Integrating Multi-Camera Tracking into a Dynamic Task Allocation System for Smart Cameras

M. Bramberger, M. Quaritsch, T. Winkler, B. Rinner
Institute for Technical Informatics
Graz University of Technology, AUSTRIA
{bramberger, quaritsch, winkler, rinner}@iti.tugraz.at

H. Schwabach
Video & Safety Technology
ARC seibersdorf research, AUSTRIA
helmut.schwabach@arcs.ac.at

Abstract

This paper reports on the integration of multi-camera tracking into an agent-based framework, which features autonomous task allocation for smart cameras targeting traf- c surveillance. Since our target platforms are distributed embedded systems with limited resources, the trackers may only be active, if the target is in the camera's eld of view. Consequently, the tracking algorithm has to migrate from camera to camera in order to follow the target, whereas the decision when and whereto the migration takes place is reached autonomously by the tracker. Consequently, no central control host is required. We further present different strategies on when to migrate a tracker, and how to determine the camera which will observe the tracked object subsequently.

We have realized the tracker's control by using heterogeneous mobile agents, which employ a state-of-the-art tracking algorithm for tracking. The tracking system has been implemented on our smart cameras (SmartCam) which are comprised of a network processor and several digital signal processors (DSPs) and provide a complex software framework.

Keywords: *smart cameras; mobile agents; traf c surveillance; embedded systems; single-object tracking*

1 Introduction

Surveillance systems currently undergo a dramatic shift. Traditional surveillance systems of the first and second generation have employed analog CCTV cameras, which transferred the analog video data to digital base stations where the video analysis, storage and retrieval takes place. Current semiconductor technology enables surveillance systems to leap forward to third generation systems which employ digital cameras with on-board video compression and communication capabilities. *Smart cameras* [9] [2] even go one step further; they not only capture and compress the grabbed video stream, but also perform sophisticated, real-time, onboard video analysis of the captured scene. Smart cameras help in (i) reducing the communication bandwidth between camera and base station, (ii) decentralizing the overall surveillance system and hence to improve the fault tolerance, as well as (iii) realizing more surveillance tasks than with traditional cameras.

Typical tasks to be run in a surveillance system targeting traffic surveillance include MPEG-4 video compression, various video analysis algorithms such as accident detection, wrong-way drivers detection and stationary vehicle detection. Additionally, the tracking of objects among many cameras is an important issue. However, computing power is limited, hence not all tasks can be allocated to a smart camera concurrently. Therefore, we have developed a resource-aware task allocation system [3], which distributes the surveillance tasks autonomously within groups of smart cameras. Based on mobile agents this task-allocation system works in a distributed manner without a central host.

This paper reports on the integration of multi-camera tracking into our task-allocation system. Since our target platforms are distributed embedded systems with limited resources, the tracking-algorithms may only be active, if the tracked object is in the camera's field of view. Additionally, since the surveillance system is fully distributed, the tracker has to migrate from camera to camera in order to follow the target.

We have realized the tracking control strategy using heterogeneous mobile agents, which comprise a well know tracking-algorithm. Consequently, these mobile agents are integrated into the task-allocation system, which provides the required infrastructure (e.g., resource monitoring, communication infrastructure).

The dynamic task allocation, and, consequently the tracking agents, have been implemented on our smart cameras (*SmartCam* [2]) which are comprised of a network processor and several digital signal processors (DSPs) and provide a complex software framework.

