Hindawi Publishing Corporation
EURASIP Journal on Embedded Systems
Volume 2007, Article ID 92827, 10 pages
doi:10.1155/2007/92827

Research Article

Autonomous Multicamera Tracking on Embedded Smart Cameras

Markus Quaritsch,[1] Markus Kreuzthaler,[1] Bernhard Rinner,[1] Horst Bischof,[2] and Bernhard Strobl[3]

[1] *Institute for Technical Informatics, Graz University of Technology, 8010 Graz, Austria*
[2] *Institute for Computer Graphics and Vision, Graz University of Technology, 8010 Graz, Austria*
[3] *Video and Safety Technology, Austrian Research Centers GmbH, 1220 Wien, Austria*

Received 28 April 2006; Revised 19 September 2006; Accepted 30 October 2006

Recommended by Udo Kebschull

There is currently a strong trend towards the deployment of advanced computer vision methods on embedded systems. This deployment is very challenging since embedded platforms often provide limited resources such as computing performance, memory, and power. In this paper we present a multicamera tracking method on distributed, embedded smart cameras. Smart cameras combine video sensing, processing, and communication on a single embedded device which is equipped with a multiprocessor computation and communication infrastructure. Our multicamera tracking approach focuses on a fully decentralized handover procedure between adjacent cameras. The basic idea is to initiate a single tracking instance in the multicamera system for each object of interest. The tracker follows the supervised object over the camera network, migrating to the camera which observes the object. Thus, no central coordination is required resulting in an autonomous and scalable tracking approach. We have fully implemented this novel multicamera tracking approach on our embedded smart cameras. Tracking is achieved by the well-known CamShift algorithm; the handover procedure is realized using a mobile agent system available on the smart camera network. Our approach has been successfully evaluated on tracking persons at our campus.

1. INTRODUCTION

Computer vision plays an important role in many applications ranging from industrial automation over robotics to smart environments. There is further a strong trend towards the implementation of advanced computer vision methods on embedded systems. However, deployment of advanced vision methods on embedded platforms is challenging, since these platforms often provide only limited resources such as computing performance, memory, and power.

This paper reports on the development of computer vision methods on a distributed embedded system, that is, on tracking objects across multiple cameras. We focus on autonomous multicamera tracking on distributed, embedded smart cameras [1]. Smart cameras are equipped with a high-performance onboard computing and communication infrastructure and combine video sensing, processing, and communication in a single embedded device [2]. Networks of such smart cameras [3] can potentially support more complex vision applications than a single camera by providing access to many views and by cooperation among the cameras. For single-camera tracking, a tracker or tracking agent is responsible for detecting, identifying, and tracking objects over time from the video stream delivered from a single camera. The basic idea of multicamera tracking is that the tracking agent follows the object over the camera network, that is, the agent has to migrate to the camera that should next observe the object. In such a scenario, the handover of the tracking agent from one camera to the next is crucial.

Our multicamera tracking approach is intended as an additional service of a surveillance system. Tracking an object is started on demand for a particular object of interest on the camera observing the object. This implies that there are only few objects of interest within the supervised area. Our approach is appropriate for large-scale camera networks due to the decentralized handover and it is also applicable for sparse camera setups where the cameras have no or little overlapping fields of view.

We have developed an autonomous handover process requiring no central coordination. The handover is managed only by the adjacent cameras. Thus, our approach is scalable, which is a very important feature for distributed applications. Currently, single-camera tracking is based on the well-known CamShift algorithm [4]. The handover mechanism is realized using a mobile agent system available at our smart cameras. Our approach has been completely implemented on our embedded smart cameras and tested on tracking persons at our campus. This research significantly extends our previous work on multicamera tracking. In [5] we have basically evaluated handover strategies on PC-based smart camera prototypes. The behavior of the trackers has only been simulated on the smart camera prototypes.

The remainder of this paper is organized as follows. Section 2 discusses some related work. Section 3 introduces the distributed embedded smart cameras used for this project. It presents the hardware and software architectures as well as the mobile agent framework. Section 4 describes our multicamera tracking approach. We first discuss the tracking requirements and present an overview of visual tracking methods. We then focus on the implemented CamShift algorithm and the handover mechanism. Section 5 presents the implementation on our smart cameras and Section 6 describes the experimental results. Section 7 concludes this paper with a short summary and a discussion about future work.

2. RELATED WORK

There exist several projects which also focus on the integration of image acquisition and image processing in a single embedded device. Heyrman et al. describe in [6] the architecture of a smart camera which integrates a CMOS sensor, processor, and reconfigurable unit in a single chip. The presented camera is designed for high-speed image processing using dedicated parts such as sensors with massively parallel outputs and region-of-interest readout circuits. However, a single-chip solution is not as scalable and flexible as a modular design.

Rowe et al. [7] promote a low-cost embedded vision system. The aim of this project is the development of a small camera with integrated image processing. Due to the very limited memory and computing resources, only low-level image processing such as threshold and filtering is possible. The image processing algorithm cannot be modified after deployment since it is integrated in the firmware of the processor.

Tracking objects on a single smart camera or a network of cameras is also an interesting research topic. Micheloni et al. [8] depict a network of cooperative cameras for visual surveillance. A set of static camera systems with overlapping fields of view is used to monitor the surveilled area and maintains the trajectory of all objects simultaneously. Active camera systems use PTZ cameras for close-up recordings of an object. A static camera can request an active camera to follow an object of interest. In this case, both camera systems track the position of the object cooperatively.

