $-50/110$  km/h). We measured the network transmission on 20000 UDP datagrams, for more than 140 minutes of video streaming over GPRS at 5 and 20 kbps and over EDGE-GPRS at 80 kbps. In these conditions, thanks to the ARQ implemented in the RLC, a very reduced percentage of datagrams was lost (0.54%) and none was received out of order.

The tests on the *computer-based video surveillance block* are designed to evaluate its effectiveness within the context of mobile video surveillance; the analysis is performed using a static camera in order to segment the moving object with the background suppression technique of SAKBOT [24]. Two different surveillance scenarios are considered: an indoor one, taken at the hall of the building of our Department and an outdoor one, taken from a camera mounted in a public park. The first scenario is characterized by few moving people and no illumination changes, while the second is a less-controlled scenario, where several people move in the scene and illumination changes. The second scenario is obviously more challenging for both the video encoder and the computer-based video surveillance system. For performance analysis we recorded two videos, called VHALL and VPARK (Table II).

### B. Experimental Results of PC-to-PC Scenario

Latency measures in PC-to-PC scenario are summarized in Table III. MoSES is configured in five different ways and compared against Windows Media, Real Media and VLC. The latency introduced by MoSES is the lowest, whatever configuration is used. With respect to the base configuration (row #1 of Table III), the introduction of the decoder-display coupling, for frame-overwriting reduction, decreases the average latency from 1.26 s to 1.21 s (row #2). Instead the use of the adaptive frame rate control with  $T_L = 5\%$ ,  $T_H = 15\%$ ,  $W = 0.005$

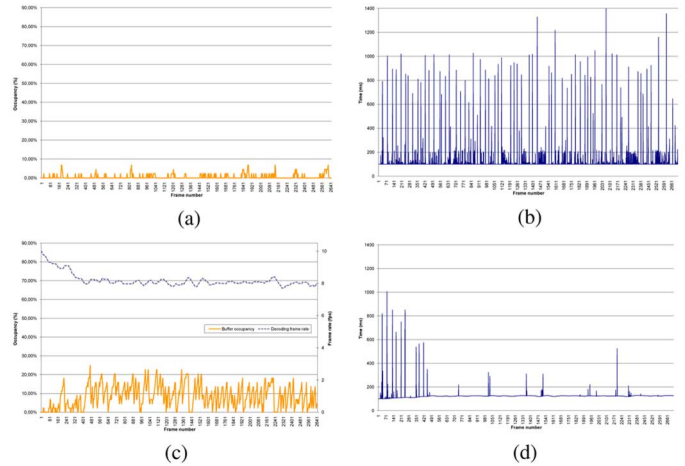


Fig. 6. Buffer occupancy and  $\Delta t_{dec}$ , with MoSES' adaptive control disabled (a), (b) and enabled (c), (d), configured with  $T_L = 5\%$ ,  $T_H = 15\%$ ,  $W = 0.005$  and  $\rho = 0.04$ . (a) Buffer occupancy (%), (b)  $\Delta t_{dec}$ , (c) Buffer occupancy (%) and playback frame rate, (d)  $\delta t_{dec}$ .

and  $\rho = 0.04$  (row #3) has the cost of a slight increase in latency (1.55 s) but greatly improves the video fluidity (see further). The introduction of a high complexity encoding profile (row #4) adds about 0.45 s of latency, that is tolerable considering the gain obtained in image quality. Finally, the reduction of the bitrate (from 80 to 20 kbps) increases the latency (row #5) because the time to fill the UDP datagrams (whose size is kept unchanged for the sake of the test) is longer.

In order to produce a fair comparison, the other three systems have been configured to minimize latency—smallest buffer size and fastest video codec and profiling. In Windows Media, the streaming server layer was removed and the video was streamed directly from the encoder to the player. VLC H.264 was configured with the same baseline profile used in MoSES; since raw UDP streaming is not supported, MPEG-TS/UDP/IP (the only one available on UDP) was used: under these conditions the latency is fairly low (row #8), but this is obtained at the cost of a fully compromised video quality, since about 48% of the video frames were corrupted or lost.