0-7803-9385-6/05/$20.00 ©2005 IEEE

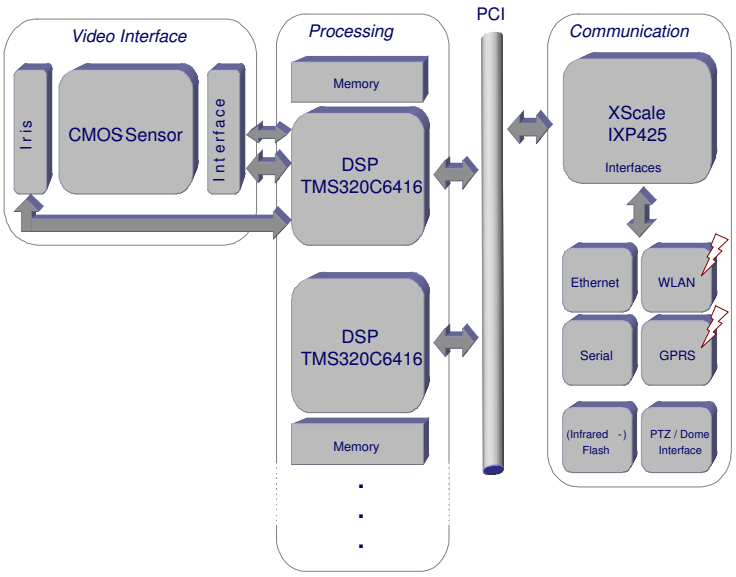

Figure 1. The hardware architecture of the smart camera

The remainder of this paper is organized as follows: Section 1.1 sketches related work. Section 2 briefly presents hardware and software of our SmartCam. Section 3 introduces the dynamic task allocation framework and explains the architecture of the surveillance system. The integration of multi-camera tracking is explained in Section 4. Section 5 and 6 present the implementation and experimental results. Finally, Section 7 concludes the paper with a summary and an outlook on future work.

1.1 Related Work

In [1], Abreu et. al present Monitorix, a video-based multi-agent traffic surveillance system, based on PCs. The presented approach, however, does not work in real-time. Ellis presents in [4] a multi-view video surveillance system, which makes use of intelligent cameras. The goal of the surveillance system is the monitoring of public sites, therefore, the system architecture is setup statically. Remagnino at. al describe in [7] the usage of agents in visual surveillance systems. An agent-based framework is used to accomplish scene understanding. The system enables handover of events between neighboring cameras.

2 The Smart Camera

Smart cameras are the core components of a 3^{rd} generation surveillance system. These cameras perform video sensing, high-level video analysis and compression and transfer the compressed data as well as the results of the video analysis to a central monitoring station. The video analysis tasks implemented in the cameras clearly depend on the overall surveillance application and may include accident detection, vehicle tracking and the computation of traffic statistics. Most of these tasks, however, require a very high computing performance on the cameras.

2.1 Hardware Architecture

Our smart camera has been designed as a low-power, high-performance embedded system. As depicted in figure 1, the smart camera consists of three main units. (1) The sensing unit, (2) the processing unit, and (3) the communication unit.

A high-dynamic, monochrome CMOS image sensor is the heart of the sensing unit. It delivers images with 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 is equipped with up to four digital signal processors (DSPs) TMS320C6415 from Texas Instruments. The DSPs deliver an aggregate computing performance of almost 20 GIPS while keeping the power consumption low. The DSPs are coupled via a local PCI bus which serves also as connection to the network processor (Intel XScale IXP425) in the communication unit. The communication unit provides access to the camera's environment.

The communication of the smart camera is basically twofold. First, the communication unit manages the internal communication between either the DSPs and the DSPs and the network processor. Second, it manages the external communication, which is usually IP-based. The XScale

processor is operated by Linux due to large number of available tools and applications available under this operating system. [2] provides a more detailed insight into the hard- and software architecture of the smart camera.

2.2 Software Architecture

The software architecture of our smart camera is designed for flexibility and reconfigurability. The software architecture consists of several layers which can be grouped into: (1) The DSP framework, which is implemented on the DSPs, and (2) the SmartCam framework, running on the network processor.

DSP Framework The main purpose of the DSP framework is (i) the abstraction of hardware and communication channels, (ii) the support for dynamic loading and unloading of applications, and (iii) the management of on-chip and off-chip resources of the DSP system by utilizing Texas Instruments Reference Framework 5 [5].

SmartCam Framework The SmartCam framework serves the following purpose: First, it provides abstraction of the DSPs to ensure platform independence of the agent-system and application layers. Second, the application layer uses the provided communication methods (messaging to the DSPs and IP-based communication to outer world) to exchange information, and work as a relay service. Finally, the agent-system layer is run on top of Java, whereas the agents are run as a part of the agent platform.

3 Task Allocation Framework

3.1 Surveillance System Architecture

The architecture of the surveillance system consists of a large number of smart cameras deployed alongside highways or in tunnels. Since these smart cameras have limited computational resources, it is not possible to run all required surveillance tasks on a smart camera. Therefore, physically co-located smart cameras are combined into logical groups, so called *surveillance clusters*. Consequently, sets of surveillance tasks (e.g. accident detection, vehicle counting, vehicle classification) are then allocated to surveillance clusters. This is feasible, since events, observed by co-located cameras are causally associated with each other. Therefore, it is not important on which smart camera a surveillance task is allocated, as long as these surveillance clusters do not span a too large area. However, not all surveillance tasks require small surveillance clusters; classifying and counting of vehicles, for example, may be spread over a larger area, while accident or fire detection tasks are usually allocated to smaller clusters. Therefore, a smart camera may be a member of several surveillance clusters (cp. figure 2).

