Distributed Embedded Smart Cameras for Surveillance Applications

Michael Bramberger, Andreas Doblander, Arnold Maier, and Bernhard Rinner
Graz University of Technology

Helmut Schwabach
Austrian Research Centers, Seibersdorf

Current high-performance embedded cameras combine video sensing, video processing, and communication within a single device. These smart cameras are key components for novel surveillance systems.

Recent advances in computing, communication, and sensor technology are pushing the development of many new applications. This trend is especially evident in pervasive computing, sensor networks, and embedded systems.

Smart cameras, one example of this innovation, are equipped with a high-performance onboard computing and communication infrastructure, combining video sensing, processing, and communications in a single embedded device. By providing access to many views through cooperation among individual cameras, networks of embedded cameras can potentially support more complex and challenging applications—including smart rooms, surveillance, tracking, and motion analysis—than a single camera.

Wayne Wolf and colleagues described the required computing and communication performance and the real-time and quality-of-service (QoS) requirements of the image-processing algorithms executed on a single embedded smart camera.[1] Their two-camera prototype uses a standard PC equipped with Tri-Media image-processing boards. Our work extends the smart camera concept even further.

We designed our smart camera as a fully embedded system, focusing on power consumption, QoS management, and limited resources. The camera is a scalable, embedded, high-performance, multiprocessor platform consisting of a network processor and a variable number of digital signal processors (DSPs). Using the implemented software framework, our embedded cameras offer system-level services such as dynamic load distribution and task reconfiguration. In addition, we combined several smart cameras to form a distributed embedded surveillance system that supports cooperation and communication among cameras.

Although smart cameras have various applications, we focus on traffic surveillance, which imposes demanding video-processing and compression-algorithm requirements on the camera's hardware and software. The "From Analog to Digital Smart Cameras" sidebar provides a discussion of the evolution of smart camera surveillance systems.[2]

To meet the requirements of an innovative traffic-surveillance system—that is, the ability to autonomously monitor the traffic along a highway section, computing traffic statistics, delivering a compressed live video to the monitoring station, and performing high-level video analysis such as detecting a wrong-way driver or an accident—the distributed surveillance architecture must be scalable and flexible.

To demonstrate our distributed surveillance system's feasibility, we developed a prototype implementation consisting of several smart cameras. We assigned video-based surveillance tasks to clusters of smart cameras and dynamically and autonomously mapped tasks to individual cameras. Both this task mapping and the tasks' QoS level autonomously adapt to the surveillance system's state and the available resources.

Published by the IEEE Computer Society
0018-9162/06/$20.00 © 2006 IEEE

From Analog to Digital Smart Cameras

Over the past two decades, surveillance systems have been an area of intense research. Recently, much research effort has focused on video-based surveillance systems, especially for public safety and transportation systems.[1,2]

Video-based surveillance systems have evolved in three generations. First-generation surveillance systems used analog equipment throughout the overall system. Analog closed-circuit television cameras captured the observed scene and transmitted the video signals over analog communication lines to the central back-end systems, which displayed and archived the video data.

Second-generation surveillance systems use digital back-end components, allowing real-time automated analysis of the incoming video data. Therefore, automated event detection and alarm generation significantly increased the amount of simultaneously monitored data and the overall surveillance system's quality.

Third-generation surveillance systems have completed the digital transformation. In these systems, the video signal is converted into the digital domain at the (digital) cameras, which transmit the video data via a computer network such as a local area network. The digital cameras can also directly compress the video data to save bandwidth. The back-end and transmission systems of third-generation surveillance systems have also increased their functionality. For example, they use intelligent hubs[3] to collect the video data, aggregate the information from different cameras, and transmit it to the video archive and the operators.

Intelligent cameras are a novelty in these surveillance systems. They perform a statically defined set of low-level image-processing operations on the captured frames onboard to improve the video compression and intelligent host efficiency.[4,5] However, most video processing and analysis in current surveillance systems is executed at a central host using standard workstation racks. For example, in traffic surveillance, typical video analysis tasks include video compression, detection of stationary vehicles and wrong-way drivers, and computation of traffic statistics such as average speed, lane occupancy, and vehicle classification.[6] The system designer statically assigns the analysis of a camera's video input to a specific workstation. Modifying or reconfiguring this assignment during the surveillance system's operation is difficult.