A possible side effect of reducing the latency is to lose video fluidity, having an irregular trend of  $\Delta t_{dec}$ . The graphs in Figs. 6(a) and (b) show the buffer occupancy and the  $\Delta t_{dec}$  respectively, with MoSES configured as in row #2 of Table III: the buffer occupancy is always close to zero, resulting in an almost-ideal latency. However, this set-up generates a frequent increase of the frame decoding time, from the expected 100 ms (due to a  $FR_{playback}$  of 10 fps) up to 1.4 s. These continuous changes in the  $\Delta t_{dec}$  bring to poor video fluidity, affecting both the overall user satisfaction and the understanding of the scene for the human-based video surveillance.

Figs. 6(c) and (d) show the improvements achieved by enabling MoSES' adaptive control in the condition of row #3 of Table III. The trend of the  $\Delta t_{dec}$  demonstrates that the playback is made fluid. As described in Section IV-B, when the buffer occupancy is lower than  $T_L$  (5% in this experiment), the control starts decreasing the decoding frame rate until the buffer occupancy is stable between  $T_L$  and  $T_H$ . Conversely, when the buffer occupancy is higher than  $T_H$ , the control increases the playback frame rate to empty the buffer.

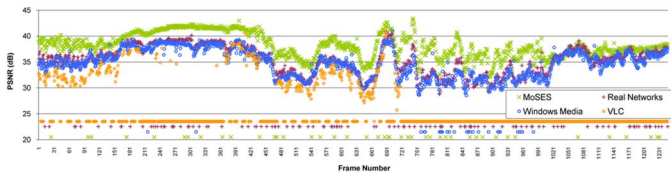


Fig. 7. Comparison of image quality (PSNR) measured on the VLAB video. The video contains scene with static camera (approximately frames 210–450 and 1030–end), moving camera (approx. frames 450–640), shaking camera (the rest). The symbols in the bottom part of the graph represent lost frames.

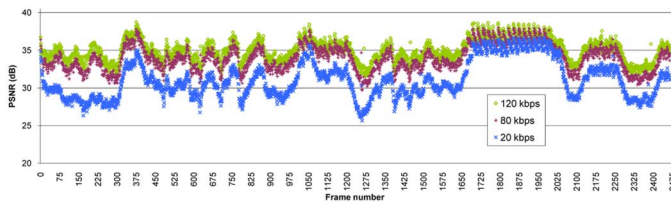


Fig. 8. PSNR measured on the VCAR video using MoSES at 20 kbps (QQVGA), 80 kbps (QVGA), and 120 kbps (QVGA).

Fig. 7 shows image quality and frame losses measurements over the VLAB at 80 kbps. Image quality is evaluated in terms of PSNR and the lost frames are represented with superimposed symbols at the bottom of the same graph. MoSES was configured with high encoding profile (row #4 of Table III). VLC could not work properly with the same high profile due to a massive video frame corruption, therefore it was configured with a slightly lighter profile. Windows Media and Real Media were configured with WMV9 and RMV10 codecs respectively. Table IV summarizes the achieved results: the statistics over the PSNR are computed on the correctly received frames only. MoSES outperforms the others, showing also very few (and fairly distributed) lost frames.

Eventually image quality on MoSES is measured at different bitrates. We used the VCAR video, encoded with baseline profile (results in Fig. 8). The difference in the PSNR along the time is mainly due to the different scenes in the video (moving or stationary car) which change the global image motion therefore varying the compression quality.

It is interesting to notice how the image quality increases significantly (becoming higher than 35 dB even in the 20 kbps video stream) when the camera is not moving, being when the car has stopped at traffic light (between frame 1700 and 2000).