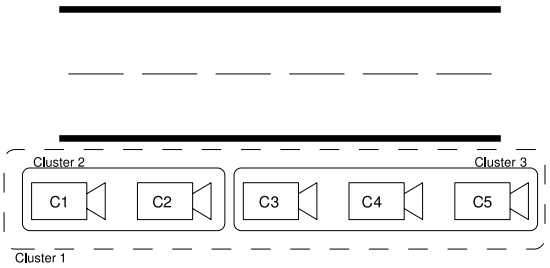

Figure 2. A tunnel surveillance scenario with three surveillance clusters

In contrast to the presented surveillance tasks, which may be allocated to any smart camera in the surveillance cluster, there are also several surveillance tasks which are required to run on a specific smart camera. These tasks possess an affinity to a scene or a camera, respectively. Tracking algorithms, for example, require to be run on a specific camera, which observes the tracked object. These scene-affine tasks are also contained in a surveillance cluster, however, these tasks are very likely allocated to the required smart cameras.

3.2 Autonomous Task Allocation

In order to automatically distribute tasks within surveillance clusters, our task allocation system is used [3]. The goal of the task allocation system is to autonomously find a mapping of tasks to smart cameras which satisfies all requirements and is optimal with respect to a specified metric, i.e. some cost function. This system is designed to operate fully distributed, hence no central host is required for operation. Since the surveillance tasks have firm real-time requirements, the task allocation system has to take care that no deadlines are missed due to an overload of a camera or a camera's subsystem, respectively.

The re-allocation of tasks may be necessary due to events, raised by hardware or software: (1) Hardware events usually originate from changed resource availability due to added or removed cameras, hardware breakdown, or re-usability of recovered resources. (2) Software events are caused by changes to resource requirements due to changes in the task set of the surveillance cluster, or because of changes in the quality-of-service level (QoS) of tasks, i.e.due to detected events in the observed scene.

The allocation of tasks to smart cameras is done in two steps. (1) In the first step, all feasible allocations of tasks to smart cameras (allocations where no real-time requirements are violated) are determined. (2) In the second step, the optimal allocation of tasks is chosen by using a cost function.

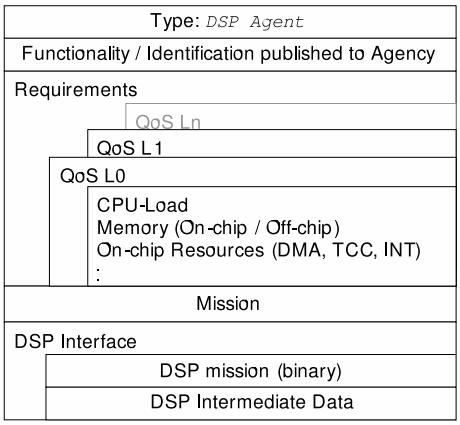

Figure 3. A DSP-agent

3.3 Mobile-Agent based Framework

The task allocation framework uses mobile agents to encapsulate the surveillance tasks. Mobile agents are the best choice, since this paradigm supports autonomous actions, mobility, and platform independence. In accordance with the hard- and software architecture (cp. section 2) the agents (DSP-agents, cp. figure 3), encapsulating the surveillance tasks, are divided into a control-part which can be seen as the agent itself, and a surveillance-part, which is downloaded to the DSP as the computational-intensive part of the surveillance task (e.g., tracking, stationary-vehicle detection, etc.). Figure 4 depicts a smart camera's agency including DSP-agents (Worker Agents — WA) and their connection to the DSP using the DSP interaction agent (DIA). The system-information agent (SA), and the cluster-information agent (CIA) determine information on the current system status (SA), and the status of the other cameras in the surveillance cluster (CIA).

As depicted in figure 3, DSP-agents can have many QoS-levels, each of which stores its resource requirements. These resource requirements are used by our task allocation system to distribute the DSP-agents among the smart camera in the surveillance cluster. Scene-affine tasks, like trackers or video-encoders, additionally contain the scene or camera, to which they have an affinity. This information is used by our task-allocation system to force the agents allocation to the required camera.