Current processor technology allows the implementation of smart cameras,[7] which directly perform highly sophisticated video analysis. These smart cameras integrate video sensing, video processing, and communication into a single embedded device; they're designed as reconfigurable and flexible processing nodes with self-reconfiguration, self-monitoring, and self-diagnosis capabilities.

Smart cameras support the ongoing paradigm shift from a central to a distributed control surveillance system. The main motivation for this shift is increasing the surveillance system's functionality, availability, and autonomy. Smart cameras are key components of these novel surveillance systems because they provide sufficient performance for onboard video processing and distributed control. Such surveillance systems can react autonomously to changes in the system's environment and to detected events in the monitored scenes.

A static surveillance system configuration is no longer feasible; the system architecture must support reconfiguration, migration, quality of service, and power adaptation of analysis tasks.

References

1. *Proc. IEEE*, special issue on video communications, processing, and understanding for third-generation surveillance systems, C.S. Regazzoni, V. Ramesh, and G.L. Foresti, eds., Oct. 2001.
2. G.L. Foresti et al., "Active Video-Based Surveillance System: The Low-Level Image and Video Processing Techniques Needed for Implementation," *IEEE Signal Processing*, Mar. 2005, pp. 25-37.
3. L.F. Marcenaro et al., "Distributed Architectures and Logical-Task Decomposition in Multimedia Systems," *Proc. IEEE*, Oct. 2001, pp. 1419-1440.
4. W. Caarls, P.P. Jonker, and H. Corporaal, "SmartCam: Devices for Embedded Intelligent Cameras," *Proc. 3rd Progress Workshop on Embedded Systems*, CD-ROM, STW, 2002.
5. N. Matsushita et al., "ID CAM: A Smart Camera for Scene Capturing and ID Recognition," *Proc. 2nd IEEE and ACM Int'l Symp. Mixed and Augmented Reality*, ACM Press, 2003, pp. 227-236.
6. V. Kastrinaki, M. Zervakis, and K. Kalaitzakis, "A Survey of Video Processing Techniques for Traffic Surveillance," *Image and Vision Computing*, vol. 21, no. 4, 2003, pp. 359-381.
7. W. Wolf, B. Ozer, and T. Lv, "Smart Cameras as Embedded Systems," *Computer*, Sept. 2002, pp. 48-53.

SYSTEM ARCHITECTURE

Smart cameras are a core component of future traffic-surveillance systems. High-level computing and communication capabilities in the smart cameras' embedded platform increase surveillance system functionality, flexibility, and scalability. For example, basic video compression (MPEG-4 advanced simple profile) and video processing (such as stationary vehicle detection) require an overall computation performance of about 10 billion instructions per second (GIPS) and a transfer rate of about 1 megabyte per second (Mbps). Additional onboard video analysis increases the computation and communication requirements.

We chose a commercial off-the-shelf software/hardware architecture to support fast prototype development and achieve flexibility and performance at a reasonable price.

