INVITED PAPER

An Introduction to Distributed Smart Cameras

Smart camera systems are designed to produce data, and to recognize and report on objects and activities of interest, rather than to capture images.

By Bernhard Rinner, *Senior Member IEEE*, and Wayne Wolf, *Fellow IEEE*

ABSTRACT | Distributed smart cameras (DSCs) are real-time distributed embedded systems that perform computer vision using multiple cameras. This new approach has emerged thanks to a confluence of simultaneous advances in four key disciplines: computer vision, image sensors, embedded computing, and sensor networks. Processing images in a network of distributed smart cameras introduces several complications. However, we believe that the problems DSCs solve are much more important than the challenges of designing and building a distributed video system. We argue that distributed smart cameras represent key components for future embedded computer vision systems and that smart cameras will become an enabling technology for many new applications. We summarize smart camera technology and applications, discuss current trends, and identify important research challenges.

KEYWORDS | Computer vision; distributed embedded system; multicamera systems; pervasive computing; sensor networks; smart cameras

I. INTRODUCTION

Distributed smart cameras (DSCs) [1], [2] are the result of a convergence of advances in computer vision, very large-scale integration (VLSI) technology, and embedded computing. DSC systems use distributed algorithms to perform complex vision tasks across multiple cameras in real time.

Computer vision requires huge amounts of computing performance and large, high-performance memory. Most computer vision systems have, therefore, been implemented on workstations. But as the capabilities of VLSI systems have advanced, much effort has been put into the development of advanced computer vision applications on embedded platforms. Migrating computer vision on embedded platforms provides several advantages over traditional implementations such as reduced size and power consumption as well as increased reliability. Embedded computer vision will enable a vast number of novel applications.

Smart cameras combine video sensing, processing, and communication on a single embedded platform. They represent a prominent example for embedded computer vision. Their onboard computation and communication infrastructure advances a shift in the processing paradigm of novel computer vision systems. The close colocation of sensing and processing in a smart camera transforms the traditional camera into a smart sensor. For multicamera applications, image processing migrates from central workstations to the distributed embedded sensors. This distributed computing approach helps to reduce the communication load within the network of cameras and to increase the reliability and scalability of the multicamera application.

Distributed smart cameras also embody the trend in sensor networks to increased in-network processing [3]. A smart camera that is part of a network performs huge amounts of computation in order to abstract the raw image data and reduce the bandwidth required to transmit the data. Such networks can take advantage of the basic techniques of ad hoc networking developed for sensor networks, but they also need additional layers to manage the local and nonlocal processing.

Embedded computer vision will penetrate everyday life in the very near future. The deployment of ubiquitous embedded computer vision systems is already happening in several areas. Deployments range from tiny camera systems in biomedical applications over midsize multicamera systems in entertainment and ambient intelligence to large-scale systems extending over hundreds of miles in security and surveillance applications.

Manuscript received January 2, 2008; revised April 29, 2008. First published October 17, 2008; current version published October 31, 2008. The work of W. Wolf was supported in part by the Army Research Office and the National Science Foundation under Grant 0325119. The work of B. Rinner was supported in part by the Austrian Research Promotion Agency under Grants 810072, 812204, and 812033.

B. Rinner is with the Institute of Networked and Embedded Systems, Klagenfurt University, 9020 Klagenfurt, Austria (e-mail: bernhard.rinner@uni-klu.ac.at).
W. Wolf is with the School of Electrical and Computer Engineering, Georgia Institute of Technology, GA 30332 USA (e-mail: wolf@ece.gatech.edu).

Digital Object Identifier: 10.1109/JPROC.2008.928742

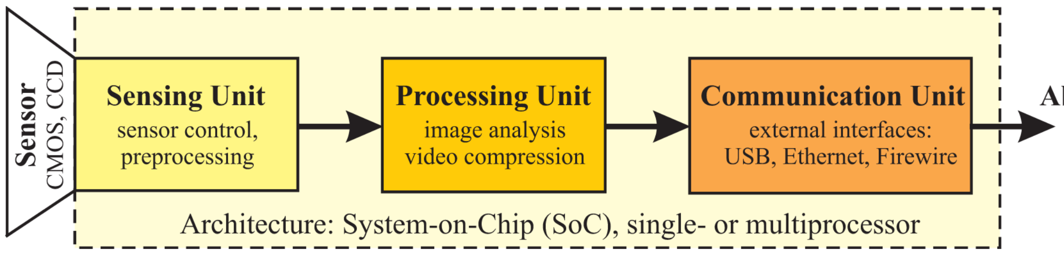

Fig. 1. ***A generic architecture of a smart camera.***

Modern camera installations are large enough that the underlying computing architecture becomes an important factor. Consider, for example, Atlanta Hartsfield-Jackson International Airport (ATL). The airport has 179 gates spread over 52.6 hectares.[1] Even with only one camera per gate, the system for this one facility would have a large number of cameras. If we want to know what is happening at a gate, we need at least several cameras, and possibly dozens. We also have to cover many other areas in the airport: the tarmac; interior workspaces such as baggage-handling areas, security inspection areas, check-in, parking lots, and roadways. These cameras will be distributed over a wide area, and it makes much more sense to perform at least some of the processing locally. Given that ATL handled 84 million passengers in 2006, hundreds or thousands of cameras required for adequate spatial coverage can be put to good use, even if they are only used to monitor congestion and space utilization.

Research on distributed smart cameras has gained a lot of attention over the last few years from a large and growing community of researchers from academia as well as industry, as demonstrated by successful events and an increasing number of dedicated publications (e.g., [4]–[6]).

We argue that distributed smart cameras represent key components for future embedded computer vision systems and that smart cameras will become an enabling technology for many new applications. The next section introduces smart cameras, which are the building blocks of distributed smart camera systems. We next describe the importance of multiple, physically distributed cameras in computer vision. We follow by showing how we can accomplish real-time processing on a distributed computing platform made of smart cameras. We then discuss over several sections the distributed computing platform that allows us to design such vision systems and the interactions between algorithms and the hardware/software platform. We close with discussions of trends and research challenges.

[1]http://www.atlanta-airport.com/applications/trakaflight/flightinfo_frames.htm.

II. SMART CAMERAS AND EMBEDDED COMPUTER VISION

Fig. 1 depicts a generic architecture of a smart camera comprised of a sensing, processing, and communication unit. The image sensor, which is implemented either in complementary metal–oxide–semiconductor (CMOS) or charge-coupled device (CCD) technology, represents the data source of the processing pipeline in a smart camera. The sensing unit reads the raw data from the image sensor and often performs some preprocessing such as white balance and color transformations. This unit also controls important parameters of the sensor, e.g., capture rate, gain, or exposure, via a dedicated interface. The main image-processing tasks take place at the processing unit, which receives the captured images from the sensing unit, performs real-time image analysis, and transfers the abstracted data to the communication unit. This unit provides various external interfaces such as USB, Ethernet, or Firewire.