4 Multi-Camera Tracking

In our system, trackers have to fulfill two basic requirements: (1) Since the tracker is not running continuously on the cameras, the tracker or the tracking-agent, respectively, has to ensure, that it is running on the appropriate camera in order to watch and track the desired object. (2) The smart

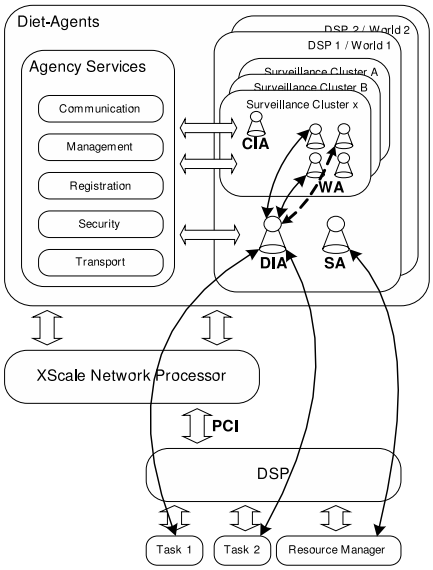

Figure 4. A DSP-agency

cameras have limited resources, which are shared between several surveillance tasks executed on the smart cameras. Therefore the tracking-agent has to use as little resources as possible. Consequently, the execution of many trackers, which are tracking the same target on different cameras concurrently should be avoided.

However, in order to continue tracking on subsequent cameras, the tracking-agents include tracking information like features or patterns to enable and to ease the next camera to find the target.

For the integration of a tracking-agent three issues have to be considered: (1) Choosing the appropriate tracking algorithm, (2) the determination of the next camera observing the tracked object, and (3) the selection of the most suitable migration strategy,

4.1 Tracking Algorithm

The focus of this paper is the integration of tracking algorithms into the task allocation framework. Consequently, we are using well-known tracking-algorithms which are state-of-the-art. For the integration of a tracking-algorithm, however, we are differentiating trackers whether a background model is required or not.

Background model required Tracking algorithms, which require a background model have long initialization times, since the background model requires several frames to be initialized properly. Consequently, these trackers have to be started well before they have to provide usable results.

No background model required In contrast, tracking algorithms (e.g., KLT [8]) which do not require a background model, can be migrated fast, since these tracking algorithms are able to provide results by the first computed frame.

4.2 Determining the Appropriate Camera

The determination of the next camera observing the target, is done using statically defined regions ("migration regions"). Therefore, every camera stores its set of migration regions, which includes the geometric region, the required motion vector, and the next camera in this direction. Figure 5 depicts a simple surveillance scene, indicating only a single migration region for simplicity. It is possible that migration regions are overlapping due to crossings and perspective overlaps. Therefore, a motion vector is computed for each target, which describes the direction of the target's motion. Consequently, each migration region also has assigned a motion vector, which is compared with the motion vector of the target. If the two motion vectors are almost identical, the migration region is used.

Crossings, which can not be clearly seen, have to be handled slightly different. In order to further track the object, two or more cameras have to chosen as the next camera. Therefore, each of these cameras has to search the target in its field-of-view until a camera identifies the tracked object.

4.3 Migration Strategy

The strategy, when and how a tracking-agent should migrate to the next camera is vital for finding the target on the next camera.

Plain Tracker The plain tracker migrates to the next camera as soon as the target enters the migration region and the next camera can be determined. This is the straight-forward strategy with the drawbacks, that (1) the tracker leaves the camera before the target is in the field of view of next camera, which means that the tracker is not aware of the position of the target until it enters the field of view of the next camera. Additionally, (2) a high network utilization can slow down the migration of the tracking-agent. Consequently, the tracking-agent may miss the target on the camera due to the late arrival.

Master/Slave Tracker The master/slave tracker overcomes the drawbacks of the plain tracker. In order to track an object, the tracker is executed on the smart camera, which observes the target (master tracker). As soon as the target enters the migration region, the tracker is cloned, and the clone, called slave-tracker, is migrated to the next camera, where the slave-tracker waits for the target. The master-tracker is active until the slave-tracker identifies the target.

Figure 5. A tracking scene with migration area and motion vector

At this point,the master-tracker is destroyed, and the slave-tracker becomes the master-tracker.