In [9], Fleck and Straßer demonstrate a particle-based algorithm for tracking objects in the field of view of a single camera. They used a commercially available camera which is comprised of a CCD image sensor, a Xilinx FPGA for low-level image processing, and a Motorola PowerPC CPU. In [10], Fleck et al. present a multicamera tracking implementation using the particle-based tracking algorithm. In this implementation each camera tracks all moving objects and transmits the obtained position of each object to a central server node.

In both approaches [8, 10], object tracking is the main task of the camera network. Each camera executes the tracking algorithm even if there is no object within the field of view. This is significantly different to our multicamera tracking approach since we consider tracking as an additional task of the network. Tracking is only loaded and executed on demand for individual objects. The tracking instance then acts autonomously and follows the target over the camera network.

The handover of an object between cameras in [10] is performed by a central server. In [8] no detailed information about the handover procedure is given. Our solution avoids a central node for coordinating the handover from one camera to the next. Instead, neighborhood relations between cameras are exploited resulting in a fully decentralized handover.

Velipasalar et al. describe in [11] a PC-based decentralized multicamera system for multiobject tracking using a peer-to-peer infrastructure. Each camera identifies moving objects and follows their track. When a new object is identified, the camera issues a labeling request containing a description of the object. If the object is known by another camera, it replies the label of the object; otherwise a new label is assigned, which results in a consistent labeling over multiple cameras.

Agent systems have also been used for multicamera video surveillance applications. Remagnino et al. [12] describe the usage of agents in visual surveillance systems. An agent-based framework is used to accomplish scene understanding. Abreu et al. present Monitorix [13], a video-based multiagent traffic surveillance system. In both approaches, agents are used as an additional layer of abstraction. We also use agents as an abstraction of different surveillance tasks but we also exploit mobility of agents in order to achieve a dynamically reconfigurable system. Moreover, we deploy the mobile agent system on our embedded platform while in [12, 13] a PC-based implementation is used.

3. THE SMART CAMERA PLATFORM

Smart cameras are the core components of future video surveillance systems. These cameras not only capture images but also perform high-level video analysis and video compression. Video analysis tasks include motion detection, object recognition, and classification. To fulfill these requirements, the smart camera has to provide sufficient computing power for analyzing video data.

The software operating the smart camera has to be flexible and dynamically reconfigurable. Hence, it has to be possible to load and unload the image processing algorithms dynamically at run time. This allows to build a flexible and fault tolerant surveillance system.

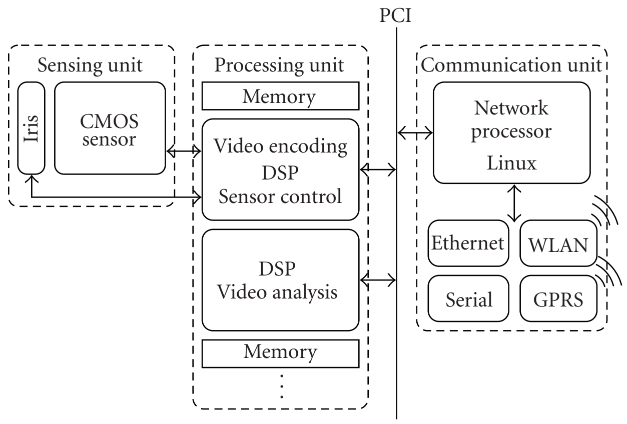


Figure 1: The hardware architecture of the smart camera.

3.1. Hardware architecture

The hardware platform has to provide the computing power required by the image processing tasks and also provide different ways to communicate with the outside world. Further design issues are to provide scalable computing power while minimizing the power consumption.

The hardware architecture of our smart camera can be grouped into three main units: (1) the sensing unit, (2) the processing unit, and (3) the communication unit [2]. Figure 1 depicts the three main units along with their top-level modules and communication channels.

The core of the sensing unit is a high-dynamic range CMOS image sensor. The image sensor delivers images up to VGA resolution at 25 frames per second via a FIFO memory to the processing unit.

Real-time video analysis and compression is performed by the processing unit which utilizes multiple digital signal processors (DSPs). In the default configuration the smart camera is equipped with two DSPs offering about 10 to 15 GIPS.[1] The number of DSPs in the processing unit is scalable and basically limited by the communication unit. Up to four DSPs can be connected without additional hardware effort but it is also possible to use up to ten DSPs. The DSPs are coupled via a PCI bus which also connects them to the communication unit.

The communication unit has two main tasks. First, it manages the internal communication between the DSPs as well as the communication between the DSPs and the communication unit. Second, it provides communication channels to the outside world. These communication channels are usually IP-based and include standard Ethernet and wireless LAN. The main component of the communication unit is an ARM-based network processor which is operated by a standard Linux system.

[1] Giga instructions per second.

3.2. Software architecture

The software architecture also reflects the partitioning into a processing unit and a communication unit. The *DSP framework* running on each DSP provides an environment for the video processing tasks and introduces a layer of abstraction as well. The *SmartCam framework* resides on the network processor and manages the communication on the smart camera [14].

3.2.1. DSP framework

The main tasks of the DSP framework are to (1) support dynamic loading and unloading of DSP applications, (2) manage the available resources, and (3) provide data services for the DSP applications. Figure 2(a) sketches the architecture of the DSP framework.