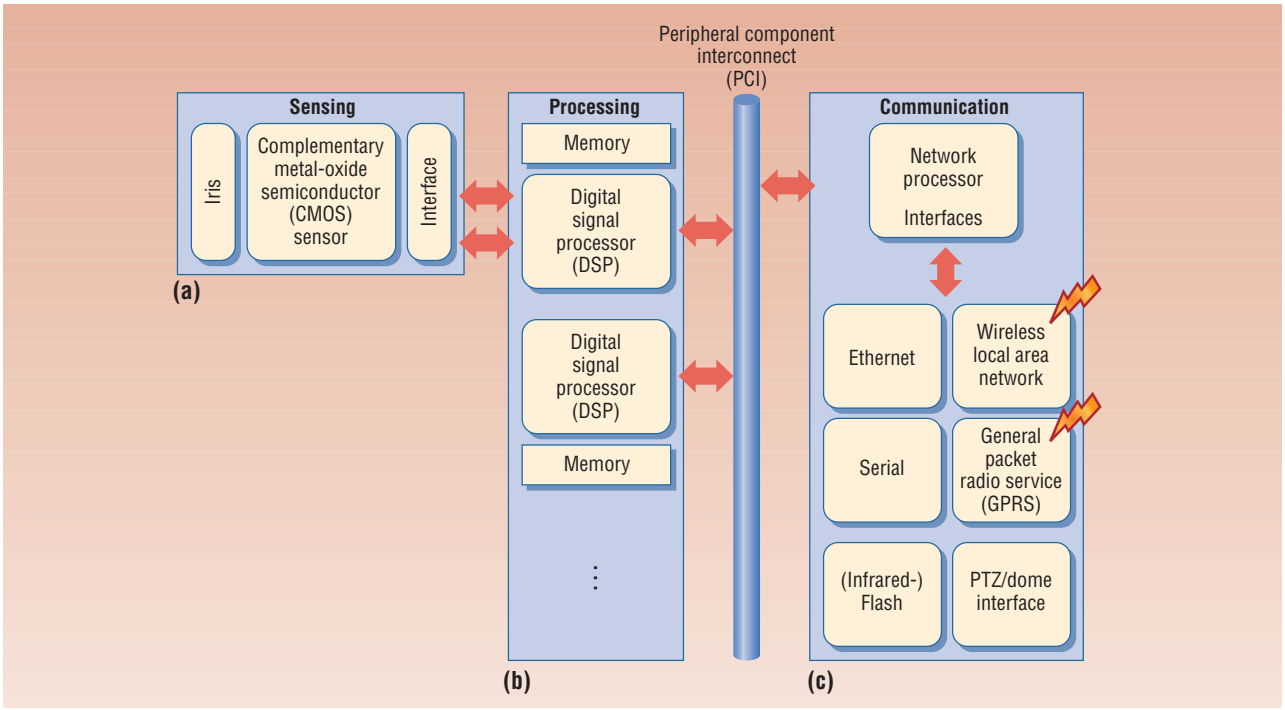

Figure 1. Scalable smart camera hardware architecture. The hardware architecture consists of three main parts: (a) sensing unit, (b) processing unit, and (c) communication unit.

Hardware architecture

Figure 1 depicts the smart camera's three main parts: the sensing unit, processing unit, and communication unit.

A highly dynamic, monochrome complementary metal-oxide semiconductor (CMOS) image sensor is the sensing unit's heart. The sensing unit delivers images with VGA resolution at up to 30 frames per second, transferring the captured images via a first-in, first-out (FIFO) memory to the processing unit.

The processing unit consists of DSPs, which offer a good compromise between performance, power consumption, and flexibility. Up to 10 Texas Instruments TMS320C64x DSPs can deliver an aggregate performance of up to 80 GIPS while keeping the power consumption low. By using an adequate number of DSPs, we adapt this scalable architecture's computing performance to the requirements of the real-time video analysis and compression tasks targeted for the smart camera.

A local peripheral component interconnect (PCI) bus couples the DSPs and connects them to the network processor (Intel XScale IXP425). The network processor establishes the connection between the processing and communication units and controls internal and external communication. We chose the XScale processor because of its communication capabilities, low power consumption, and integration of various interfaces. Internal communication between the DSPs or between the DSPs and the network processor occurs through the PCI bus.

The communication unit currently supports two interfaces for IP-based external communication: wired Ethernet and wireless Global System for Mobile Communications/general packet radio service (GSM/GPRS).

Software architecture

We designed the smart camera's software architecture for flexibility and reconfigurability. It consists of several layers, which we group into two frameworks:

- the DSP framework, running on every DSP in the system, and
- the SmartCam framework, running on the network processor.

We based the architecture on the abstraction that the application logic runs on the network processor and loads and unloads the actual analysis algorithms onto the DSPs as needed. Figure 2 is an overview of our smart camera's software architecture.

DSP framework. The DSP framework's main purposes are to

- provide an abstraction of the hardware and communication channels,
- support dynamic loading and unloading of application tasks, and
- manage the DSP's on-chip and off-chip resources.