### C. Experimental Results of PC-to-PDA Scenario

Table V and Fig. 9 show the latency measurements in the PC-to-PDA scenario on the VLAB video; VLC is excluded because its PDA player is not actually available. Fig. 9(a) shows the comparison between Windows Media PDA player and Real Media PDA player according to row #7/8 of Table V. Both these systems show an almost-constant, rather-high latency. Moreover, Windows Media shows latency scattering from second 60 to second 90, corresponding to the part of the video sequence when the camera is shaking; also the lost frames are concentrated in this part of the video.

Fig. 9(b) plots the latency of MoSES, encoding video in the three conditions of row #1/2/3 of Table V. The orange plot

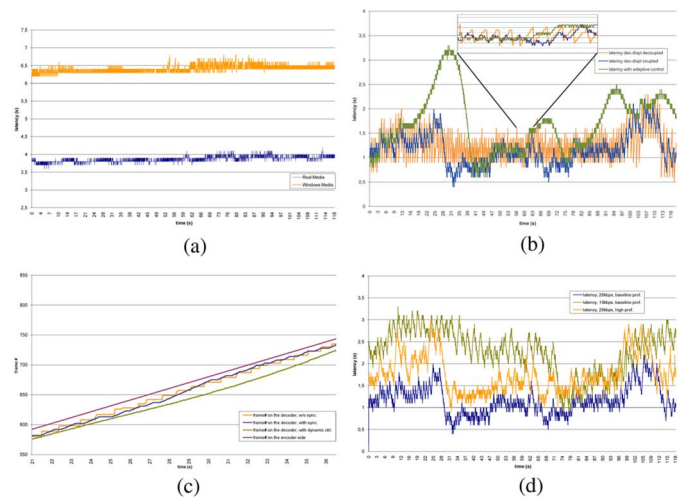


Fig. 9. Comparison in terms of latency over VLAB at QQVGA. The scale of the latency is different on graphs (a), (b), and (d).

(Table V, row #1) shows the latency when the decoder and display threads are running decoupled. This generates a massive presence (79%) of frame losses, due to overwriting. The plot shows sharp and regular peaks due to a twofold reason: buffer underflows (sharp latency raises) and very fast playback (sharp latency falls). Avoiding frame overwriting the frame losses are reduced to 2%: this is the blue plot (Table V, row #2), that shows smaller peaks as expected. The introduction of the decoder-display coupling increases the dependency of the latency on video sequence complexity: for example, the latency suddenly drops around second 28, when the camera starts to move. This is due to the tolerance of the bitrate control: the more complex the scene is, the higher the encoding bitrate and the lower the time to fill up and deliver a datagram. The opposite effect is visible around second 96, when the camera stops moving. In addition, row #2 shows again the positive effect on the average latency because of the introduction of the decoder-display coupling. The green plot (Table V, row #3) of Fig. 9(b) shows the latency when the adaptive control is turned on: since the buffer is initially empty, it reduces  $FR_{\text{playback}}$  of a factor  $\epsilon = 1.002$  each frame; after a few seconds (approximately 15),  $FR_{\text{playback}}$  has reached a critical value and the latency starts to increase until the buffer occupancy becomes definitely greater than zero: at this point, the reaction of the adaptive control becomes effective and reduces the latency through the increase of  $FR_{\text{playback}}$  of  $\epsilon^2$  each frame. The adaptive control drastically reduces the presence of peaks in the latency. Fig. 9(c) shows the frame number trend of the streams of Fig. 9(b) in a short time interval. The plot clearly shows the smoothness in the playback introduced by the adaptive control.

Eventually, Fig. 9(d) shows the latency measured for MoSES in the three conditions of row #2/4/5 of Table V. As expected, the high profile increases the latency. Also the 10 kbps stream latency is higher: as aforementioned, the reduction of encoding bitrate increases the time to fill the UDP datagram.