Exploiting dynamically loadable applications allows to launch different video processing tasks depending on the current requirements and context. This results in more flexible smart cameras which also makes the whole surveillance system more flexible. The integration of dynamically loadable DSP applications introduces the need of an extended resource management which is capable of dealing with the dynamic use of resources. Data services provide uniform access for the DSP applications to the data sources and data sinks available on the smart camera.

3.2.2. SmartCam framework

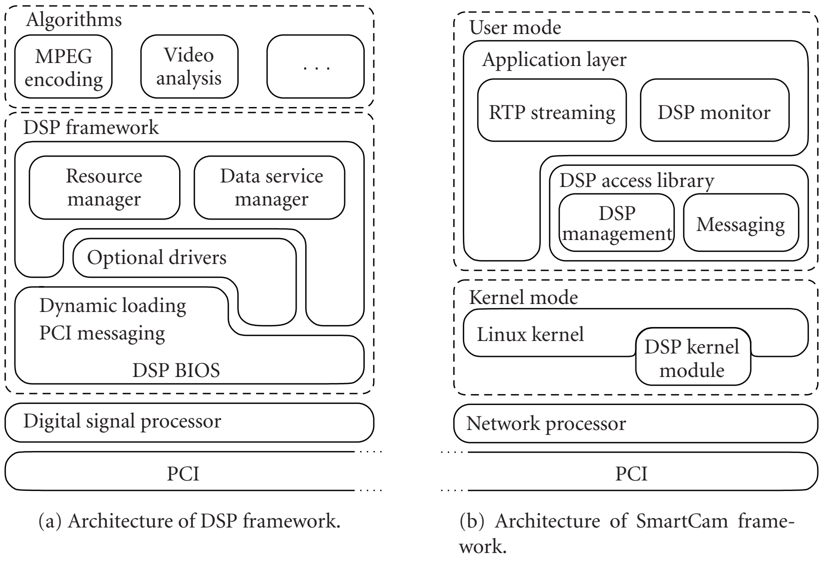
The *SmartCam framework* is executed on the network processor. On the one hand, this framework manages the low-level interprocessor communication. On the other hand, it allows applications running on the network processor to interact with the DSPs. Hence, the SmartCam framework is divided into two layers: (1) the kernel-mode layer, and (2) the user-mode layer. Figure 2(b) depicts these layers along with their main components.

The kernel-mode layer is implemented as a kernel module which builds the base of the SmartCam framework. This layer has direct access to the PCI bus and thus accomplishes the management of interprocessor communication. Additionally, this layer offers a low-level interface which allows user-space programs to communicate with the DSPs.

The user-mode layer is based on the kernel-mode layer and provides a DSP access library (DSPLib). This library interacts with the kernel module and provides a simplified interface for sending and receiving messages. Applications executing on the network processor can use this layer to load and unload dynamic executables to the DSPs.

3.3. Mobile agent framework

The mobile agent framework [15] is the highest level of abstraction in our smart cameras. Each video processing task is represented by an instance of a mobile agent. Each agent acts autonomously and carries out the required actions in order to fulfill its mission. Two different types of agents are



(a) Architecture of DSP framework.

(b) Architecture of SmartCam framework.

Figure 2: The software architecture of the smart camera.

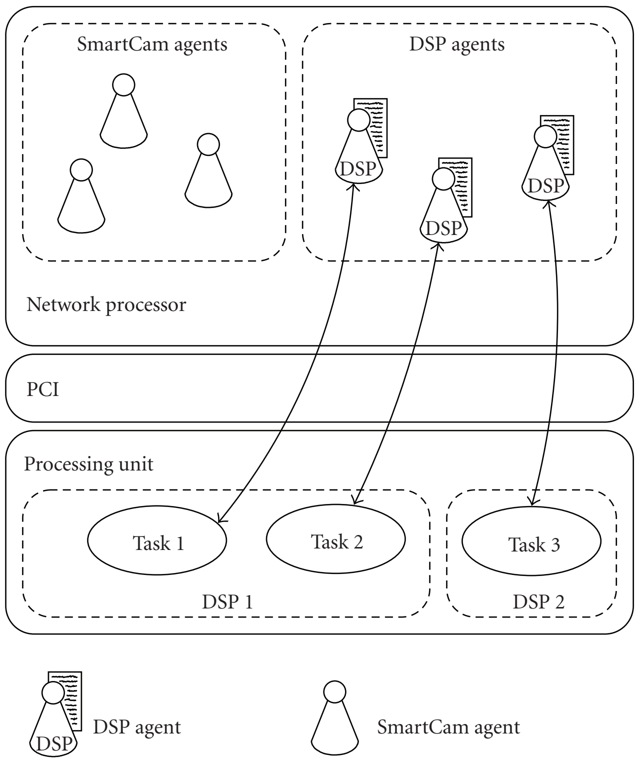


Figure 3: An agency hosting DSP agents and SmartCam agents.

available on our smart cameras: (1) DSP agents, and (2) SmartCam agents. Figure 3 shows an agency hosting both types of agents.

DSP agents are used to represent video processing tasks. This type of agent has a tight relation to the DSPs as their main mission—analyzing the video data—is executed on the DSP. The agent contains the DSP executable and is responsible for starting, initializing, and stopping the DSP application as required. The agent also knows how to interact with the DSP application in order to obtain the information required for further actions. Using DSP agents enables to move video processing tasks dynamically from one smart camera to another. In contrast to this, SmartCam agents do not interact with the DSPs. Usually they perform control and management tasks.