Of course, only the DSP connected to the image sen-

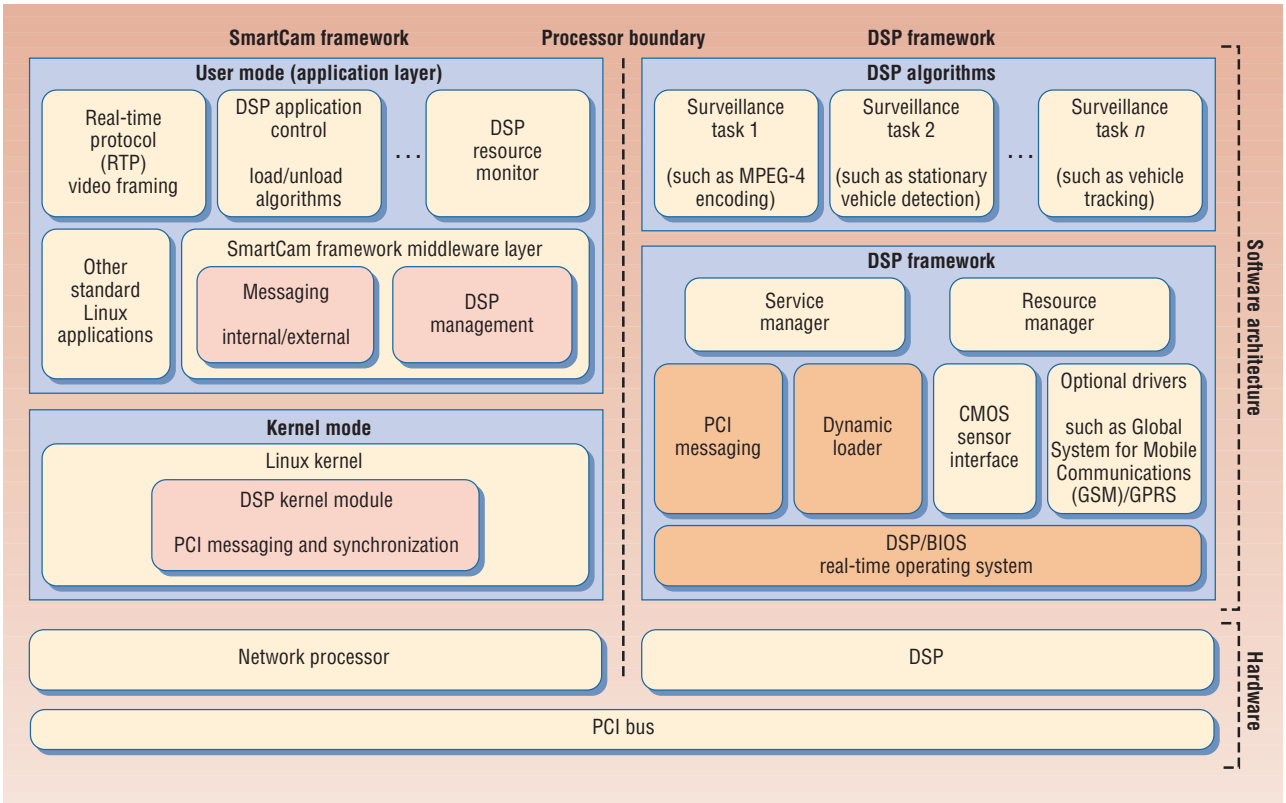

Figure 2. Smart camera software architecture. The overall architecture consists of the SmartCam framework (left) and the DSP framework (right).

sor needs the sensor interface module. Algorithms on different DSPs use the service-management facilities to dynamically establish connections to each other. This is important because the dynamic loader module can load and unload all algorithms at runtime. Actually, only the shaded modules on the left in the DSP framework in Figure 2 must be available at startup. The system can dynamically load all other components at runtime. We built the DSP framework on Texas Instruments' DSP/BIOS operating system.

SmartCam framework. The SmartCam framework, illustrated in Figure 2, serves two main purposes:

- It provides an abstraction of the DSPs to ensure the application layer's platform independence.
- The application layer uses the provided communication methods—that is, internal messaging to the DSPs and external IP-based communication—to exchange information or offer data-relay services for DSP algorithms.

Modules of this part of the software architecture support application development by providing high-level interfaces to DSP algorithms and the DSP framework's functions.

The XScale processor runs standard Linux, easing application development and providing many modules for seamless internal and external communication on our smart camera. The only customization of the Linux kernel is the DSP kernel module, which the processor uses to establish the connection to the DSPs via the PCI bus.

Distributed system architecture

Developers can use our embedded smart cameras to implement a distributed intelligent video-surveillance (IVS) system consisting of dozens of cameras.

To design a scalable distributed architecture, we partition an IVS into distributed logical groups of typically collocated smart cameras, or *surveillance clusters*. It isn't necessary to assign most traffic-surveillance tasks—such as accident detection, vehicle classification, and computation of traffic statistics—to a specific camera. Our IVS architecture only requires an assignment to a specific cluster. The IVS then dynamically and autonomously maps surveillance tasks onto individual cameras depending on the cameras' available resources and the system's current state.