These generic units are implemented on various architectures ranging from system-on-chip (SoC) platforms over single processor platforms to heterogeneous multiprocessor systems. Field-programmable gate arrays (FPGAs), digital signal processors (DSPs), and/or microprocessors are popular computing platforms for smart camera implementations. A variety of smart cameras have been developed over the years ranging from matchbox-sized modules with a total power consumption of a few hundred milliwatts (e.g., [7] and [8]) to commercial off-the-shelf based prototypes requiring some tens of watts (e.g., [1] and [9]).

Smart cameras deliver some abstracted data of the observed scene. It is natural that the delivered abstraction depends on the camera's architecture and application, and almost every smart camera currently delivers a different output. Smart cameras perform a variety of image-processing algorithms such as motion detection, segmentation, tracking, object recognition, and so on. They typically deliver color and geometric features, segmented objects, or rather high-level decisions such as wrong-way drivers or suspect objects. The abstracted results may be transferred either within the video stream, e.g., by color coding, or as a separate data stream. Note that the onboard computing infrastructure of smart cameras is often exploited to perform high-level video compression and only transfer the compressed video stream.

A. Architectural Issues

Smart cameras are enabled by advances in VLSI technology and embedded system architecture. Modern embedded processors provide huge amounts of performance; one recent example of an embedded processor tailored to image computation is DaVinci from Texas Instruments [10].

However, smart cameras are not simply cost-reduced versions of arbitrarily selected computer vision systems. Embedded computer vision requires distinct techniques from non-real-time computer vision because of the particular stresses that vision algorithms put on computer systems. Unfortunately, CPU clock speed does not equate to performance because some units in the processor may be underutilized—data may not always be available to keep the units busy. Memory is a principal bottleneck of computer system performance because memory speed does not increase with Moore's law [11]. As a result, memory continually gets slower relative to the processor. In general-purpose computer systems, caches are used to store frequently used values and increase the average performance of the memory system. However, computer vision algorithms, much like video compression algorithms, use huge amounts of data, and often with less frequent reuse. As a result, caches may be less effective. At a minimum, software must be carefully optimized to make best use of the cache; at worst, the memory system must be completely redesigned to provide adequate memory bandwidth [12].

Beside memory capacity and memory bandwidth, computing power is a crucial resource for embedded computer vision. The individual stages of the typical image-processing pipeline raise different requirements on the processing elements. Low-level image processing such as color transformations and filtering operates on individual pixels in regular patterns. These low-level operations process the complete image data at the sensor's frame rate but typically offer a high data parallelism. Thus, low-level image processing is often realized on dedicated hardware such as application-specific integrated circuits, FPGAs, or specialized processors [13]. High-level image processing, on the other hand, operates on (few) features or objects, which reduces the required data bandwidth but increases the complexity of the operations significantly. These complex processing tasks exhibit typically a data-dependent and irregular control flow. Thus, programmable processors are the prime choice for these tasks. Depending on the complexity of the image-processing algorithms, even multicore or multiprocessor platforms may be deployed [1], [7].

The native datapath is a related issue of the processing elements. Low-level image processing is often implemented on fixed-point architectures since they are faster and more power-efficient than floating-point architectures. High-level processing tasks often require higher dynamic range and precision. When such algorithms are implemented on fixed-point architectures, floating-point arithmetic must be emulated in software [14].

Standardized communication interfaces are used to transfer the processed image data. The main distinction here is whether the communication interface is able to transfer the captured image data in real-time. Cameras with wired interfaces such as Ethernet, gigabit Ethernet, or Firewire provide sufficient bandwidth for real-time transfer of raw image data. Smart cameras with wireless interfaces have been developed more recently, especially using communication protocols for sensor networks such as ZigBee. These low-bandwidth protocols basically inhibit real-time streaming of raw image data but keep the power consumption low [3].

B. Smart Camera Evolution

Smart cameras have been the subject of study at both research labs and companies for quite some time. While in the 1980s some camera prototypes that integrate sensing with some low-level processing were developed, commercial "intelligent" cameras appeared in the 1990s. However, the onboard processing capabilities were very limited: the intelligent camera provided just an SVGA output.

Research on single smart cameras intensified in the late 1990s. Moorhead and Binnie [15] presented one of the first fabricated CMOS implementations. Their system on a chip (SoC) smart camera integrated edge detection into the image sensor. VISoc [16] represents another smart camera-on-a-chip implementation featuring a 320×256 pixel CMOS sensor, a 32-bit reduced instruction set computer processor, and a vision/neural coprocessor. Heyrman *et al.* [17] have proposed an SoC smart camera with on-chip motion detection. Programmable hardware such as FPGAs has been frequently used as processing platform for smart cameras. The applications of smart-camera FPGA implementations range from ID recognition [18] over fast range-finding [19], [20] to high-speed tracking [21]. Another application of smart cameras on programmable hardware is active vision, which aims at integrating the control of the image sensor in the perception loop, especially in the early vision processes. Chalimbaud *et al.* [22], [23] have implemented an active vision system composed of a CMOS sensor and a single FPGA.

Boult *et al.* [24], Rinner *et al.* [1], and Ozer *et al.* [25] have developed real-time embedded computer vision systems. In the meantime, many companies have come up with smart cameras as well. Heterogeneous multiprocessor architectures, including application-specific accelerators, are often used to improve the performance of embedded systems [26]. An example of an accelerator designed for embedded computer vision is the optical flow unit designed by Schlessman *et al.* [27]. This unit implements the Kanade–Lucas–Tomaso (KLT) algorithm [28] using floating-point arithmetic. Because IEEE floating point is too expensive, it uses a custom-designed limited-precision floating-point format. The dynamic range and precision of the floating-point format was determined by numerical analysis of the KLT algorithm.

C. Applications

Smart cameras have already been used in many applications. We briefly review representative examples in different application categories.

- *Intelligent Video Surveillance Systems (IVSS)*. IVSS is a very active research area. The fundamental goal in IVSS is to detect "abnormal" behaviors in the observed scene [29], [30]. This requires complex image analysis starting from motion detection to segmentation, feature extraction, and classification. To extend the spatial sensor coverage IVSS are typical examples for multicamera systems. However, most traditional IVSS require no or only little cooperation among individual cameras.
- *Intelligent Transportation Systems (ITS)*. Vision systems in ITS can be divided into infrastructure-based systems and vehicle-based systems. The infrastructure-based systems are closely related to IVSS. Infrastructure-based systems are typically large multicamera deployments with stationary cameras. The image analysis tasks are focused on traffic monitoring. Vehicle-based systems are typically built up of single but mobile cameras. Smart cameras already play an important role in intelligent automobiles. Embedded vision components are used to monitor the vehicle's environment as well as the driver's state and attention inside of the vehicle. Example implementations include a smart-camera based adaptive cruise control for intelligent vehicles or forward collision-warning systems.
- *Medicine*. Smart cameras can be used in many medical applications. The RVT system by Leeser [31] is an FPGA-based smart camera that allows surgeons to see live retinal images with vasculature highlighted in real time during surgery. Smart cameras can also be used to automate experiments; the PhenoScan system from Clever Systems[2] analyzes mouse behavior for drug experiments. Smart cameras can also be used to monitor patients and medical personnel. For example, U.S. laws require that more than one person be present when narcotics are handled, a rule that can be checked by computer vision.
- *Entertainment and Smart Environments*. There are many important applications for smart cameras including robotics, medical imaging, and entertainment. Gesture recognition will play an important role for multimodal user interfaces. Such a gesture-recognition system using distributed smart cameras has been developed at Princeton [32]. Aghajan *et al.* [33] have also developed a gesture—recognition system using distributed cameras. They focus on fusing gesture elements from different cameras to reliably identify the human gesture. In robotics, smart cameras are deployed as a reliable smart sensor [34].
- *Machine Vision*. "Intelligent" cameras have been extensively deployed in machine vision, which deals with the application of computer vision methods to manufacturing, inspection, and robotics. These applications often pose strong requirements on the processing speed and robustness of the vision methods, which are therefore embedded in the overall application.