We also measured latency in a mobile-to-mobile (specifically laptop to PDA) setup (row #6): the plot of the latency is similar to the other cases, but the average and the standard deviation tend to increase. In fact this configuration introduces a further

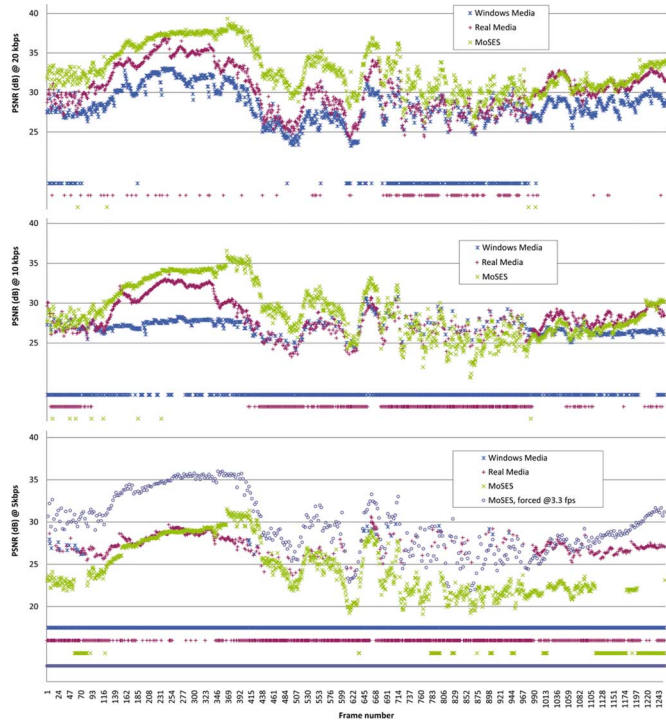


Fig. 10. Comparison of video quality (PSNR and frame losses) over VLAB at QQVGA and 20, 10, 5 kbps. The symbols in the lower part of each graph indicate frame losses.

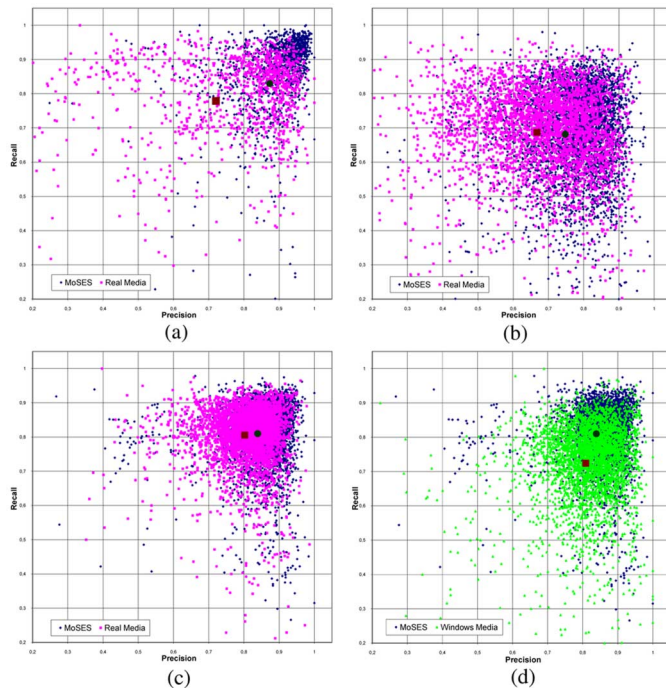


Fig. 11. Recall versus Precision for pixel-level segmentation over VHALL and VPARK at QVGA resolution.

degree of instability in the network communication, due to the additional step of the video data flow on radio mobile channels.

Eventually we calculated the PSNR of the compressed frames and the frame losses of the VLAB on the three systems. MoSES was configured in high profile (row #4 in Table V). Fig. 10 shows

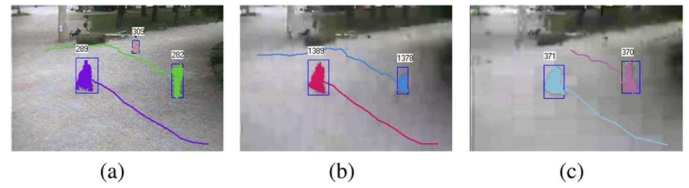


Fig. 12. Snapshots showing the video tracking obtained with Sakbot on the Park sequence: (a) original, (b) MoSES at 5 kbps, (c) Real Media at 5 kbps. The tracking numbers on the objects are just sequential IDs and do not need to be consistent between one video and the others.