We implement this dynamic reconfiguration of tasks onto cameras using a mobile agent system (MAS) built atop the SmartCam framework. In our MAS, agents represent surveillance tasks that can migrate dynamically between the cluster's cameras. Significant changes in the observed environment or in the available resources trigger a task migration. A dedicated agent in our MAS uses

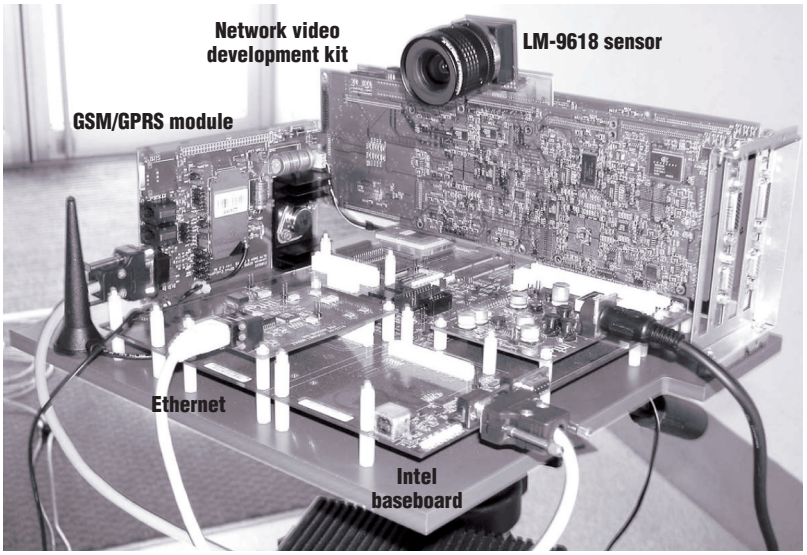

Figure 3. SmartCam prototype. This COTS-based prototype consists of a baseboard equipped with an IXP425 XScale network processor, two TMS320C6416 PCI boards, a CMOS image sensor, and a GSM/GPRS module.

a complex cost function to compute the optimal assignment of tasks to cameras.[3]

QoS is a major concern in a distributed IVS. In video-based surveillance, typical QoS parameters include frame rate, transfer delay, image resolution, and video-compression rate.[4] The surveillance tasks might also offer several QoS levels. Furthermore, the provided QoS levels can change over time due to user interactions or changes in the monitored environment. Thus, novel IVS systems must include dedicated QoS management mechanisms.

Power awareness is another important design aspect in distributed IVS systems because they must deliver high-level QoS while using embedded devices that are partly solar or battery powered. Our smart camera supports combined power and QoS management (PoQoS)[5] for distributed IVS systems. PoQoS dynamically configures the power and QoS level of the camera's hardware and software to adapt to user requests and changes in the environment.

SMARTCAM PROTOTYPE

We've developed a prototype SmartCam using COTS hardware components to test and evaluate our video-surveillance system. The prototype shown in Figure 3 demonstrates our approach's feasibility. It provides the required computing and communication performance, but not the desired form factor and reliability for real-world use in harsh environments.

A single SmartCam consists of a network processor, several DSPs, and a CMOS image sensor.

Hardware platform

Intel's IXDP425 development board serves as the camera platform. The baseboard is equipped with an Intel IXP425 XScale network processor running at 533 MHz, with 256 Mbytes of external memory and four PCI slots. The IXP425 provides on-chip support for Ethernet access, multiple serial ports, and, most importantly, an on-chip PCI host controller.

We use Ateme's network video development kits (NVDK) as the DSP platform. These PCI boards, which are plugged into the baseboard, consist of Texas Instruments TMS320C6416 DSPs running at 600 MHz. Each NVDK board contains 264 Mbytes of memory accessible via two different DSP external memory interfaces.

Eastman Kodak's monochrome sensor LM-9618 captures the images in our SmartCam prototype. The sensor provides a high-dynamic range of up to 110 decibels at VGA resolution. FIFO memory connects the sensor to one of the DSPs.

System software

Linux (Kernel 2.6.8.1) operates the network processor, allowing access to a broad variety of open source software modules. The SmartCam framework, which executes atop Linux, ensures interoperability with the DSPs. Java also runs atop Linux, supporting platform-wide applications.

The DSP/BIOS real-time operating system operates the DSPs. The DSP framework runs atop this RTOS and serves as the SmartCam framework's counterpart on the network processor.

