Decentralized Object Tracking in a Network of Embedded Smart Cameras

M. Quaritsch, M. Kreuzthaler, B. Rinner
Institute for Technical Informatics
Graz University of Technology, AUSTRIA
{quaritsch, kreuzthaler, rinner}@iti.tugraz.at

B. Strobl
Video & Safety Technology
ARC Seibersdorf research, AUSTRIA
bernhard.strobl@arcs.ac.at

Abstract

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 decentralized solution for tracking objects across multiple embedded smart cameras. Smart cameras combine video sensing, processing and communication on a single embedded device which is equipped with a multi-processor computation and communication infrastructure. Tracking an object within the multi-camera system is done by a single tracking instance which follows the target by migrating to the camera which observes the object next. Our multi-camera tracking approach focuses on a fully decentralized handover between adjacent cameras. Having no central coordination results in an autonomous and scalable tracking method.

We have fully implemented this novel multi-camera 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. For visualization, the smart cameras send the observed scene and tracking results to a separate PC. Our approach has been successfully evaluated on tracking people at our campus.

Keywords: embedded systems, distributed smart cameras, tracking, mobile agent systems

1 Introduction

Surveillance systems tend towards automating the process of observing the supervised area and identifying suspicious events. Therefore, advanced computer vision methods are required. A common approach is to integrate image acquisition and image processing in a single embedded device.

This paper reports on the development of computer vision methods on a distributed embedded system, i.e., on tracking objects across multiple smart cameras [1]. Smart cameras are equipped with a high-performance on-board computing and communication infrastructure and combine video sensing, processing and communication in a single embedded device [2]. Networks of such smart cameras can potentially support more complex vision applications than a single camera by providing access to many views and by cooperation among the cameras.

The basic idea of our multi-camera tracking solution is to employ a single instance of a tracker which follows the object of interest whereas the tracker resides on the camera observing the object. This means that only a single camera has to perform the tracking task while all other cameras are not affected. A tracking instance consists of the tracking algorithm which tracks the object in the video stream and a mobile agent which is responsible for following the object among the camera network. This means, the agent has to migrate conjointly with the tracking algorithm to the camera that should next observe the object when it is about to leave the viewport of the current camera. In such a scenario, the handover of the tracking agent from one camera to the next is crucial. The concrete algorithm used for identifying the object of interest and obtaining the position within a single video stream is exchangeable and can be tuned for special needs.

We have developed a fully decentralized handover procedure where the handover is realized autonomously by adjacent cameras. The handover procedure uses a master/slave approach whereas the master is the instance which observes the object and during the handover, one or more slaves are created on the neighboring cameras waiting for the object to appear. Neighborhood relations are stored locally on each camera, i.e., each camera knows the adjacent cameras in a certain direction. Thus, our approach is scalable which is a very important feature for distributed applications. Single camera tracking is based on the well-known CamShift algorithm [3]. We have completely implemented the presented tracking method on our embedded smart cameras and tested on tracking people at our campus. The position of the tracked person can be visualized on a separate PC showing the current scene and highlighting the tracked person.

2 Related Work

There exist several projects which also focus on the integration of image acquisition and image processing in a single embedded device. In [4], Fleck and Straßer present a particle filter 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. They also implemented a multi-camera tracking [5] using the particle filter tracking algorithm. However,

in this work, the handover between cameras is managed by a central server node.

Velipasalar et al. describe in [6] a PC based decentralized multi-camera system for multi-object 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.

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 limited memory and computing resources, only low-level image processing like threshold and filtering are possible. The image processing algorithm cannot be modified during runtime because it is integrated into the processor's firmware.

Agent systems have also been used as form of abstraction in multi-camera applications. Remagnino et al. [8] describe the usage of agents in visual surveillance systems. An agent-based framework is used to accomplish scene understanding. Abreu et al. present Monitorix [9], a video-based multi-agent traffic surveillance system based on PCs. Agents are used as representatives in different layers of abstraction.

3 The Smart Camera Platform

The smart cameras used in our work are composed of a heterogeneous multiprocessor systems [2]. This approach combines high processing power, dynamic reconfiguruability and flexible communication channels with low power consumption.