[2]http://www.cleversysinc.com.

III. DISTRIBUTED CAMERAS

The term *distributed camera* refers in computer vision to a system of physically distributed cameras that may or may not have overlapping fields of view. The images from these cameras are analyzed jointly. Distributed cameras allow us to see a subject of interest from several different angles. This, in turn, helps us solve some very hard problems that arise in single-camera systems.

Occlusion is a major problem in single-camera systems. A subject may be occluded by another object; if the subject is nonconvex, part of the subject may be occluded by another part. When we have multiple views of a subject, we are much more likely to be able to see the parts of an object occluded in one view by switching to another camera's view.

Another way to think about distributed cameras is *pixels on target*. Our ability to analyze a subject is limited by the amount of information, measured in pixels, that we have about that subject. Not only do distributed cameras give us several views, but one camera is more likely to be closer to the subject. A traditional camera setup would use a single camera to cover a large area. Subjects at the opposite end of the space would be covered by very few pixels. Distributed camera systems help us cover the space more evenly.

Occlusion may be *static* or *dynamic*. A fixed object, such as a wall or a table, causes occlusion problems that are easier to predict. When one moving object occludes another, such as when two people pass each other, occlusion events are harder to predict.

The number of cameras we need to cover a space depends on both the field of view of the camera and the required number of pixels on target. For a typical camera with a rectangular image sensor, the field of view is a pyramid extending from the lens, as shown in Fig. 2. The angular field of view of the lens determines the size of this pyramid. A *normal lens* provides the same angular field of view as does the human eye, between 25° and 50°. A wider lens covers more area in the scene, spreading a given number of pixels over a larger area in the scene. A longer lens covers less area in the scene, putting more pixels on the target.

The pixels-on-target criterion and the size of the smallest target of interest tell us how far this pyramid extends from the camera. For example, a common

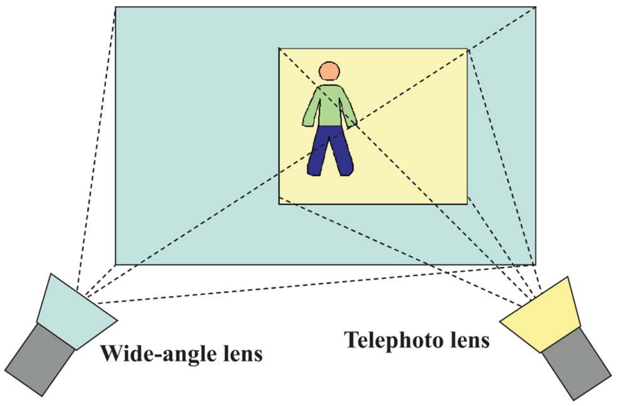

Fig. 2. ***Field of view and pixels-on-target of a camera.***

intermediate format image is 362 × 240 pixels. When we place this array of pixels across the field-of-view pyramid, as shown in Fig. 2, we can easily calculate the number of pixels that cover a target of a given size at various distances from the camera.

Given the field-of-view volumes dictated by our cameras and application requirements, and given the number of different cameras that should cover any given point in a space, we can determine the number of cameras required to provide that coverage. A simple case is a rectangular room as shown in Fig. 3. Once we add even a simple occluding object, such as a box, covering the space becomes harder. If the occluding object occupies less volume than the field-of-view that it blocks, then we must add cameras to maintain the same coverage. A thin occluding object, such as a wall or table, is a worst case occlusion since it occupies little spatial volume but can block a large field-of-view volume. An occlusion may be static or dynamic. For example, person 1 walks behind the box, causing a temporary, dynamic occlusion, while a static object behind the box is statically and permanently occluded. Subjects can also occlude each other, as when person 3 blocks person 2 in this illustration.

A. Tracking

Tracking is one of the major topics in computer vision. A variety of algorithms have been developed to track moving objects. In this section, we review traditional tracking algorithms [35].

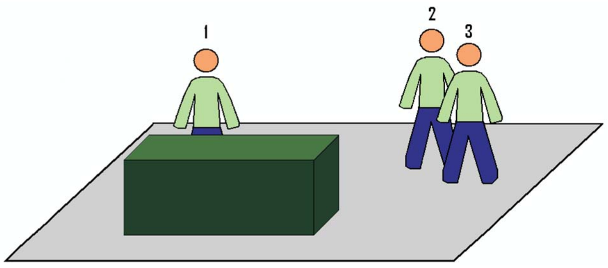

Fig. 3. ***Occlusion and fields of view.***

Several models have been developed for the appearance of a target, most notably the mean shift [36]. Different algorithms can be used to generate and predict the track, including Kalman filters [37] and the condensation algorithm, also known as particle filtering [38].

Many groups have designed tracking algorithms using various combinations of algorithms. Comaniciu *et al.* [39] regularize target description histograms by spatial masking with an isotropic kernel and use mean shift to determine similarity. Ricquenbourg and Bouthemy [40] track moving persons by tracking contours of the person without a three-dimensional model. Zhao *et al.* [41] track a segmented human shape using a Kalman filter with explicit occlusion handling. Comaniciu *et al.* [42] use color cues and mean-shift iterations to find the most probable target position.

Cai and Aggarwal [43] discuss tracking with a distributed camera system. Javed *et al.* [44] developed an appearance model for tracking with multiple, nonoverlapping cameras. Cevher *et al.* [45] used particle filters to track using audio and video sensors.

An extension of the single-target tracking problem is multitarget tracking, which tries to disambiguate multiple targets in the face of mutual and nonmutual occlusion. One multitarget tracking approach was developed by Oh *et al.* [46], who proposed Markov chain Monte Carlo algorithms. Liu *et al.* [47] separated position from target modeling information in order to create a distributed algorithm for multitarget tracking. Cevher *et al.* [45] tracked multiple targets by fusing information from video and audio sensors using particle filters.

IV. DISTRIBUTED COMPUTING AND DISTRIBUTED SMART CAMERAS

The above algorithms, although they use distributed cameras, do not operate on a distributed computing platform. A distributed computer is a network of processors in which the nodes (processors) do not have direct knowledge of the state of other nodes. A node can retrieve the state of another node only by receiving a message from that node. Messages in distributed systems have nontrivial costs, so distributed algorithms are designed to minimize the number of messages required to complete the algorithm.