Distributed software

We use mobile agents to support the development of our distributed surveillance system, consisting of loosely coupled smart cameras. Mobile agents are most suitable for this distributed application because we can encapsulate each surveillance task within a mobile agent, which can then migrate between cameras. An MAS supports autonomous operation of the surveillance tasks. Moreover, this approach is highly scalable and flexible.

In a typical MAS, individual agents migrate between hosts, which execute the surveillance tasks. We've thus extended the mobile agent to keep a DSP binary that can be downloaded from the SmartCam's network processor to one of the DSPs.

The DSP agents have three parts:

- a module that manages the agent's integration into its environment,
- a DSP binary representing the agent's functionality, and
- an optional set of intermediate data.

DSP agents further comprise a set of DSP resource requirements. The task allocation mechanism requires these parameters to autonomously allocate surveillance tasks to smart cameras.

SmartCam agents perform status information and communication tasks. They are executed on the network processor and can access the DSPs, but they don't include resource requirements or DSP binaries.

We've implemented additional agents that provide system functionality, such as in the task-allocation system. The system exploits mobile SmartCam agents to determine in a distributed manner how to optimally allocate surveillance tasks to the cluster's SmartCams.[3]

EXPERIMENTAL RESULTS

We used two identical SmartCam prototypes for evaluation and integrated up to three additional PCs (Pentium III running under Linux at 1 GHz) into our experimental setup to evaluate larger SmartCam networks. This integration was rather easy because the complete SmartCam framework and the MAS could execute on the PC without any modification.

We used diet agents (http://diet-agents.sourceforge.net) running under Java as the MAS and applied the JamVM Java virtual machine on the smart camera prototype.

In our first experiments, we compared the SmartCam prototype's Java performance with that of a standard PC. The results showed that the interpreter-based JamVM is about 20 times slower than the Sun Java runtime environment (JRE) 1.4.2 (exploiting a just-in-time compiler) on the PCs. Note that the native computing performance between a Pentium III PC and the SmartCam (XScale) differs only by a factor of two.

Table 1 lists the most important hardware and software parameters of a SmartCam equipped with two DSPs. Because the SmartCam prototype uses COTS components, the amount of memory and, consequently, the power dissipation, are higher than the design would require.

To evaluate and demonstrate the SmartCam hardware, software, and distributed system architecture, we implemented a multicamera, object-tracking application.

In our tracking approach, the multicamera system instantiates only a single tracker task. This tracker (agent) follows the tracked object, migrating to the SmartCam that should next observe the object. We based the tracking agent on a Kanade-Lucas-Tomasi feature tracker.[6] The tracker's main advantage is its short initialization time, which makes it applicable for multicamera object tracking by mobile agents. Tracking agents control the handover process, using predefined migration regions in the observed scenes. When the tracked object enters a migration region, the tracker initiates handover to the next SmartCam.

We assign each migration region to one or more possible next SmartCams. Motion vectors help distinguish among several SmartCams assigned to the same migration region. We also use these motion vectors to check whether the object moves in the correct direction. The migration regions and the motion vectors represent the spatial relationship among the cameras. In many traffic-surveillance scenarios, this spatial relationship is rather simple, with only a single succeeding camera.

Table 1. Smart camera prototype's key features.

Parameter	Value
Hardware-related	
Processing power	9,600 MIPS
Onboard memory	784 Mbytes
Internal data transfer rate	100 Mbps
Estimated power dissipation	35 watts
Software-related	
Basic Linux footprint	2.5 Mbytes
Complete Linux system (including Java and the multiagent system)	20 Mbytes
Static DSP framework footprint	494 Kbytes

The colored polygons in Figure 4 represent migration regions. We use a master-slave approach for the tracked object handover. The master tracker identifies the object in its field of view and tracks its positions (top left image in Figure 4). As soon as the object enters a migration region, the master tracker creates slave trackers on every smart camera assigned to that migration region (center images in Figure 4). The master tracker initializes these slave trackers with the object's identified features. When the slave tracker identifies the tracked object in its field of view, it terminates the master tracker and other slave trackers, and becomes the master tracker.

Our experiments show that a tracking agent's migration between SmartCams takes up to 1 second. The migration time is an important parameter for multicamera tracking, limiting both maximum vehicle speed and minimum camera distance.