TABLE IV  
PSNR AND PERCENTAGE OF LOST FRAMES IN THE PC-TO-PC SCENARIO OVER VLAB AT QVGA AND 80 KBPS

	PSNR in dB avg. (std. dev.)	% of lost frames
Windows Media	34.79 (2.61)	2.09%
Real Media	35.32 (2.66)	9.38%
VLC	32.76 (3.04)	73.96%
MoSES	38.23 (2.16)	4.00%

TABLE V  
LATENCY OF WINDOWS MEDIA, REAL MEDIA, MOSES. PC-TO-PDA SCENARIO, VLAB AT QQVGA. \*DECODER-DISPLAY COUPLING

	Tool	Profile	Bit rate (kbps)	D-D Cpl*	Playb. frame rate	Avg lat. (sec)	Std dev	
1	MOSES	baseline	20	no	fastest	1.25	0.28	
2	MOSES	baseline	20	yes	fastest	1.16	0.30	
3	MOSES	baseline	20	yes	adaptive	1.76	0.39	
4	MOSES	high pr.	20	yes	fastest	1.68	0.58	
5	MOSES	baseline	10	yes	fastest	2.29	0.47	
6	MOSES	mob-mob	baseline	20	yes	fastest	2.96	0.49
7	Win. M.		20			6.42	0.14	
8	Real M.		20			3.87	0.10	

the results at 20 kbps, 10 kbps and 5 kbps. The stronger the compression becomes, the higher the frame loss rate is. It is evident that MoSES outperforms, on average, the other two, especially in terms of lost frames. Table VI reports a summary of the percentage of lost frames and the average PSNR. MoSES could sustain 10 fps even at 5 kbps, but, as expectable, the frame rate is maintained only at the cost of PSNR. Forcing our encoder to skip 2 frames on 3, PSNR increases significantly (Fig. 10): the percentage of lost frames is 67.48% (consider that 66.6% was due to the forced frame skipping at the encoder side), but the average PSNR is 29.76 dB. However, as stated above, this is not a suitable solution for the computer-based video surveillance due to an excessive frame skipping that prevents correct tracking.

D. Experimental Results of Computer-Based Surveillance

The accuracy of the overall system has been measured in terms of both pixel-level segmentation and object-level tracking, by comparing the results achieved by SAKBOT on the original, non-compressed video with those obtained on the compressed, streamed video. Since frames are lost during the transmission, we need a way to align the two videos (original and compressed) for having a correct comparison. The embedding of the frame number used to measure the latency (Section V) is exploited also for video alignment.

TABLE VI  
PSNR AND PERCENTAGE OF LOST FRAMES IN THE PC-TO-PDA SCENARIO OVER VLAB AT QVGA RESOLUTION

	PSNR in dB. Avg. (std. dev.)			% of lost frames		
	20 kbps	10 kbps	5 kbps	20 kbps	10 kbps	5 kbps
Windows Media	28.52 (2.14)	27.02 (0.97)	27.81 (1.01)	17.27%	56.73%	96.11%
Real Media	30.39 (2.80)	28.59 (2.24)	27.36 (1.78)	8.93%	30.04%	60.01%
MoSES	32.80 (2.77)	28.93 (3.28)	24.47 (3.01)	0.32%	0.64%	15.86%
MoSES, forced @3.3 fps	n/a	n/a	29.76 (3.28)	n/a	n/a	67.48%