The mobile agent framework is executed on the network processor. Each smart camera hosts an agency which provides the environment for the mobile agents. The agency further contains a set of system agents which provide services for the DSP agents and SmartCam agents. The DSPLibAgent, for example, provides an interface to the DSPs of the processing unit for the DSP agents. Other agents contain information about the location and configuration of the current smart camera as well as information about its actual internal state.

Employing mobile agents allows to dynamically reconfigure the entire surveillance system at run time. This reconfiguration is usually performed autonomously by the agents and helps to better utilize the available resources of the surveillance system [16]. We use mobile agents to realize the handover mechanism in our multicamera tracking approach.

4. MULTICAMERA TRACKING

Our approach for multicamera tracking focuses on autonomous and decentralized object tracking. Since the tracker is executed on the DSP, it is implemented as a DSP agent in our framework. The tracking algorithm running on the DSP reports its agent only abstract information about the object of interest such as the current position and the trajectory. The agent uses this information to take further actions.

If, for example, the tracked object is about to leave the camera's field of view, the agent has to take care to track the object on the adjacent cameras.

4.1. Tracking requirements

Using this autonomous and decentralized approach for tracking an object among several cameras introduces some requirements for the tracking algorithm. Most of these requirements are a consequence of loading the tracking algorithm dynamically as needed. The main issues are the following.

Short initialization time

Because the tracking algorithm is loaded only when needed, the algorithm must not require a long initialization time (e.g., for generating a background model).

Internal state of the tracker

When migrating the tracking agent from one camera to the next, the current internal state of the tracking algorithm must be stored and transferred, too. The internal state usually contains the description of the tracked object such as templates or appearance models. During setup on the new camera, the tracking task must be able to initialize itself from a previously saved state.

Robustness

The tracking algorithm has to be robust not only with respect to the position of an object in a continuous video stream but also to identify the same object on the next camera. The object may appear differently due to the position and orientation of the camera.

4.2. Visual tracking

Visual tracking involves the detection and extraction of objects from a video stream and their continuous tracking over time to form persistent object trajectories. Visual tracking is a well-studied problem in computer vision with a wide variety of applications, for example, visual surveillance, robotics, autonomous vehicles, human-machine interfaces, or augmented reality. The main requirement and challenge for a tracking algorithm is a robust and stable behavior, very often real-time (20–30 fps) behavior is required. The tracking task is complicated due to the potential variability of the object over time.

There are numerous different approaches that have been developed for visual tracking. Template tracking methods, for example, [17, 18] are based on a template of the object that is redetected by correlation measures. More sophisticated methods take into account the object deformations and illumination changes. Appearance-based methods are related to template tracking but they build a parameterized model of the objects appearance in the scene [19], for example. In a similar spirit, active shape-based trackers build models of the object that is to be tracked based on the object's shape. There are trackers that use 3D models of the objects to be tracked [20]. Other tracking methods based on motion blobs do not need any model of the object. The idea is to detect moving objects by motion segmentation and then track the obtained blobs; for a typical example, see [21]. Another class of tracking algorithms is based on features. Probably the best-known feature tracker is the KLT tracker [22] which is based on tracking corner features that can be well localized in images and reliably tracked using a correlation measure. Another class of popular tracking methods is based on color. The well-known mean-shift algorithm [23] uses color distributions for tracking the object. Related is the CamShift algorithm (continuously adaptive mean-shift) [4] that updates the color distribution of the object while tracking. Very recently methods that use classifiers for tracking have been proposed [24, 25]. The idea is to use a very fast classification algorithm to detect the previously trained object of interest.

Taking the requirements listed in Section 4.1 into account, the CamShift algorithm was chosen to demonstrate the feasibility of the presented tracking approach.

4.3. CamShift algorithm

The continuously adaptive mean-shift algorithm [4], or CamShift algorithm, is a generalization of the mean-shift algorithm [23]. CamShift operates on a color probability distribution image produced from histogram back-projection. It is designed for dynamically changing distributions. These occur when objects in video sequences are being tracked and the object moves so that the size and location of the probability distribution change over time. The CamShift algorithm adjusts the search window size in the course of its operation. For each video frame, the color probability distribution image is tracked and the center and size of the color object are found via the CamShift algorithm. The current size and location of the tracked object are reported and used to set the size and location of the search window in the next video image. The process is then repeated for continuous tracking. Instead of a fixed—or externally adapted—window size, CamShift relies on the zeroth moment information, extracted as part of the internal workings of the algorithm, to continuously adapt its windows within or over each video frame. The main steps of the CamShift algorithm are (for more details see [4]) the following:

(1) choose an initial 2D location of the 2D search window;
(2) calculate the color probability distribution in a region slightly larger than the search window;
(3) run the mean-shift algorithm to find the center of the search window;
(4) for the next frame, center the search window at the location found in the mean-shift iteration;
(5) calculate the 2D orientation using second moments.

CamShift has been used successfully for a variety of tracking tasks. In particular for tracking skin-colored regions, for example, faces and hands.

4.4. Handover mechanism

In order to extend tracking from single isolated cameras to multiple cameras, a handover process is necessary. The handover of a tracker from one camera to the next requires the following steps:

(1) select the "next" camera(s);
(2) migrate the tracking agent to the next camera(s);
(3) initialize the tracking task;
(4) redetect the object of interest;
(5) continue tracking.