Distributed computers introduce several complications. However, we believe that the problems they solve are much more important than the challenges of designing and building a distributed video system. As in many other applications, distributed systems scale much more effectively than do centralized architectures. Many realistic applications require enough cameras that server-based architectures are not realistic.

Processing all the data centrally poses several problems. Video cameras generate large quantities of data. If we transmit raw video to a server, the network must be able to handle the required bandwidth in steady state. Furthermore, the server itself must be able to handle such large data quantities and move it from the network interface through the memory, into the processor, and out to mass storage.

Moving video over the network also consumes large amounts of energy. In many systems, communication is 100 to 1000 times more expensive in energy than computation. We can afford to perform a great deal of computation in order to reduce bandwidth requirements. We do not expect camera systems to be run from batteries for long intervals, but power consumption is a prime determinant of heat dissipation. A realistic camera system should not heat up the surrounding environment too much. Distributing larger amounts of power also requires more substantial power distribution networks, which increases the installation cost of the system.

Although data must be compared across several cameras to analyze video, not all pairs of cameras must communicate with each other. If we can manage the data transfer between processing nodes, we can make sure that data only go to the necessary nodes. A partitioned network can protect physically distributed cameras so that the available bandwidth is used efficiently.

Real-time considerations also argue in favor of distributed computing. The round-trip delay to a server and back adds to the latency of making a decision, such as whether a given activity is of interest. A distributed system can make sure only relevant nodes are involved in a given decision. To the extent that we avoid sharing common resources, we also increase the predictability of processing time.

A. Representative Distributed Smart Camera Systems

The VSAM project [48] was one of the first surveillance systems using distributed sensor processing units (SPUs), which can be seen as some form of embedded cameras. These SPUs are capable of detecting and tracking objects, classifying the moving objects into categories such as "human" or "vehicle" and identifying simple human behavior such as walking. The SPUs perform some cooperation and sensor fusion.

Mallet and Bove [49] developed a set of cameras that cooperated to perform vision tasks. Fleck *et al.* [50] present a surveillance system consisting of a distributed network of smart cameras that allows for tracking and handoff of multiple persons in real time. Their multiobject tracker is based on color-based particle filters. Tracking handoff is achieved by cooperation among the smart cameras based on a centralized 3-D model of the observed scene. Bramberger *et al.* [51], [52] introduce a different multicamera tracking approach on distributed smart cameras. In this approach, tracking agents "follow" the tracked object, i.e., when the object leaves the field of view of a camera, the tracking task is migrated to the camera(s) that should next observe the object. The handoff is autonomously handled among adjacent cameras in the network. Velipasalar *et al.* [53] developed a peer-to-peer architecture for tracking. This system performs multiple-camera tracking without resorting to a central server.

B. Distributed Gesture Recognition

To help us understand the structure of a distributed smart camera system and the choices we face in designing one, we will use an example—the distributed gesture recognition system of Lin *et al.* [32]. This distributed system was based on the single-node smart camera system of Ozer *et al.* [2]. One useful way to design a distributed smart camera system is to start with a single-camera algorithm and decide how to partition it into a distributed system.

Fig. 4 shows a subject captured by a distributed smart camera system. Neither camera has a full view of the subject. As a result, we must combine data from multiple cameras to build a complete model of the subject.

We could send raw or compressed video frames between nodes, but this would consume considerable bandwidth and does not take advantage of our knowledge of the data. In many vision algorithms, we perform several stages of analysis, producing increasingly abstract representations of the video signal at each stage. If a processing stage can be

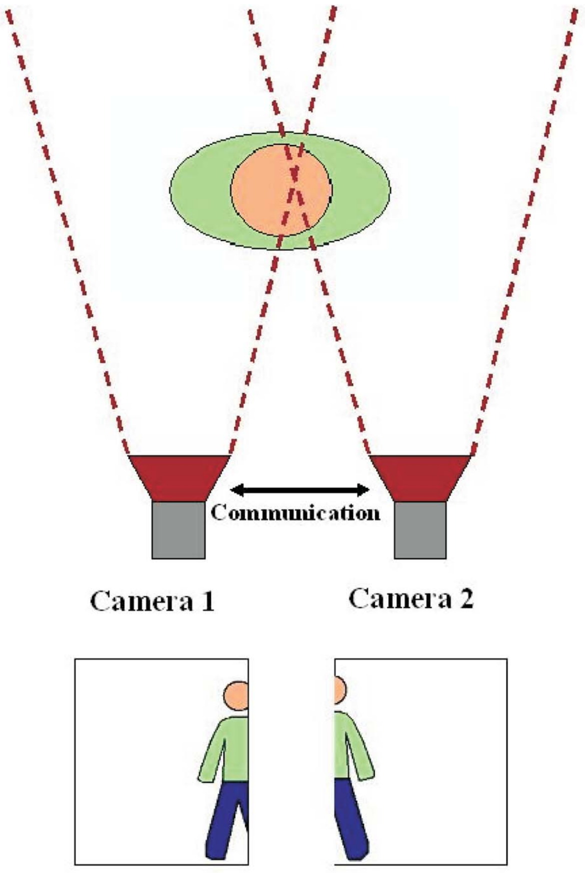

Fig. 4. *A gesture-recognition subject moving along a wall of cameras.*

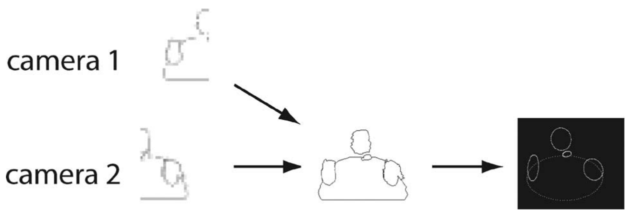

Fig. 5. ***Building a unified model of the target from multiple cameras.***

performed without combining the results of multiple cameras, we can then perform that stage locally and pass the abstraction to the other camera for fusion. In the case of our gesture-recognition system, a good intermediate stage is the outline of the selected foreground region. As illustrated in Fig. 5, one camera can extract its contour and send it to the other camera, which then proceeds to complete the construction of the model (the next step being, in this case, generation of ellipses to model the contour).

However, the communication patterns between the cameras should not remain fixed. As the subject moves, not only do different cameras gain and lose view of the subject but also the computational roles of the camera nodes should change. If a camera loses view of a subject, we generally want to remove that camera node from the subject's computation (except, perhaps, for load-balancing or fault-tolerance reasons). Furthermore, the choice of which camera performs the final steps of recognition may change. In the example of Fig. 5, the choice of lead camera is somewhat arbitrary since each camera has an equal view of the subject. We therefore should be willing to move the locus of computation on the subject—represented by a token—around the network as the subject moves. A protocol moves the token around the network, based upon the subject movement, load balancing considerations, etc.