TABLE VII  
SEGMENTATION AND TRACKING ACCURACY OVER VHALL AND VPARK AT QVGA RESOLUTION

	Video	% of lost frames	Segmentation				Tracking	
			Recall		Precision		Number of Objects	Tracking Accuracy
			avg.	std. dev.	avg.	std. dev.		
MoSES	VHALL - 5 kbps	0.41%	82.95%	1.99%	87.31%	1.40%	29	96.51%
Real Media	VHALL - 5 kbps	8.87%	77.90%	1.63%	72.0%	2.54%	29	81.32%
MoSES	VPARK - 5 kbps	0.11%	68.15%	1.46%	74.8%	1.68%	49	89.11%
Real Media	VPARK - 5 kbps	14.60%	68.67%	1.35%	66.8%	1.91%	49	67.95%
MoSES	VPARK - 20 kbps	0.34%	81.00%	0.98%	83.9%	1.40%	49	91.91%
Real Media	VPARK - 20 kbps	0.02%	80.53%	0.93%	80.2%	1.08%	49	91.83%
Windows Media	VPARK - 20 kbps	1.81%	72.42%	1.32%	80.9%	1.18%	49	89.87%

Fig. 11 shows segmentation recall and precision in different conditions. Each dot represents the recall-precision of a video frame. Obviously, the closer the points are to the upper-right corner (corresponding to  $R = 1$  and  $P = 1$ ) the higher the accuracy is. The graphs also report the average recall and precision, represented by a green circle (MoSES) and brown square (Real Media and Windows Media) circles. The average and variance of recall and precision, computed on the correctly received frames only, are summarized in Table VII. This table also shows the percentage of frame losses due to the strong compression rates.

We initially considered the hardest case in terms of bandwidth, by supposing to send four video streams over GPRS, coding each video at 5 kbps. As a comparison, we tested both MoSES and Real Media. Windows Media and VLC were unable to encode QVGA video at such a low bandwidth.

The graph in Fig. 11(a) shows precision and recall for the VHALL: even with such a limited bandwidth, the segmentation based on MoSES streaming is very close to the one obtained on the original video. This result does not hold in the case of the outdoor sequence of the VPARK [Fig. 11(b)]: in fact, the extensive presence of moving objects and the frequent illumination changes make the compression less effective; in this case the average recall of MoSES is less than 70% and the precision only about 75% (see Table VII). Thus, we also performed a test over EDGE-GPRS using 20 kbps for each video stream. Windows Media supports this bitrate (VLC still does not), and it is then added to the comparative tests [Fig. 11(c) and (d)]. Real Media shows a better recall on Windows Media, but has similar precision. However, MoSES performs better than both the compared systems in almost all working conditions, on recall, precision and frame losses.

Table VII summarizes also the results for object-level tracking. It is evident that with 20 kbps the performance of the tracking (whatever system is used) is not strongly affected, having approximately 90% of accuracy compared to the tracking on the original video. Instead, when the bitrate falls to 5 kbps, only MoSES is able to maintain reasonable

performances. The tracking accuracy on the compressed video depends not only on segmentation accuracy but also on the frame loss rate. The tracking of Real Media on the VPARK at 5 kbps strongly suffers from the high frame loss rate (14.60%), that is concentrated in the portion of the video with higher motion. A sample of the tracking in the park video sequence at 5 kbps is shown in Fig. 12: the tracking consistency is visually represented by the superimposed trajectory of the objects.

## VII. CONCLUSION

This paper reports the efforts for building a complete streaming system for mobile video surveillance. The three basic layers of such systems, specifically encoder, decoder and video surveillance, were implemented with suitable optimization of open source modules to obtain efficient video streaming over GPRS/EDGE-GPRS networks. Measures over image quality, video latency, frame loss and video fluidity were gathered for MoSES and compared with other systems with stored videos even if the system was thoroughly tested in real-time live-video scenario. Then, the performance degradation of the computer-based video surveillance system due only to video compression and streaming has been measured in terms of both pixel-level segmentation and object-level tracking. Our extensive set of experiments has demonstrated the effectiveness of the proposed system in all the three layers. Specifically, we can draw the following conclusions:

- 1) for computer-based surveillance, where low latency is crucial and fluidity is unnecessary, the MoSES system is to be configured with the adaptive playback control disabled. In these conditions, the latency introduced in our system is much lower than in all the compared solutions;
- 2) for human-based surveillance, the adaptive frame rate control strongly improves fluidity, at a cost of latency raise, which remains still much lower than the compared solutions;
- 3) for the tradeoff quality/compression bitrate, in terms of both PSNR and frame losses, the proposed system outperforms the others; this increase in quality makes the com-

puter-based video surveillance feasible with static camera even at bitrates as low as 5 kbps.