3.1 Hardware Architecture

The hardware architecture of our smart cameras is depicted in figure 1. It shows the three main units, which are (1) the sensing unit, (2) the processing unit, and (3) the communication unit. Images are acquired by the sensing unit which is comprised of a high-dynamic CMOS sensor. The images are delivered to the processing unit via a FIFO memory. The processing unit utilizes multiple digital signal processors (DSPs) for real-time image analysis and video compression. The number of DSPs in the processing unit is scalable and basically limited by the communication unit. In the default configuration the smart camera is equipped with two DSPs. The DSPs are coupled via a PCI bus which also connects them to the communication unit. The main component of the communication unit is a general purpose network processor. Its main tasks are: (1) managing the internal communication between the DSPs as well as the communication unit and the DSPs, and (2) providing communication channels to the outside world. These communication channels are usually IP-based and include standard Ethernet and wireless LAN.

3.2 Software Architecture

The software architecture is based on the abstraction, that the application logic runs on the network processor and loads and unloads the required image analysis tasks to the processing unit as needed.

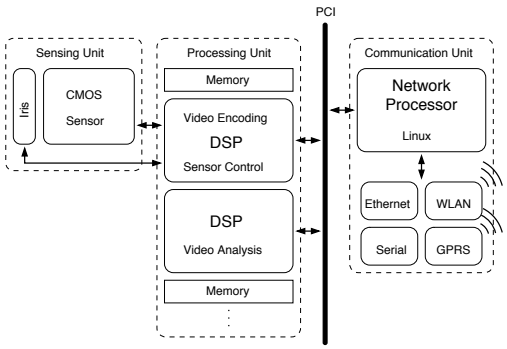

Figure 1. The hardware architecture of the smart camera.

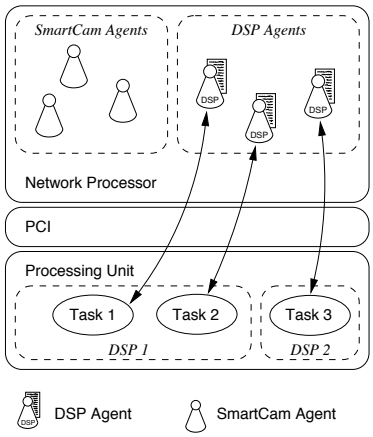

Figure 2. An agency hosting DSP agents and SmartCam agents.

On the processing unit, the *DSP framework* provides an environment for the video processing tasks and introduces a layer of abstraction for the DSP applications. Its main tasks 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 which includes data exchange between DSP applications as well as data exchange between DSP applications and applications executed on the network processor [10].

The *SmartCam framework* is executed on the network processor. It provides an abstracted view of the processing unit for applications on the network processor and allows them to load and unload executables to the processing unit.

3.3 Mobile Agent Framework

The mobile agent framework [11] is the highest level of abstraction in our smart cameras. Each video processing task is represented by an instance of a mobile agent. The agents act autonomously and carry out the required actions in order to fulfill their mission. Two different types of agents can be identified: (1) DSP agents, and (2) SmartCam agents. Figure 2 shows an agency hosting both types of agents.

DSP agents are used to represent video processing tasks. This type of agents 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 responsi-

ble 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. Exploiting mobility of agents allows to easily migrate video processing tasks from one smart camera to another. In contrast, 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 is the environment for the mobile agents. The agency further hosts a set of system agents which provide services for the DSP agents and SmartCam agents. The DSPLibAgent for example acts as an interface to the DSPs of the processing unit. Thus, DSP agents can interact with their image processing tasks in the same manner as with other agents. Further agents contain information about the location and configuration of the current smart camera as well as information about the 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 [12].

4 Multi-Camera Tracking

In this paper, we present an approach for autonomous and decentralized object tracking. The foundation of our approach is a well-known tracking algorithm which is able to track a single object within a scene. This tracking algorithm is encapsulated by an agent which contains the application logic for building a decentralized tracking environment.

The tracking algorithm is implemented as a dynamically loadable DSP executable because the tracker has to process the acquired images in real-time. Only abstract information about the tracked object such as the current position and the trajectory is reported to the agent. The agent in turn 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 Tracker Requirements

Our approach for autonomous, decentralized tracking of objects is basically independent of the concrete tracking algorithm used. However, the presented method introduces some requirements for the tracking algorithm which limit the number of possible tracking algorithms. Most of these requirements are a consequence of loading the tracking algorithm dynamically as needed. The main issues are:

- 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).
- Compact internal state of the tracker. When migrating the tracking agent from one camera to the next, the current internal state of the tracker must be stored and transferred as well. 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 the identification of the same object on the next camera. The object may appear differently due to the position and orientation of the camera.

Taking these requirements into account, the CamShift algorithm was chosen do demonstrate the feasibility of the presented tracking approach.