In order to identify potential next cameras for the handover, we exploit the a priori known neighborhood relations of the smart camera network. Tracking agents control the handover process by using predefined migration regions in the observed scenes. The migration region is defined by a polygon in the 2D image space and a motion vector. Each migration region is assigned to one or more next smart cameras. Motion vectors help to distinguish among several smart cameras assigned to the same migration region.

The migration regions and their assigned cameras represent the spatial relationship among the cameras. All information about the migration region is managed locally by the SceneInformationAgent, a system agent present on each smart camera. When the tracked object enters a migration region and the trajectory matches the motion vector of the migration region, the tracking agent initializes the handover to the assigned adjacent camera(s).

The next two steps of the handover process (migration and initialization) are implicitly managed by our mobile agent system. The color model of the tracked object is included as local data to the tracking agent. The (migrated) tracking agent uses this local data for the initialization on the new camera. Object redetection and tracking are then continued on the new camera.

Master/slave handover

The tracking agent may use different strategies for the handover [5]. The approach presented in this paper follows the master/slave paradigm. Figure 4 shows the handover procedure along with the instances of tracking agents for a sample scenario of two consecutive cameras. During the handover, there exist two instances of a tracking agent dedicated to one object of interest. As *master* tracking agent, we denote the agent which currently tracks the object. When the object enters a migration region, the master agent creates a slave on the neighboring cameras. The master also queries the current description of the object from the tracking algorithm and transfers it to the slave. The slave in turn starts the DSP application and initializes the tracking algorithm with the information received from the master. The slave is now waiting for the object to appear. When the object enters the field of view of the slave, the roles of the tracking agents change. The slave becomes the master as it observes and tracks the target now. The new master notifies the old master that the target is now in its field of view, whereupon the old master terminates itself.

This approach is also feasible, if a camera has more than one neighbor for the same migration region. In this case, the master creates a slave on all adjacent cameras. When a slave notifies the master that it has detected the target object, the master instructs all other slaves to terminate, too.

The information required for initializing the tracking algorithm on the next camera heavily depends on the tracking algorithm. In the case of the CamShift algorithm, only the description of the object to track is used, which is obtained from the algorithm itself and contains the color histogram of the object.

The color variations of an object observed by different cameras is another issue which has to be taken into account when using a color-based tracking algorithm. The same object may appear in a slightly different color when captured by another camera due to variations in illumination, changes in the angle of view, and variations of the image sensor. Therefore, the SceneInformationAgent contains a color-correction table. The tracking agent passes this information to the tracking task during initialization. The color correction is obtained during an initialization of the surveillance system.

5. IMPLEMENTATION

5.1. Hardware setup

In order to evaluate our approach in practice, two similar prototypes of smart cameras were used. The first prototype consists of an *Intel IXDP425 development board* which is equipped with an *Intel IXP425* network processor running at 533 MHz. For the processing unit two *Network Video Development Kit (NVDK)* from ATEME are used. Each board is comprised of a TMS320C6416 DSP from Texas Instruments running at 600 MHz with a total of 264 MB of onboard memory. Images are captured using the *Eastman Kodak LM9628* color CMOS image sensor which is connected to one of the DSP boards. The second prototype uses an *Intel PXA255 Evaluation Board* from Kontron. All other components are the same as in the first prototype.

5.2. Software implementation

The operating system used for the network processor is based on a standard GNU/Linux distribution for embedded systems using kernel version 2.6.17.

For the prototype implementation, we have selected a Java-based mobile agent system due to its platform independence. We use the DIET-Agents platform as mobile agent framework (see http://diet-agents.sf.net) which provides all required features to support mobility and autonomy. Moreover, it is reasonably small and thus it is also applicable for embedded systems.

For the Java virtual machine, version 1.3.0 of JamVM (see http://jamvm.sourceforge.net/) with GNU classpath version 0.14 was used. This virtual machine is also rather small and thus suitable for use in an embedded system. However, JamVM does not feature a just-in-time compiler but only interprets the Java bytecode. This results in longer execution