The overall migration time also includes the task-allocation system's setup time—approximately 190 milliseconds. Invoking task allocation provides the resources for the tracking agent on the SmartCam. Although the migration times are rather long—basically caused by the SmartCam Java implementation—the master-slave architecture significantly reduces the influence of migration times and network bandwidth on the handover procedure between smart cameras. However, the master-slave approach increases resource utilization because two or more trackers are active at the same time.

Because tracking agents don't reside on the same smart camera for long, the task-allocation system manages them as temporary agents. Therefore, the task-allocation system doesn't try to reconfigure the surveillance cluster but reduces the agents' QoS level on the smart camera, which is a fast way to free resources.

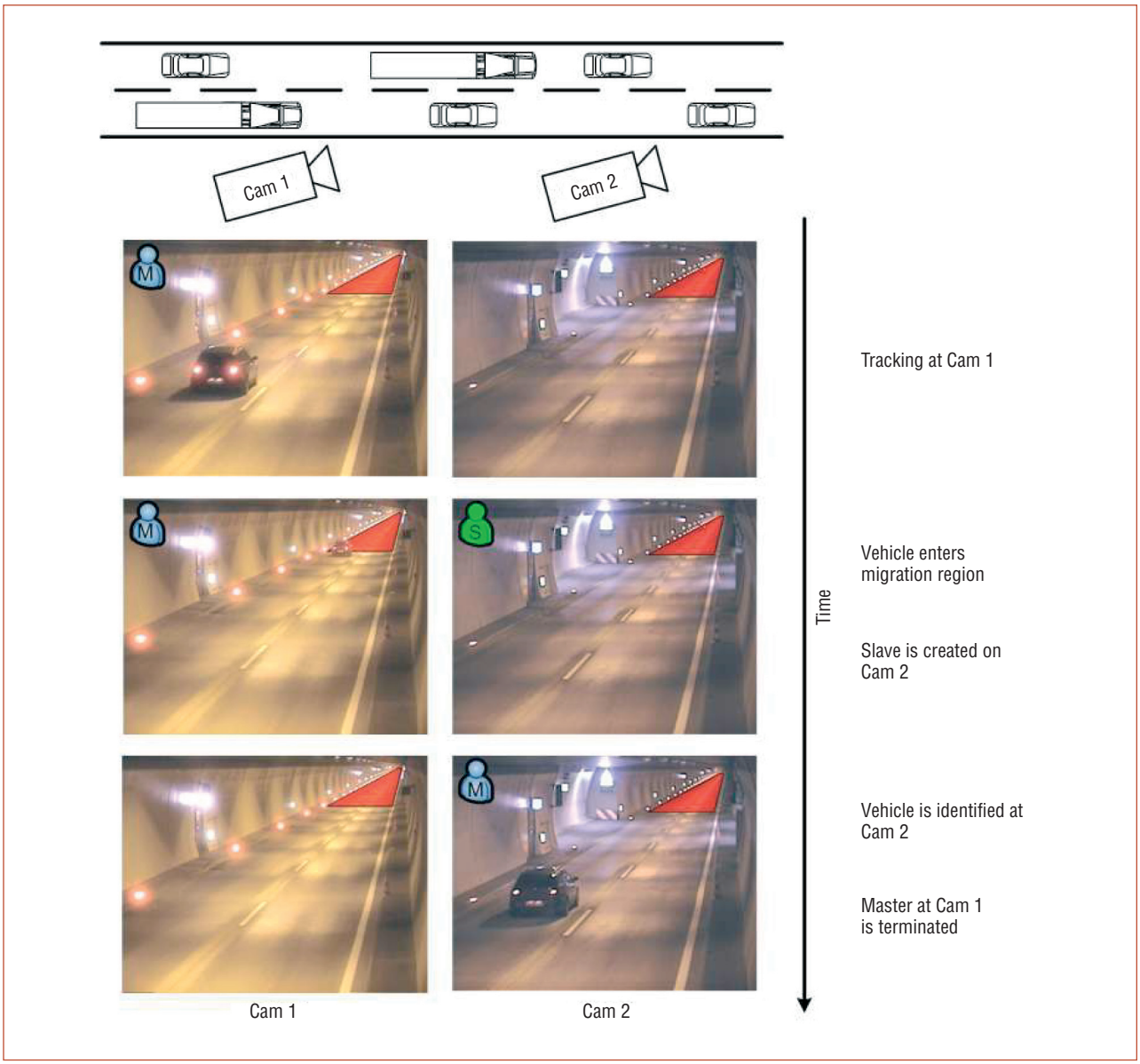

Figure 4. Tracking a vehicle between succeeding smart cameras in a tunnel. The tracked vehicle's handover at two adjacent cameras (Cams 1 and 2) is based on a migration region (red areas). When the vehicle enters this region at Cam 1, the master tracker initiates a slave tracker at Cam 2. When the slave tracker identifies the vehicle, it terminates the master tracker and becomes the new master tracker.