V. DISTRIBUTED ALGORITHMS AND MIDDLEWARE

The distribution of data is related to the movement of the locus of processing in the network. Since individual cameras collaborate in the processing of image data, the cameras must share their (abstracted) data. This data distribution is more complex in a network of collaborative smart cameras than in a centralized multicamera network. A protocol distributes the data around in the network, considering spatial and temporal collaboration patterns, synchronization, and so on.

Implementing such protocols without support from system-level software might be tedious. So we would like to take advantage of middleware services well known on general-purpose computer networks [54] on networks of smart cameras as well. A middleware abstracts the network such as an operating system abstracts the hardware of a single computer. It may provide some fairly generic services for distributed computing such as networking, data exchange, and distributed control, but also specific services for DSC networks.

However, the requirements of a middleware for distributed image processing on embedded devices are significantly different. Component-based middleware such as DCOM or CORBA are targeted for general-purpose computing and are not suitable for resource-limited devices. Recent research in *wireless sensor networks* (WSNs) has come up with some interesting middleware concepts as well [55]. Due to the nature of WSNs, these middleware systems especially focus on reliable services for ad hoc networks and energy awareness [56].

DSC networks are different from WSNs in various aspects. First, the amount of data to be processed is much higher in DSC networks than in WSNs. Secondly, individual processing nodes in a DSC network provide more computing resources than in WSNs. While resource constraints on the embedded smart cameras are important, the resource limitations, especially energy, are of top priority in WSN. Thirdly, due to ad hoc networking, communication in WSN has a very dynamic nature. DSCs, on the other hand, are typically connected via wired networks providing higher communication bandwidths.

We will use the software architecture of our smart cameras [1] as an example for a middleware. Fig. 6 presents the architecture of this lightweight middleware. The individual services are organized in a layered structure. The operating system along with its device drivers and communication channels builds the basic layer of a single camera, which consists in our case of a network processor and digital signal processors (DSPs). Host services provide a flexible message-oriented communication between applications on the network processor and multimedia algorithms on the DSPs. Dynamic loading enables the exchange of multimedia algorithms on the DSPs at time of deployment and during runtime. Monitoring services help to supervise the current state of the camera.

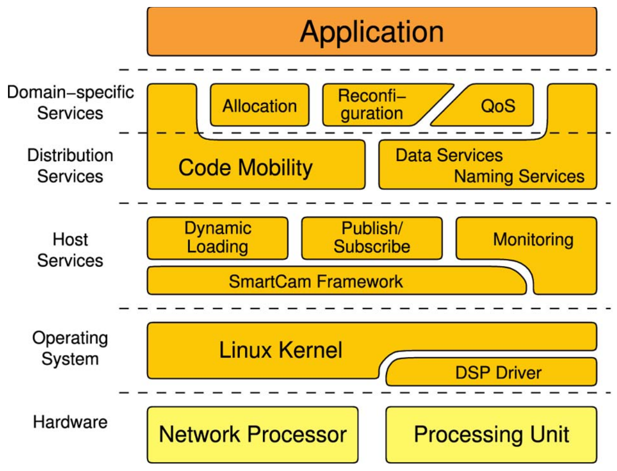

Fig. 6. ***The architecture of the SmartCam middleware.***

While the lower layers provide services for applications on a single camera, distribution services integrate multiple smart cameras to a distributed image-processing system. In our implementation, we use a *mobile agent system* as foundation for distributed applications [57], [58].

An agent system usually supports communication between agents independent of the current host an agent resides on. The agent system further facilitates the abstraction of image-processing tasks by mobile agents. Low-level image processing is implemented as DSP executable while the agent contains the application logic and controls the image-processing algorithm. Code mobility is also inherent to a mobile agent system. Mobile agents can migrate between the hosts as required. Exploiting dynamic loading of DSP executables allows one to migrate the image-processing algorithm as well.

We have demonstrated our middleware in a multicamera tracking application [52] with an autonomous handoff process. The handoff is managed by the adjacent cameras only achieving high scalability. This handoff of a tracker from one camera to the next is realized by mobile agents, i.e., the tracking agent "follows" the tracked object in the camera network. This multicamera tracking has been completely implemented on the embedded smart cameras—requiring no central coordination at all.

VI. MODELING

Modeling of the elements of an image—tracking targets, for example—is an issue that cuts across computer vision and distributed computing. On the one hand, we need representations and models that allow us to perform hierarchical image processing, in which we move from lower level representations like pixels to higher level representations that can be manipulated without reference to their source signals. On the other hand, we want to choose compact representations that can be efficiently moved around the network.

Early distributed systems that performed handoff during tracking could rely on minimal models of the target—position is sufficient to hand off targets in many situations. The tracker of Velipasalar *et al.* [53] added bounding box and appearance information. While this representation is relatively compact, it also cannot be used for certain operations by other nodes. For example, if we wanted to combine information from several nodes to split and merge blobs into individual targets, we would need to include the blobs in the model. The abstraction used by the gesture recognition system of Lin *et al.* [32] is well suited to the algorithms used by that system, since the pixel-level boundaries can be easily combined and analyzed. If we wanted to add new processing at remote nodes, we may need to adapt its model.

Many questions remain in hierarchical image processing. Hierarchical algorithms are well known in video compression and in some aspects of feature extraction for image search. However, these applications generally do not require building high-level models of the image, as is done in applications like tracking. A powerful set of features that describe objects of interest in scenes would allow us to build models without dipping back into the pixels that underlie the features.

VII. TRENDS AND CHALLENGES

Considering the recent advances of smart cameras in research and industrial practice, we can identify several trends and research challenges.

A. From Static to Dynamic and Adaptive

Until recently, the camera's functionality was determined at time of assembly; only some customization was possible at best. Increased onboard processing power now enables the realization of adaptive behavior. Adaptivity is the key capability to tackle the ever increasing complexity. Novel camera networks are expected 1) to execute adaptive algorithms in order to better account for changes in the observed scene, 2) to exploit static and mobile cameras such as PTZ cameras, and 3) to change their functionality to provide increased autonomy. Foresti *et al.* [59], Pflugfelder *et al.* [60], and Chen *et al.* [61], among others, have worked on some aspects for increasing the autonomy.

B. From Small to Very Large Camera Sets

As the airport example showed, a very large number of cameras may be required to analyze a single space. In many cases, that space will be undersampled, with some parts of the area not visible. Analysis algorithms must be able to infer the activity that happens in hidden parts of the space based upon the activity visible to them. Niu and Grimson [62] and Song and Roy-Choudhury [63], among others, have experimented with large camera sets.

Very large camera systems also lead us to consider new types of information that we want to extract from the cameras. For example, when we use many cameras to analyze very large areas, we may be less interested in tracking individuals and more interested in statistics on behavior.

C. From Vision-Only to Multisensor Systems

We expect to see researchers start to integrate different sensors—audio, seismic, thermal, etc.—into distributed smart sensor networks. By fusing data from multiple sensors, the smart camera exploits the distinct characteristics of the individual sensors resulting in an enhanced overall output (e.g., [64] and [65]). Peer-to-peer algorithms are useful in multimodal systems for many of the same reasons they are used in camera networks.