4.2 CamShift Algorithm

The Continuously Adaptive Mean-shift algorithm [3], or CamShift, algorithm is a generalization of the Mean-shift algorithm [13]. CamShift operates on a color probability distribution image produced from histogram back-projection. It is designed for dynamically changing distributions which occur when objects in video sequences are being tracked and the object moves so that the size and location of the probability distribution changes 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 is found by 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.

4.3 Handover Mechanism

The handover mechanism is the crucial part of the presented autonomous, decentralized tracking method. It extends a tracking algorithm designed for tracking an object in a single video stream to a multi-camera tracking solution. The handover procedure of a tracked object from one camera to the next one requires the following basic steps:

1. Select the "next" camera(s)
2. Migrate the tracking agent to the next camera(s)
3. Initialize the tracking task
4. Discover the object of interest
5. Continue tracking

Identifying the potential next cameras for the handover is done without a central point of coordination. Moreover, we exploit neighborhood relations within the smart camera network. Each camera has defined a set of migration regions which are described by a polygon in the 2D image space and a motion vector. Each migration region is assigned to one or more neighboring cameras. The 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 regions is managed locally by the SceneInformationAgent, a new system agent on each smart camera. When the tracked object enters a migration region

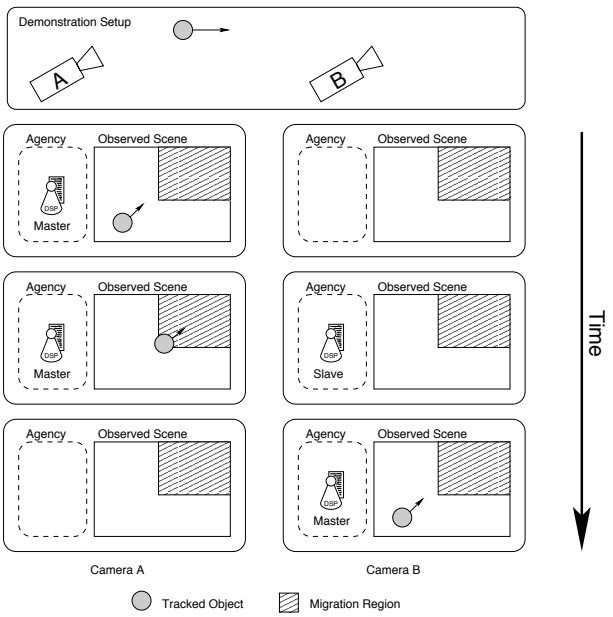

Figure 3. Master/Slave handover strategy.

and the trajectory matches the motion vector of the migration region, the tracking agent initiates the handover to the corresponding adjacent camera(s).

The next two steps of the handover process (migration and initialization) are implicitly managed by our mobile agent system. The internal state of the tracking algorithm is included as local data to the tracking agent. In the case of the CamShift tracking algorithm the internal state is the color distribution of the tracked object. The (migrated) tracking agent uses the local data for initialization on the new camera. Object discovery and continuing tracking is then executed by the migrated tracking algorithm.

Master/Slave Handover

The tracking agent may use different strategies for the handover [14]. The approach presented in this paper follows the master/slave paradigm. Figure 3 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 now becomes the master as it observes and tracks the target now. The new master then 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 discovered the target object, the master instructs all other slaves to terminate as well. The number of slaves created in a practical surveillance system is limited. Even in a very dense surveillance system, the number of created slaves will be in the range of three when taking the motion vectors into account. This makes our approach also applicable for large surveillance systems where a single object should be tracked.

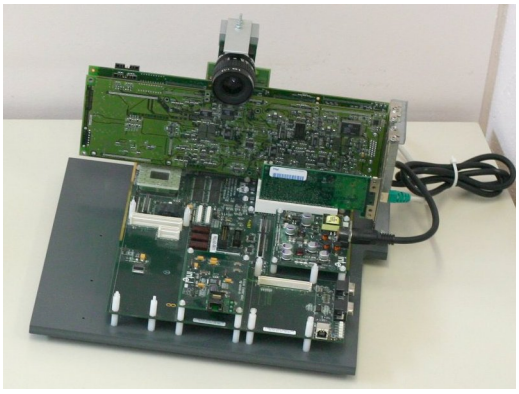

Figure 4. The SmartCam prototype.

The information required for initializing the tracking algorithm on the next camera heavily depends on the tracking algorithm itself. In the case of the CamShift algorithm, the description of the object to track and the initial search window are used for initialization. The description of the object is obtained from the algorithm itself and contains the color-histogram of the object. The initial search window is obtained by the agent from the SceneInformationAgent.