## REFERENCES

- [1] K.-Y. Lam and C. Chiu, "The design of a wireless real-time visual surveillance system," *Multimedia Tools Applicat.*, vol. 33, no. 2, pp. 175–199, 2007.
- [2] I. Haritaoglu, D. Harwood, and L. Davis, "W4: Real-time surveillance of people and their activities," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 22, no. 8, pp. 809–830, Aug. 2000.
- [3] Advanced Video Coding for Generic Audiovisual Services ITU Rec. H264/ISO IEC 14996-10 AVC, Tech. Rep., 2003.
- [4] T. Wiegand, G. Sullivan, G. Bjntegaard, and A. Luthra, "Overview of the H.264/AVC video coding standard," *IEEE Transactions on Circuits Syst. Video Technol.*, vol. 13, no. 7, Jul. 2003.
- [5] A. Puri, X. Chen, and A. Luthra, "Video coding using the H.264/MPEG-4 AVC compression standard," *Signal Process.: Image Commun.*, vol. 19, pp. 793–849, 2004.
- [6] [Online]. Available: <https://helixcommunity.org/>, Last Accessed: 3 Apr. 08.
- [7] A. MacAulay, B. Felts, and Y. Fisher, WHITEPAPER-IP Streaming of MPEG-4: Native RTP vs MPEG-2 Transport Stream Envivio, Inc., Oct. 2005.
- [8] [Online]. Available: [iphone.hhi.de/suehring/tml/](http://iphone.hhi.de/suehring/tml/), Last accessed: 3 Apr. 08.
- [9] [Online]. Available: [www.intel.com/cd/software/products/asm-na/eng/302910.htm](http://www.intel.com/cd/software/products/asm-na/eng/302910.htm), Last accessed: 3 Apr. 08.
- [10] [Online]. Available: [www.videolan.org/developers/x264.html](http://www.videolan.org/developers/x264.html), Last accessed: 3 Apr. 08.
- [11] [Online]. Available: [ffmpeg.mplayerhq.hu/](http://ffmpeg.mplayerhq.hu/), Last accessed: 3 Apr. 08.
- [12] J. Lu, "Signal processing for internet video streaming—A review," in *Proc. Conf. on Image and Video Communications and Processing*, 2000, pp. 246–259.
- [13] M.-T. Lu, C.-K. Lin, J. Yao, and H. Chen, "Complexity-aware live streaming system," in *Proc. of IEEE Int. Conf. on Image Processing*, 2005, vol. 1, pp. 193–196.
- [14] G. Conklin, G. Greenbaum, K. Lillevoid, A. Lippman, and Y. Reznik, "Video coding for streaming media delivery on the internet," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 11, no. 3, pp. 269–281, Mar. 2001.
- [15] M. Guo, M. Ammar, and E. Zegura, "V3: A vehicle-to-vehicle live video streaming architecture," in *Proc. IEEE Int. Conf. on Pervasive Computing and Communications*, 2005, pp. 171–180.
- [16] K. Lim, D. Wu, S. Wu, R. Susanto, X. Lin, L. Jiang, R. Yu, F. Pan, Z. Li, S. Yao, G. Feng, and C. Ko, "Video streaming on embedded devices through GPRS network," in *Proc. IEEE Int. Conference on Multimedia and Expo.*, 2003, vol. 2, pp. 169–172.
- [17] Z. Liu and G. He, "An embedded adaptive live video transmission system over GPRS/CDMA network," in *Proc. Int. Conf. on Embedded Software and Systems*, 2005.
- [18] H. Chuang, C. Huang, and T. Chiang, "Content-aware adaptive media playout controls for wireless video streaming," *IEEE Transactions on Multimedia*, vol. 9, no. 6, pp. 1273–1283, 2007.
- [19] C.-F. Wong, W.-L. Fung, C.-F. Tang, and S.-H. Chan, "TCP streaming for low-delay wireless video," in *Second Int. Conf. Quality of Service in Heterogeneous Wired/Wireless Networks*, 2005.
- [20] D. Agrafiotis, T.-K. Chiew, P. Ferre, D. Bull, A. Nix, A. Doufexi, J. Chung-How, and D. Nicholson, "Seamless wireless networking for video surveillance applications," in *Proc. SPIE—Int. Soc. Optical Engineering*, 5685 (Pt. 1), 2005, pp. 39–53.
- [21] X. Cai, F. Ali, and E. Stipidis, "Mpeg4 over local area mobile surveillance system," *IEE Colloq. (Dig.)*, vol. 3–10062, pp. 81–83, 2003.
- [22] Z. He, "Resource allocation and performance analysis of wireless video sensors," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 16, no. 5, pp. 590–599, 2006.
- [23] P. Mahonen, "Wireless video surveillance: System concepts," in *Proc. Int. Conf. Image Analysis and Processing*, 1999, pp. 1090–1095.
- [24] R. Cucchiara, C. Grana, M. Piccardi, and A. Prati, "Detecting moving objects, Ghosts and shadows in video streams," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 25, no. 10, pp. 1337–1342, Oct. 2003.
- [25] W. Ajib and P. Godlewski, "Acknowledgment procedures at radio link control level in GPRS," *Wireless Netw.*, vol. 7, no. 3, pp. 237–247, 2001.
- [26] J. Brunheroto, R. Chernock, P. Dettori, X. Dong, J. Paraszczak, F. Schaffa, and D. Seidman, "Issues in data embedding and synchronization for digital television," in *Proc. IEEE Int. Conf. on Multimedia and Expo.*, 2000, vol. 3, pp. 1233–1236.
- [27] B. Schmidt, J. Northcutt, and M. Lam, "A method and apparatus for measuring media synchronization," *Lecture Notes in Computer Science*, vol. 1018, pp. 190–202, 1995.