D. Development Process of Distributed Image-Processing Systems

Designing, deploying, and operating applications for distributed smart cameras require novel approaches to support the development of distributed, reliable, and scalable applications. This development process must combine the approaches from distributed computing, sensor networks, and computer vision but considering the specialties of distributed image processing such as rather high data transfers between the nodes, complex computations on the nodes, and strong real-time constraints.

Until now, DSC applications have been developed more or less from scratch without much tool support. We expect to see progress in the development process, i.e., introducing new design methods, adapting tools as well as unifying platforms and system-level software.

E. Network Services

Distributed smart camera systems can be seen as some form of sensor network with different characteristics: 1) fewer nodes; 2) higher computing, communication, and power resources per node; and 3) in most cases stationary nodes with established networking.

The unique characteristics of distributed smart cameras as sensor networks leads us to some important problems. Higher computing and communication rates lead us to consider more sophisticated distributed computing services, such as task migration and load balancing. These distributed services are particularly important for networks that are used in safety-critical applications. Camera networks must be accurately calibrated in both space and time; when audio information is added to the system, temporal calibration becomes even more important.

F. Privacy and Security

The paper by Widen in this Special Issue [69] describes the law on privacy as it relates to cameras. Because camera imagery can be used to identify individuals, new algorithms and system architectures are needed to protect privacy while allowing useful information to be gathered. By being able to perform onboard image analysis and hence to avoid transferring raw data, smart camera cameras have great potential for increasing privacy and security. Boult *et al.* [66] and Fleck *et al.* [67], among others, have explored smart cameras in privacy-sensitive applications by omitting the transfer of images of some parts of the observed scene. However, a more holistic approach towards privacy and security is needed [68].

VIII. CONCLUSIONS

Distributed smart cameras have emerged thanks to the simultaneous advances in four key disciplines: computer vision, image sensors, embedded computing, and sensor networks. Today's abundance of CMOS image sensors has allowed us to seriously consider building large systems of multiple cameras; VLSI technology has provided the processing power to handle the data generated by those image sensors. Computer vision has developed many basic algorithms for analyzing imagery. Embedded computing and sensor network topology has provided the tools to perform computer vision in real time and to organize large networks of cameras.

However, to make progress in this field, we cannot consider these contributing technologies as separate. Computer vision algorithms must be appropriately designed to operate at low latencies. Ad hoc network architectures must be extended to handle high data rates and substantial distributed processing.

As the technology for distributed smart cameras advances, we expect to see many new applications open up. Distributed smart cameras are one aspect of the revolution in cameras that is taking place at the dawn of the twenty-first century—lenses, image sensors, processors, and networks. This revolution will change our conception of cameras as boxes that capture images into a more general notion of cameras as spatially distributed that generate data and events. ■

REFERENCES

[1] M. Bramberger, A. Doblander, A. Maier, B. Rinner, and H. Schwabach, "Distributed embedded smart cameras for surveillance applications," *Computer*, vol. 39, no. 2, pp. 68–75, Feb. 2006.

[2] W. Wolf, B. Ozer, and T. Lv, "Smart cameras as embedded systems," *Computer*, vol. 35, no. 9, pp. 48–53, Sep. 2002.

[3] I. F. Akyildiz, T. Melodia, and K. R. Chowdhury, "A survey on wireless multimedia sensor networks," *Comput. Netw.*, vol. 51, pp. 921–960, 2007.

[4] B. Rinner and W. Wolf, Eds., *Proc. Int. Workshop Distrib. Smart Cameras (DSC 06).*, Boulder, CO, Oct. 2006.

[5] H. Aghajan and R. Kleihorst, Eds., *Proc. ACM/IEEE Int. Conf. Distrib. Smart Cameras (ICDSC 06).*, Vienna, Austria, Sep. 2007.

[6] H. Aghajan, R. Kleihorst, B. Rinner, and W. Wolf, "Special issue on distributed processing in vision networks," *IEEE J. Sel. Topics Signal Process.*, vol. 2, no. 4, 2008.

[7] R. Kleihorst, A. Abbo, B. Schueler, and A. Danilin, "Camera mote with a high-performance parallel processor for real-time frame-based video processing," in *Proc. ACM/IEEE Int. Conf. Distrib. Smart Cameras (ICDSC 2007)*, Vienna, Austria, Sep. 2007, pp. 109–116.

[8] A. Rowe, D. Goel, and R. Rajkumar, "Firefly mosaic: A vision-enabled wireless sensor networking system," in *Proc. 28th IEEE Int. Real-Time Syst. Symp. (RTSS 2007)*, D. Goel, Ed., 2007, pp. 459–468.

[9] S. Hengstler, D. Prashanth, S. Fong, and H. Aghajan, "Mesheye: A hybrid-resolution smart camera mote for applications in distributed intelligent surveillance," in *Proc. 6th Int. Symp. Inf. Process. in Sensor Netw. (IPSN 2007)*, Apr. 25–27, 2007, pp. 360–369.

[10] D. Talla and J. Golston, "Using DaVinci technology for digital video devices," *IEEE Computer*, vol. 40, pp. 53–61, Oct. 2007.

[11] D. A. Patterson and J. L. Hennessy, *Computer Architecture: A Quantitative Approach*, 4th ed. San Francisco, CA: Morgan Kaufman, 2006.

[12] W. Wolf, *High Performance Embedded Computing*. San Francisco, CA: Morgan Kaufmann, 2006.

[13] R. Kleihorst, B. Schueler, and A. Danilin, "Architecture and applications of wireless smart cameras (networks)," in *Proc. IEEE Int. Conf. Acoust., Speech, Signal Process. (ICASSP 2007)*, Honolulu, HI, Apr. 2007.

[14] D. Menard, D. Chillet, F. Charot, and O. Sentieys, "Automatic floatingpoint to fixedpoint conversion for DSP code generation," in *Proc. IEEE Int. Conf. Compilers, Architect. Syst. Embed. Syst. (CASES)*, 2002.

[15] T. W. J. Moorhead and T. D. Binnie, "Smart CMOS camera for machine vision applications," in *Proc. Inst. Elect. Eng. Conf. Image Process. Applicat.*, Manchester, U.K., Jul. 1999, pp. 865–869.

[16] L. Albani, P. Chiesa, D. Covi, G. Pedegani, A. Sartori, and M. Vatteroni, "VISoc: A smart camera SoC," in *Proc. 28th Eur. Solid-State Circuits Conf.*, Florence, Italy, Sep. 2002, pp. 367–370.

[17] B. Heyrman, M. Paindavoine, R. Schmit, L. Letellier, and T. Collette, "Smart camera design for intensive embedded computing," *Real-Time Imag.*, vol. 11, pp. 282–289, 2005.

[18] N. Matsushita, D. Hihara, T. Ushiro, S. Yoshimura, J. Rekimoto, and Y. Yamamoto, "ID CAM: A smart camera for scene capturing and ID recognition," in *Proc. 2nd IEEE/ACM Int. Symp. Mixed Augmented Reality (ISMAR'03)*, Tokyo, Japan, Oct. 2003, pp. 227–236.