5 Experimental Results

5.1 Evaluation Setup

To show the feasibility of our approach, we have installed a test environment in our laboratory for tracking persons. The experimental setup consists of two smart camera prototypes as depicted in figure 4. Each camera consists of an *Intel IXDP425 Development Board* which is equipped with an *Intel IXP425* network processor. Two *Network Video Development Kits (NVDK)* from Ateme build the processing unit. Each board is comprised of a TMS320C6416 DSP from Texas Instruments running at 600 MHz with a total of 264 MB of on-board memory.

The smart cameras are operated by a standard GNU/Linux system for embedded systems. The agent system is implemented in Java and uses the Diet-Agents System (see `http://diet-agents.sf.net`) as foundation. For the java virtual machine JamVM version 1.3.0 (see `http://jamvm.sf.net`) with GNU classpath 0.14 has been chosen because it is rather small and thus suitable for use in an embedded system. However, JamVM does not feature a just-in-time compiler; it only interprets the Java bytecode which introduces noticeable performance penalties compared to virtual machines using just-in-time compilers.

The first part of the evaluation addresses the implementation of the CamShift tracking algorithm while the second

Code size: (dynamically loadable executable)	15 kB
Memory:	300 kB
Internal state:	256 Bytes
Initialize color-histogram:	$<$ 10 ms per frame
Identify tracked object:	$<$ 1 ms per frame

Table 1. Characteristics of the CamShift algorithm.

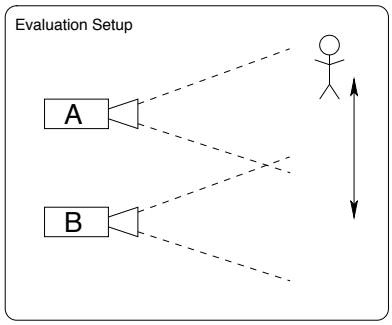

Figure 5. Camera setup for demonstrating the handover.

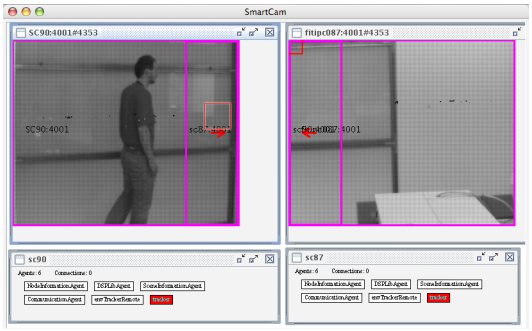

Figure 6. Visualizing the position of the tracked object.

Loading dynamic executable:	0.18 s
Initializing tracking algorithm: (5 frames @ 20 fps)	0.25 s
Creating slave on neighboring camera:	2.13 s
Reinitializing tracking algorithm on slave camera:	0.04 s
Total	2.57 s

Table 2. Evaluation of the handover time

part focuses on the integration of the tracking algorithm in the agent system as well as the handover procedure for multi-camera tracking.

5.2 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. The code-size of the dynamic DSP executable is about 15 kB. The internal state of the CamShift algorithm is the color histogram of the tracked object which requires 256 Bytes of memory.

When initializing the algorithm to track a concrete object, it 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. When tracking the object in a video stream, the tracking algorithm requires less than 1 ms for obtaining the new position of the object in an image.

5.3 Multi-Camera Tracking

The evaluation of the handover between two adjacent cameras is done by tracking a person in our laboratory. Figure 5 sketches the setup of our configuration. The fields of view of both cameras overlap, but this is due to spatial constraints and not a requirement of the tracker. Tracking a person is started on camera A by creating a tracking instance. The tracking algorithm on the processing unit first learns the description of the target within a given initialization region provided by the agent. When the person walks out of the current field of view, i.e., it enters the migration region, the tracking agent migrates to camera B and continues tracking on this camera.

Figure 6 shows a screenshot of the visualizer displaying the view of both cameras together with the agents on the camera during the handover. The current position of the person is indicated by the red square on the left camera (the background image is refreshed less frequently and thus not up to date). The master tracking agent is also on the left camera since the person is still within the cameras field of view. Because the person has already entered the migration region, a slave has already been created on the right camera waiting for the peson to appear.

Master/Slave Handover

The crucial part for multi-camera tracking is the handover from one camera to the next. Therefore, the four major time intervals during handover have been quantified. Table 2 enlists the obtained results.