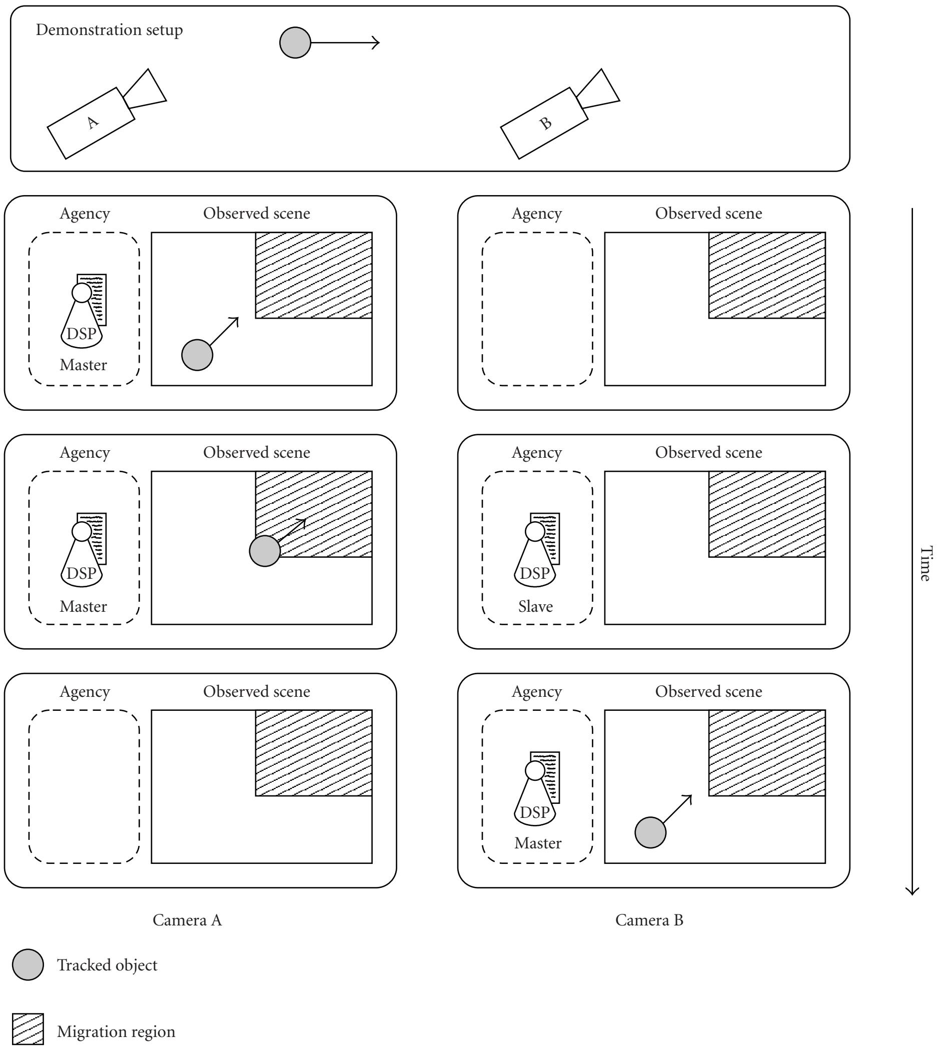


FIGURE 4: Master/slave handover strategy.

times. Of course, the execution times would be dramatically reduced when exploiting a just-in-time compiler, but currently there are no implementations available which can be used on our ARM-based prototypes.

The CamShift tracking algorithm has been implemented and optimized for the DSP platform used in our prototypes. Furthermore, the necessary extensions for multicamera tracking have also been implemented.

6. EXPERIMENTAL RESULTS

The experimental setup for evaluating our autonomous multicamera tracking approach consists of two smart camera prototypes as described in Section 5. Figure 5 depicts the prototype of our smart camera.

The first part of the evaluation addresses the implementation of the CamShift tracking algorithm, while the second part focuses on the handover procedure for multicamera tracking as well as the integration of the tracking algorithm into the agent system.

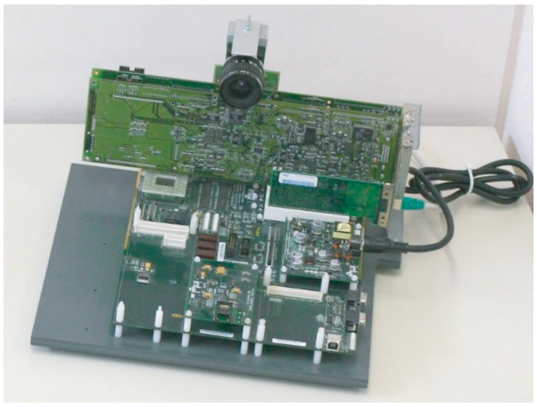


FIGURE 5: The Intel XScale-based prototype.

Table 1: Characteristics of the CamShift algorithm.

Code size (dynamically executable)	9 kB
Memory	300 kB
Internal state for migration	256 bytes
Initialize color-histogram	< 10 ms per frame
Identify tracked object	< 1 ms per frame

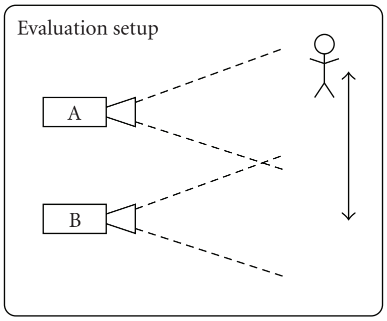


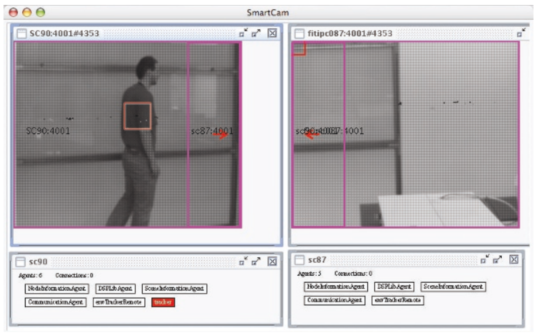
Figure 6: Outline of the camera setup for person tracking.

6.1. *CamShift implementation*

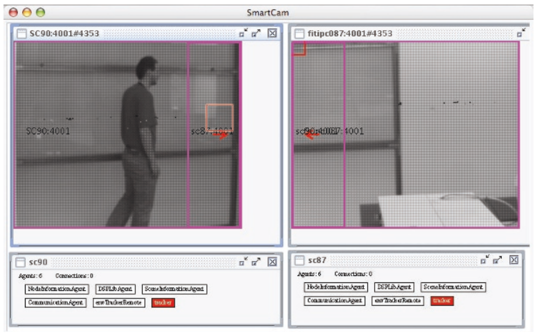
The evaluation of the CamShift tracking algorithm focuses on the resource requirements and the achieved performance of our implementation. Table 1 summarizes the results.