[19] N. Lepistö, B. Thörnberg, and M. O'Nils, "High-performance FPGA based camera architecture for range imaging," in *Proc. 23rd NORCHIP Conf.*, Nov. 2003, pp. 165–168.

[20] N. Blanc, T. Oggier, G. Gruener, J. Weingarten, A. Codourey, and P. Seitz, "Miniaturized smart cameras for 3D-imaging in real-time," in *Proc. IEEE Sensors 2004*, Vienna, Austria, Oct. 2004, pp. 471–474.

[21] U. Muehlmann, M. Ribo, P. Lang, and A. Pinz, "A new high speed CMOS camera for real-time tracking applications," in *Proc. 2004 IEEE Int. Conf. Robot. Automat.*, New Orleans, LA, April 2004, pp. 48–53.

[22] P. Chalimbaud and F. Berry, "Design of an imaging system based on FPGA technology and CMOS imager," in *Proc. 2004 IEEE Int. Conf. Field-Programmable Technology (ICFPT'04)*, Brisbane, Australia, Dec. 2006, pp. 407–411.

[23] F. Dias, P. Chalimbaud, F. Berry, J. Serot, and F. Marmoiton, "Embedded early vision systems: Implementation proposal and hardware architecture," in *Proc. Conf. Cogn. Sensors Interact. Sensros (COGIS 2006)*, Paris, France, Mar. 2006.

[24] T. E. Boult, R. C. Johnson, T. Pietre, R. Woodworth, and T. Zhang, "A decade of networked intelligent video surveillance," in *Proc. ACM Workshop Distrib. Camera Syst.*, 2006.

[25] I. B. Ozer, T. Lu, and W. Wolf, "Design of a real-time gesture recognition system," *IEEE Signal Process. Mag.*, vol. 22, no. 3, pp. 57–64, May 2005.

[26] W. Wolf, *Computers as Components: Principles of Embedded Computing System Design*. San Francisco, CA: Morgan Kaufmann, 2000.

[27] J. Schlessman, B. Ozer, K. Fujino, K. Itoh, and W. Wolf, "FPGA-based design of a surveillance system employing optical flow," in *Proc. Workshop Synthesis System Integr. Mixed Inf. Technol.*, 2006.

[28] B. D. Lucas and T. Kanade, "An iterative image registration technique with an application to stereo vision," in *Proc. IEEE Imag. Understand. Workshop*, 1981, pp. 121–130.

[29] C. Regazzoni, V. Ramesh, and G. L. Foresti, "Scanning the issue/technology," *Proc. IEEE (Special Issue on Video Communications, Processing, and Understanding for Third Generation Surveillance Systems)*, vol. 89, pp. 1355–1367, Oct. 2001.

[30] M. Valera and S. A. Velastin, "Intelligent distributed surveillance systems: A review," *Proc. Inst. Elect. Eng. Vision, Image Signal Process.*, vol. 152, no. 2, pp. 192–204, Apr. 2005.

[31] M. Leeser, S. Miller, and H. Yu, "Smart camera based on reconfigurable hardware enables diverse real-time applications," in *Proc. 12th Annu. IEEE Symp. Field-Program. Custom Comput. Mach. (FCCM'04)*, Napa, CA, Apr. 2004, pp. 147–155.

[32] C. H. Lin, T. Lv, W. Wolf, and I. B. Ozer, "A peer-to-peer architecture for distributed real-time gesture recognition," in *Proc. 2004 IEEE Int. Conf. Multimedia Expo (ICME 2004)*, Taipei, Taiwan, R.O.C., Jun. 2004, pp. 57–60.

[33] C. Wu and H. Aghajan, "Model-based human posture estimation for gesture analysis in an opportunistic fusion smart camera network," in *Proc. IEEE Int. Conf. Adv. Video Signal-based Surveil. (AVSS 2007)*, London, U.K., Sep. 2007.

[34] A. Rowe, C. Rosenberg, and I. Nourbakhsh, "A second low cost embedded color vision system," in *Proc. IEEE Workshop Embed. Comput. Vision (ECVW 2005)*, San Diego, CA, Jun. 2005, pp. 136–136.

[35] A. Yilmaz, O. Javed, and M. Shah, "Object tracking: A survey," *ACM Comput. Surv.*, vol. 38, no. 4, p. 45, Dec. 2006.

[36] D. Comaniciu and P. Meer, "Mean shift: A robust approach toward feature space analysis," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 24, pp. 603–619, May 2002.

[37] V. Boykov and D. Huttenlocher, "Adaptive Bayesian recognition in tracking rigid objects," in *Proc. IEEE Conf. Comput. Vision Pattern Recognit.*, 2000, pp. 697–704. IEEE.

[38] M. Isard and A. Blake, "Condensation-conditional density propagation for visual tracking," *Int. J. Comput. Vision*, vol. 29, no. 1, 1998.

[39] D. Comaniciu, V. Ramesh, and P. Meer, "Kernel-based object tracking," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 25, pp. 564–577, May 2003.

[40] Y. Ricquenbourg and P. Bouthemy, "Real-time tracking of. moving persons by exploiting spatiotemporal image slices," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 22, no. 8, pp. 797–808.

[41] T. Zhao, R. Nevatia, and F. Lv, "Segmentation and tracking of multiple humans in complex situations," in *Proc. IEEE Conf. Comput. Vision Pattern Recognit.*, Dec. 2001, vol. 2, pp. 8–14.

[42] D. Comaniciu, V. Ramesh, and P. Meer, "Real-time tracking of non-rigid objects using mean shift," in *Proc. IEEE Conf. Comput. Vision Pattern Recognit.*, Jun. 2000, vol. 2, pp. 142–149.

[43] Q. Cai and J. K. Aggarwal, "Tracking human motion in structured environments using a distributed camera system," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 21, pp. 1241–1247, Nov. 1999.

[44] O. Javed, K. Shafique, and M. Shah, "Appearance modeling for tracking in multiple non-overlapping cameras," in *Proc. IEEE Comput. Soc. Conf. Comput. Vision Pattern Recognit.*, 2005, vol. 2, pp. 26–33.

[45] V. Cevher, A. C. Sankaranarayanan, J. H. McClellan, and R. Chellappa, "Target tracking using a joint acoustic video system," *IEEE Trans. Multimedia*, vol. 9, pp. 715–727, Jun. 2007.

[46] S. Oh, S. Russell, and S. Sastry, "Markov chain Monte Carlo data association for general multiple-target tracking problems," in *Proc. 43rd IEEE Conf. Decision Contr.*, Paradise Island, Bahamas, Dec. 2004.

[47] J. Liu, M. Chu, J. Liu, J. Reich, and F. Zhao, "Distributed state representation for tracking problems in sensor networks," in *Proc. IEEE 3rd Int. Symp. Inf. Process. Sensor Netw. (IPSN 2004)*, 2004, pp. 234–242.