Developing the SmartCam prototype has given us insight that might also be applicable to other distributed embedded systems. Our experiences show that the keys to successful deployment of smart cameras are

- the integration of sensing, computing, and communication in a small, power-aware embedded device;
- the availability of high-level image/video processing algorithms or libraries for the embedded target processors (the DSPs);
- a lightweight software framework supporting glueless intra- and intercamera communication; and
- the availability of various system-level services such as task mapping and QoS adaptation to allow autonomous and dynamic operation of the overall multicamera system.

Embedded smart cameras could potentially be deployed in applications such as smart environments, intelligent infrastructures, and pervasive computing. Augmenting the smart cameras with additional sensors could transform them into a high-performance multisensor system. By combining visual, acoustic, tactile, or location-based information, the smart cameras become more sensitive and can deliver more accurate results, making them even more widely applicable.

Acknowledgments

We performed this work at the Institute for Technical Informatics, Graz University of Technology. We acknowledge the support received from Texas Instruments.

References

1. W. Wolf, B. Ozer, and T. Lv, "Smart Cameras as Embedded Systems," *Computer*, Sept. 2002, pp. 48-53.
2. G.L. Foresti, C. Mahonen, and C.S. Regazzoni, *Multimedia Video-Based Surveillance Systems*, Kluwer Academic Publishers, 2000.
3. M. Bramberger, B. Rinner, and H. Schwabach, "A Method for Dynamic Allocation of Tasks in Clusters of Embedded Smart Cameras," *Proc. Int'l Conf. Systems, Man and Cybernetics*, IEEE Press, 2005, pp. 2595-2600.
4. R. Steinmetz and K. Nahrstedt, *Multimedia Systems*, Springer, 2004.
5. A. Maier, B. Rinner, and H. Schwabach, "A Hierarchical Approach for Energy-Aware Distributed Embedded Intelligent Video Surveillance," *Proc. IEEE/IFIP Int'l Workshop Parallel and Distributed Embedded Systems*, IEEE Press, 2005, pp. 12-16.
6. J. Shi and C. Tomasi, "Good Features to Track," *Proc. IEEE Int'l Conf. Computer Vision and Pattern Recognition*, IEEE Press, 1994, pp. 593-600.

Michael Bramberger *is a consultant for infotainment systems in the automotive industry. His research interests focus on distributed dynamic task allocation in embedded multiprocessor systems. Bramberger received a PhD in telematics from Graz University of Technology. He is a member of the IEEE. Contact him at bramberger@iti.tugraz.at.*

Andreas Doblander *is a PhD candidate at the Institute for Technical Informatics at Graz University of Technology. His research interests include middleware for smart embedded DSP systems, software fault tolerance, and model-based software development for embedded systems. Doblander received an MS in electrical engineering from Graz University of Technology. Contact him at doblander@iti.tugraz.at.*

Arnold Maier *is a PhD candidate at the Institute for Technical Informatics at the Graz University of Technology. His research focuses on dynamic power-aware reconfiguration methods for distributed embedded smart systems. Maier received an MSc in telematics from Graz University of Technology. He is a student member of the IEEE. Contact him at maier@iti.tugraz.at.*

Bernhard Rinner *is an associate professor in the Department of Electrical Engineering and Information Technology at Graz University of Technology. His research interests include parallel and distributed processing, embedded systems, and mobile and pervasive computing. Rinner received a PhD in telematics from Graz University of Technology. He is member of the IEEE, the AAAI, and Telematik Ingenieurverband, Austria. Contact him at b.rinner@computer.org.*

Helmut Schwabach *works on business development for embedded vision systems in the Austrian Research Centers, Seibersdorf. His research interests include innovations in embedded vision systems for surveillance and intelligent traffic applications. Contact him at helmut.schwabach@arcs.ac.at.*