**Giovanni Galdi** is currently pursuing the Ph.D. at the University of Modena and Reggio Emilia, Italy.

In 2002–2003, he served as Visiting Scholar in the CVRR Lab at the University of California, San Diego and in 2003–2004, he was with the Mobile & Media Systems Lab, Hewlett Packard Labs, Palo Alto, CA, where he addressed vision-based object tracking in sensor networks for supply chains. His actual research focuses on video surveillance systems in mobile scenarios and object tracking under camera motion.



**Andrea Prati** (M'98) received the M.S. degree in 1998 and the Ph.D. degree in 2001.

He is Assistant Professor at University of Modena and Reggio Emilia, Italy. His research interests are motion analysis for surveillance applications and behavior analysis. He collaborates to research projects at national, European and international level. He is the author of more than 90 papers in national and international journals and conference proceedings.

Dr. Prati has been actively involved in the organization of ACM VSSN'06 and IAPR ICIAP'07, and has been guest editor for special issues on IEEE TRANSACTIONS ON VEHICULAR TECHNOLOGY and for *Expert Systems*.



**Rita Cucchiara** (M'92) received the M.S. degree in 1989 and the Ph.D. degree in 1992.

She is Full Professor at University of Modena and Reggio Emilia, Italy. She is coordinator of the Ph.D. Course of Computer Science and Engineering and heads the ImageLab ([imagelab.ing.unimore.it](http://imagelab.ing.unimore.it)). She is responsible for many Italian and international projects in video surveillance, multimedia, robot vision and medical imaging.

Dr. Cucchiara is Director of the NATO project BE-SAFE for behavioral analysis with machine learning. She was General Chair at ACM VSSN Workshop '05 and '06 and ICIAP '07 and Guest Editor of a Special Issue on ACM *Multimedia Systems* journal in '06. She has more than 40 journal publications and almost 100 refereed conference papers. She is a Fellow of IAPR.