[48] R. T. Collins, A. J. Lipton, H. Fujiyoshi, and T. Kanade, "Algorithms for cooperative multisensor surveillance," *Proc. IEEE*, vol. 89, pp. 1456–1477, Oct. 2001.

[49] J. Mallett and V. M. Bove, Jr., "Eye society," in *Proc. IEEE ICME 2003*, 2003.

[50] S. Fleck, F. Busch, P. Biber, and W. Strasser, "3D surveillance—A distributed network of smart cameras for real-time tracking and its visualization in 3D," in *Proc. 2006 Conf. Comput. Vision Pattern Recognit. Workshop (CVPRW 2006)*, New York, Jun. 2006.

[51] M. Bramberger, M. Quaritsch, T. Winkler, B. Rinner, and H. Schwabach, "Integrating multi-camera tracking into a dynamic task allocation system for smart cameras," in *Proc. IEEE Conf. Adv. Video Signal Based Surveil. (AVSS 2005)*, Pisa, Italy, Sep. 2005, pp. 474–479.

[52] M. Quaritsch, M. Kreuzthaler, B. Rinner, H. Bischof, and B. Strobl, "Autonomous multicamera tracking on embedded smart cameras," *EURASIP J. Embed. Syst.*, 2007.

[53] S. Velipasalar, J. Schlessman, C.-Y. Chen, W. Wolf, and J. P. Singh, "SCCS: A scalable clustered camera system for multiple object tracking communicating via message passing interface," in *Proc. IEEE Int. Conf. Multimedia Expo 2006*, 2006.

[54] D. C. Schmidt, "Middleware for real-time and embedded systems," *Commun. ACM*, vol. 45, no. 6, pp. 43–48, Jun. 2002, ACM.

[55] M. M. Molla and S. I. Ahamed, "A survey of middleware for sensor networks and challanges," in *Proc. IEEE 2006 Int. Conf. Parallel Process. Workshops (ICPPW'06)*, Columbus, OH, Aug. 2006, pp. 223–228.

[56] Y. Yu, B. Krishnamachari, and V. K. Prasanna, "Issues in designing middleware for wireless sensor networks," *IEEE Network*, vol. 18, pp. 15–21, Jan./Feb. 2004.

[57] B. Rinner, M. Jovanovic, and M. Quaritsch, "Embedded middleware on distributed smart cameras," in *Proc. IEEE Int. Conf. Acoust., Speech, Signal Process. (ICASSP 2007)*, Honolulu, HI, Apr. 2007.

[58] M. Quaritsch, B. Rinner, and B. Strobl, "Improved agent-oriented middleware for distributed smart cameras," in *Proc. ACM/IEEE Int. Conf. Distrib. Smart Cameras (ICDSC 2007)*, Vienna, Austria, Sep. 2007, pp. 297–304.

[59] C. Micheloni, G. L. Foresti, and L. Snidaro, "A network of co-operative cameras for visual surveillance," *Proc. Inst. Elect. Eng. Vision, Image Signal Process.*, vol. 152, no. 2, pp. 205–212, Apr. 8, 2005.

[60] R. Pflugfelder and H. Bischof, "People tracking across two distant self-calibrated cameras," in *Proc. IEEE Int. Conf. Adv. Video Signal-Based Surveil. (AVSS 2007)*, London, U.K., Sep. 2007, p. 6.

[61] T. Chen, A. Del Bimbo, F. Pernici, and G. Serra, "Accurate self-calibration of two cameras by observations of a moving person on a ground plane," in *Proc. IEEE Int. Conf. Adv. Video Signal-Based Surveillance (AVSS 2007)*, London, U.K., Sep. 2007, p. 6.

[62] C. Niu and E. Grimson, "Recovering non-overlapping network topology using far-field vehicle tracking data," in *Proc. IEEE 18th Int. Conf. Pattern Recognit.*, 2006, vol. 4, pp. 944–949.

[63] B. Song and A. Roy-Chowdhury, "Stochastic adaptive tracking in a camera network," in *Proc. IEEE Int. Conf. Comput. Vision*, 2007.

[64] C.-Y. Chen, T.-M. Lin, and W. Wolf, "A visible/infrared fusion algorithm for

distributed smart cameras," *IEEE J. Sel. Topics Signal Process.*, vol. 2, no. 4, 2008.

[65] A. Klausner, A. Tengg, and B. Rinner, "Distributed multi-level data fusion for networked embedded systems," *IEEE J. Sel. Topics Signal Process.*, vol. 2, no. 4, 2008.

[66] A. Chattopadhyaya and T. Boult, "PrivacyCam: A privacy preserving camera using uCLinux on the BlackFin DSP," in *Proc. Workshop Embed. Comput. Vision (ECVW 2007)*, Minneapolis, MP, Jun. 2007.

[67] S. Fleck, R. Loy, C. Vollrath, F. Walter, and W. Strasser, "Smartclassysurv—A smart camera network for distributed tracking and activity recognition and its application to assisted living," in *Proc. ACM/IEEE Int. Conf. Distrib. Smart Cameras (ICDSC 2007)*, Vienna, Austria, Sep. 2007, pp. 109–116.

[68] A. Senior, S. Pankanti, A. Hampapur, L. Brown, Y.-L. Tian, A. Ekin, J. Connell, C. F. Shu, and M. Lu, "Enabling video privacy through computer vision," *IEEE Security Privacy*, vol. 3, no. 3, pp. 50–57, 2005.

[69] W. H. Widen, "Smart cameras and the right to privacy," *Proc. IEEE*, vol. 96, no. 10, pp. 1688–1697, Oct. 2008.

ABOUT THE AUTHORS

Bernhard Rinner (Senior Member, IEEE) received the M.Sc. and Ph.D. degrees in telematics from Graz University of Technology, Austria, in 1993 and 1996, respectively.

He is a Full Professor and Chair for Pervasive Computing at Klagenfurt University, Austria, where he is currently serving as Vice Dean of the Faculty of Technical Sciences. He held research positions with Graz University of Technology from 1993 to 2007 and with the Department of Computer Science, University of Texas at Austin, from 1998 to 1999. His research interests include parallel and distributed processing, embedded systems, and mobile and pervasive computing. He has authored or coauthored about 100 papers for journals, conferences, and workshops, led several research projects, and served as a Reviewer, Program Committee Member, Program Chair, and Editor-in-Chief.

Wayne Wolf (Fellow, IEEE) received the B.S., M.S., and Ph.D. degrees in electrical engineering from Stanford University, Stanford, CA, in 1980, 1981, and 1984, respectively.

He is Farmer Distinguished Chair and Georgia Research Alliance Eminient Scholar at Georgia Institute of Technology, Atlanta. He was with AT&T Bell Laboratories from 1984 to 1989. He was on the Faculty of Princeton University, Princeton, NJ, from 1989 to 2007. His research interests include embedded computing, embedded video and computer vision, and VLSI systems.

Prof. Wolf is a Fellow of ACM. He received the ASEE Terman Award and IEEE Circuits and Systems Society Education Award.