The memory requirements of the tracking algorithm depend on the resolution of the acquired images. In our experimental setup, we used images in CIF-resolution which results in a memory usage of about 300 kB (double buffered image plus an additional mask). The code size of the dynamic DSP executable is about 9 kB. The internal state of the tracker consists of the color histogram of the tracked object along with the position and size of the search window in image space. When migrating the tracking algorithm from one camera to another, only the color histogram has to be transmitted, which requires 256 Bytes.

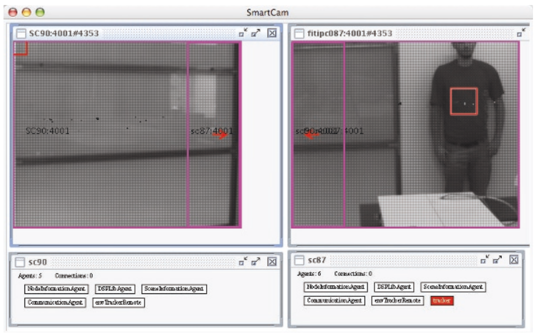
Initializing the algorithm to track a concrete object requires less than 10 ms per frame for calculating the color histogram. In our implementation, the color histogram used for tracking the object is the average of five consecutive frames. Tracking the object in a video stream requires less than 1 ms for obtaining the new position of the object in an image.



(a) Tracker on camera A.



(b) Handover to camera B: the person is in the migration region (red square).



(c) Tracker on camera B.

Figure 7: Visualizer. The left column of the window visualizes camera A while the right part shows camera B. The center of the tracked person is highlighted by the red square. Note that the acquired image of a camera and the current position of the person are updated at different rates. Hence, in image (b) the highlighted position is correct while the background image is inaccurate.

6.2. *Multicamera setup*

To demonstrate the feasibility of our autonomous multicamera tracking approach, we used a setup for tracking persons in our laboratory. Figure 6 sketches our evaluation configuration. The fields of view of both cameras overlap, but this is due to spatial constraints and not a requirement of the tracker. The tracking instance is created on camera A. The tracking algorithm learns the description of the target within a given initialization region provided by the agent and starts tracking the position of the person. Before the person walks out of the field of view, it enters the migration region. This triggers the migration of the tracking agent to camera B where the agent continues tracking the person.

Figures 7(a)–7(c) show the visualizer running on a PC during the handover. The left column is dedicated to camera A and the right one to camera B. In the upper part, the current images acquired by the cameras overlaid with the defined migration regions are displayed. The center of the tracked person is illustrated by the red square. Below, the agents residing on the cameras are outlined whereas the tracking agents are highlighted in red.

Table 2: Evaluation of the handover time.

Loading dynamic executable	0.18 s
Initializing tracking algorithm (5 frames at 20 fps)	0.25 s
Creating slave on neighboring camera	2.13 s
Reinitializing tracking algorithm on slave camera	0.04 s
Total	2.60 s

Table 3: Handover with multiple neighboring cameras.

Number of neighbors	*Time to create slaves*
1	2.60 s
2	3.03 s
3	3.51 s

Evaluating the handover procedure, the four major time intervals have been quantified. Table 2 enlists the obtained results.

Starting the tracking algorithm from a DSP agent requires 180 milliseconds. This includes loading the dynamic executable to the DSP, starting the tracking algorithm, and reporting the agent that the tracking algorithm is ready to run.

When the tracked object enters the migration region, it takes about 2.6 seconds to create the slave agent on the next camera and launch the tracking algorithm on the DSP. A large portion of this time interval (about 2.1 seconds) is required for creating the slave agent. This time penalty is a consequence of the Java virtual machine used which only interprets the bytecode instead of using a just-in-time compiler. Creating a new agent further uses Java reflections, which has a negative impact on the performance. Initializing the tracking algorithm by the slave agent using the information obtained from the master agent takes 40 milliseconds which is negligible compared to the time required for creating the slave agent. We have also evaluated the migration times between two PCs without loading the tracking algorithm using Sun's virtual machine. In this scenario, it takes about 75 milliseconds to move an agent from one host to the other.

To show the scalability of our approach, we have also conducted experiments where a camera has more than one neighbor. Due to the lack of additional embedded smart cameras, two additional PCs (PIII, 1 GHz) have been used. These PCs have no cameras attached but they host an agency where the tracking agents can migrate to. When the person enters the migration region, a slave is created on the next camera and also on each PC. When the slave on the next camera detects the person, it notifies its master, which in turn terminates the other slaves and itself. We have evaluated the time required for creating the slaves depending on the number of adjacent cameras. Table 3 shows that the time required to create the slaves is linearly dependent on the number of slaves. Hence the slaves are created in parallel, the required time equals the largest time interval for creating a single slave. The linear factor is introduced by the limited performance of the agent system on our embedded platform initiating the creation of the slaves.

7. CONCLUSION

In this paper, we have presented our novel multicamera tracking approach implemented on embedded smart cameras. The tracker follows the tracked object, migrating to the smart camera that should next observe the object. The spatial relationships among cameras are exploited by migration regions augmented in the cameras' image space. This results in a decentralized handover process which in turn is important for high autonomy and scalability.

On the one hand, mobile agents introduce a level of abstraction which eases the development of distributed applications. Communication and code migration are implicitly handled by the agent system. On the other hand, mobile agents require additional resources. Especially, the Java-based implementation causes a significant performance penalty on our embedded platform. Note that this penalty is not inherent of the mobile agent systems. It is primarily caused by the lack of an efficient virtual machine for our platform.