Starting the tracking algorithm from a DSP agent requires 180 ms. 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 2.5 s 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 2.3 s) 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 ms which is negligible compared to the time required for creating the slave agent.

Multiple neighboring cameras

In the above evaluation, only two consecutive cameras have been used. To show the behaviour when a camera has more than one neighbor we have extended the setup by two additional PCs hosting an agent system. When the target enters the migration region, additional slaves are created on

Number of neighbors	*Time to create slaves*
1	2.52 s
2	3.03 s
3	3.51 s

Table 3. Handover with multiple neighboring cameras.

these PCs.

When a migration region directs to more than one neighboring camera, the time required to create the slaves is linearly dependent on the number of slaves (cf. table 3). Hence the slaves are created in parallel, the time required to create all slaves equals the largest time interval for creating a single slave. The linear factor is introduced by the limited performance of the agent system initiating the creation of the slaves.

6 Conclusion

In this paper we have presented our novel multi-camera tracking approach implemented on embedded smart cameras. Each object of interest has a corresponding tracking instance which is represented by a mobile agent. The agent follows the tracked object from camera to camera as the object moves within the surveilled area. The handover from one camera to the next one exploits the spatial relationship between cameras, expressed by the migration regions, is used for a local handover between neighboring cameras. This results in a fully distributed handover process with no central coordination which in turn is important for high autonomy and scalability.

Our multi-camera tracking is implemented using a mobile agent system. On the one hand, mobile agents ease the development of distributed applications. On the other hand, the java-based agent system requires substantial computing performance on our embedded platform and limits the handover times.

Future work includes (1) replacing the java-based agent system by a more efficient (middleware) system providing services for data and code migration, (2) deploying our tracking approach on larger networks of cameras, and (3) improving the visualization of the tracked object.

7 References

[1] W. Wolf, B. Ozer, and T. Lv, "Smart cameras as embedded systems," *Computer*, vol. 35, no. 9, pp. 48–53, Sept. 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] G. R. Bradski, "Computer vision face tracking for use in a perceptual user interface," *Intel Technology Journal*, no. Q2, p. 15, 1998.

[4] S. Fleck and W. Straßer, "Adaptive Probabilistic Tracking Embedded in a Smart Camera," in *IEEE Embedded Computer Vision Workshop (ECVW) in conjunction with IEEE CVPR 2005*, 2005, pp. 134 – 134.

[5] 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 *Computer Vision and Pattern Recognition Workshop, 2006 Conference on*, Jun. 2006, pp. 118 – 118.

[6] 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 *Multimedia and Expo, 2006. ICME 2006. IEEE International Conference on*, 2006.

[7] A. Rowe, C. Rosenberg, and I. Nourbakhsh, "A second generation low cost embedded color vision system," in *IEEE Embedded Computer Vision Workshop (ECVW) in conjunction with IEEE CVPR 2005*, vol. 3, 2005, pp. 136 – 136.

[8] P. Remagnino, J. Orwell, D. Greenhill, G. A. Jones, and L. Marchesotti, *Multimedia Video Based Surveillance Systems: Requirements, Issues and Solutions*. Kluwer Academic Publishers, 2001, ch. An Agent Society for Scene Interpretation, pp. 108 – 117.

[9] B. Abreu, L. Botelho, A. Cavallaro, D. Douxchamps, T. Ebrahimi, P. Figueiredo, B. Macq, B. Mory, L. unes, J. Orri, M. J. Trigueiros, and A. Violante, "Video-based multi-agent traffic surveillance system," in *Intelligent Vehicles Symposium, 2000. IV 2000. Proceedings of the IEEE*, 2000, pp. 457 – 462.

[10] A. Doblander, B. Rinner, N. Trenkwalder, and A. Zoufal, "A Middleware Framework for Dynamic Reconfiguration and Component Composition in Embedded Smart Cameras," in *WSEAS Transactions on Computers*, vol. 5, no. 3. WSEAS Press, March 2006, pp. 574 – 581.

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

[12] M. Bramberger, B. Rinner, and H. Schwabach, "A Method for Dynamic Allocation of Tasks in Clusters of Embedded Smart Cameras," in *Proceedings of the IEEE International Conferens on Systems, Man and Cybernetics*, October 2005, pp. 2595 – 2600.

[13] D. Comaniciu, V. Ramesh, and P. Meer, "Real-time tracking of non-rigid objects using mean shift," pp. 142–151.

[14] 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 the IEEE International Conference on Advanced Video and Signal Based Surveillance*, September 2005.