Future work includes (1) replacing the Java-based agent system by a more efficient (middleware) system providing services for data and code migration, (2) implementing color-adaptation schemes during tracker initialization in order to compensate color variations between different cameras, and (3) deploying our tracking approach on larger networks of cameras.

REFERENCES

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

[2] 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, 2006.

[3] B. Rinner and W. Wolf, Eds., *Proceedings of the Workshop on Distributed Smart Cameras (DSC '06)*, Boulder, Colo, USA, October 2006.

[4] G. R. Bradski, "Computer vision face tracking for use in a perceptual user interface," *Intel Technology Journal*, vol. 2, no. 2, p. 15, 1998.

[5] 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 *Proceedings of IEEE Conference on Advanced Video and Signal Based Surveillance (AVSS '05)*, pp. 474–479, Como, Italy, September 2005.

[6] B. Heyrman, M. Paindavoine, R. Schmit, L. Letellier, and T. Collette, "Smart camera design for intensive embedded computing," *Real-Time Imaging*, vol. 11, no. 4, pp. 282–289, 2005.

[7] A. Rowe, C. Rosenberg, and I. Nourbakhsh, "A second generation low cost embedded color vision system," in *Proceedings of IEEE Embedded Computer Vision Workshop (ECVW '05) in conjunction with IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR '05)*, vol. 3, p. 136, San Diego, Calif, USA, June 2005.

[8] C. Micheloni, G. L. Foresti, and L. Snidaro, "A network of cooperative cameras for visual surveillance," *IEE Proceedings: Vision, Image and Signal Processing*, vol. 152, no. 2, pp. 205–212, 2005.

[9] S. Fleck and W. Straßer, "Adaptive probabilistic tracking embedded in a smart camera," in *Proceedings of IEEE Embedded Computer Vision Workshop (ECVW '05) in conjunction with IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR '05)*, vol. 3, p. 134, San Diego, Calif, USA, June 2005.

[10] 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 *Proceedings of IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR '06)*, p. 118, New York, NY, USA, June 2006.

[11] 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 *Proceedings of IEEE International Conference on Multimedia and Expo*, pp. 277–280, Toronto, ON, Canada, July 2006.

[12] P. Remagnino, J. Orwell, D. Greenhill, G. A. Jones, and L. Marchesotti, "An agent society for scene interpretation," in *Multimedia Video Based Surveillance Systems: Requirements, Issues and Solutions*, pp. 108–117, Kluwer Academic, Boston, Mass, USA, 2001.

[13] B. Abreu, L. Botelho, A. Cavallaro, et al., "Video-based multi-agent traffic surveillance system," in *Proceedings of IEEE Intelligent Vehicles Symposium (IV '00)*, pp. 457–462, Dearbon, Mich, USA, October 2000.

[14] A. Doblander, B. Rinner, N. Trenkwalder, and A. Zoufal, "A middleware framework for dynamic reconfiguration and component composition in embedded smart cameras," *WSEAS Transactions on Computers*, vol. 5, no. 3, pp. 574–581, 2006.

[15] N. M. Karnik and A. R. Tripathi, "Design issues in mobile agent programming systems," *IEEE Concurrency*, vol. 6, no. 3, pp. 52–61, 1998.

[16] M. Bramberger, B. Rinner, and H. Schwabach, "A method for dynamic allocation of tasks in clusters of embedded smart cameras," in *Proceedings of IEEE International Conference on Systems, Man and Cybernetics (SMC '05)*, vol. 3, pp. 2595–2600, Waikoloa, Hawaii, USA, October 2005.

[17] F. Jurie and M. Dhome, "Real time robust template matching," in *Proceedings of the British Machine Vision Conference (BMVC '02)*, pp. 123–132, Cardiff, UK, September 2002.

[18] G. D. Hager and P. N. Belhumeur, "Efficient region tracking with parametric models of geometry and illumination," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 20, no. 10, pp. 1025–1039, 1998.

[19] M. J. Black and A. D. Jepson, "Eigentracking: robust matching and tracking of articulated objects using a view-based representation," *International Journal of Computer Vision*, vol. 26, no. 1, pp. 63–84, 1998.

[20] D. Koller, K. Daniilidis, and H. H. Nagel, "Model-based object tracking in monocular image sequences of road traffic scenes," *International Journal of Computer Vision*, vol. 10, no. 3, pp. 257–281, 1993.

[21] C. Beleznai, B. Frühstück, and H. Bischof, "Human detection in groups using a fast mean shift procedure," in *Proceedings of International Conference on Image Processing (ICIP '04)*, vol. 1, pp. 349–352, Singapore, October 2004.

[22] J. Shi and C. Tomasi, "Good features to track," in *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR '94)*, pp. 593–600, Seattle, Wash, USA, June 1994.

[23] D. Comaniciu, V. Ramesh, and P. Meer, "Real-time tracking of non-rigid objects using mean shift," in *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR '00)*, vol. 2, pp. 142–149, Hilton Head Island, SC, USA, June 2000.

[24] S. Avidan, "Support vector tracking," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 26, no. 8, pp. 1064–1072, 2004.

[25] O. Williams, A. Blake, and R. Cipolla, "Sparse Bayesian learning for efficient visual tracking," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 27, no. 8, pp. 1292–1304, 2005.