5 Implementation

For the implementation of our task allocation system, and the tracking system, we have chosen to use the Diet-Agents System (see http://diet-agents.sf.net) which runs on top of Java, since it provides all required features like mobility and autonomy, and it is reasonable small and fast, which is inevitable for embedded systems.

We have chosen to implement and integrate the master/slave-tracker due its mentioned advantages in comparison with the plain-tracker. Furthermore, a KLT-tracker is used for tracking due to its short initialization time. The porting of the KLT-tracker to the DSP is, however, not yet finished. Therefore a tracking-simulation agent simulates the behavior of the KLT-tracker in a realistic way by reporting predefined trajectories and features of the tracked object to the tracking agent. The tracking-agent communicates with the tracking-simulation agent using the same interface as it is defined for the KLT-tracker. Consequently, the behavior of the tracking system will not change as soon as we start using the KLT-tracker.

5.1 Hardware Setup

To verify, test and evaluate the presented tracking system, we have used two hardware platforms. Two prototypes of our smart camera and PCs equipped with DSP boards.

The prototypes of our smart camera consist of *Intel IXDP425 Development Boards*, which are equipped with an *Intel IXP425* network processor running at 533 MHz. The boards are operated with Linux Kernel 2.6.8.1, which allows the usage of standard software packages, and enables interoperability with PC-based Linux systems. The

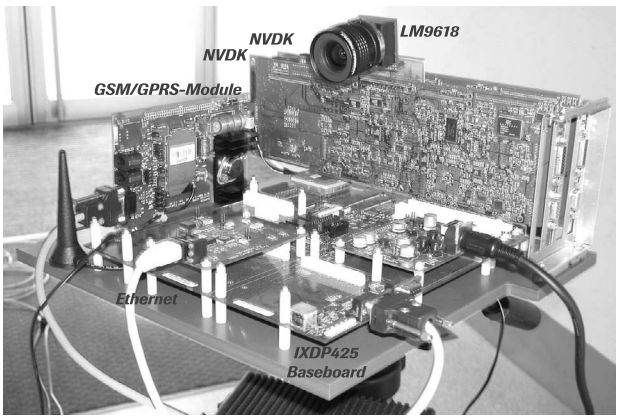

Figure 6. The prototype of the smart camera

boards support up to four DSP boards, however, our prototypes are equipped with two *Network Video Development Kits (NVDK)* from ATEME. 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. Image acquisition is done using the *National Semiconductor LM9618* monochrome CMOS image sensor, which is connected to one of the DSP boards. Due to the lack of additional smart camera prototypes, we are using additional PCs, which are equipped with an NVDK board.

6 Experiments

We have conducted several experiments in order to test and evaluate the tracking system. Therefore, we have implemented three different tracking simulations:

1. Simple one-way traffic tunnel scenario to test the basic functionality of migrations and finding the target on the next camera.
2. Simple two-way traffic scenario in order to additionally test the effectiveness of the motion vectors, since the migration regions are overlapping.
3. A complex highway scenario including crossings, which are only contained partially in the field of view. Hence, agents have to be cloned and migrated to all possible smart cameras in order to find the tracked object.

The tracking system worked well and delivered the correct results with all three tracking simulators. The migration of tracking agents between smart cameras required approximately 900 ms due to the low performance of the used Java-VM. Since the master/slave tracker creates its slaves as soon as the target enters the migration region, the migration regions can be enlarged, if the migration times are too high.

7 Conclusion

In this paper we have reported on the integration of multi-camera tracking into our task allocation system for smart cameras. In order to reduce the resource usage, the tracking algorithm is only run on the appropriate smart camera as long as the tracked object is in its field of view. Therefore, the tracking algorithm is encapsulated into a mobile agent, which decides autonomously, based on the results from the tracking algorithm, when to migrate to the next smart camera. In order to test and evaluate the mobile-agent based tracking, we have implemented the proposed strategies on the prototype of our smart camera.

Future work includes (1) the test and evaluation using a KLT-tracker, (2) more extensive tests under real-world conditions, and (3) the implementation of other tracking algorithms.

References

[1] B. Abreu, L. Botelho, A. Cavallaro, D. Douxchamps, T. Ebrahimi, P. Figueiredo, B. Macq, B. Mory, L. Nunes, J. Orri, M. J. Trigueiros, and A. Violante. Video-Based Multi-Agent Traffic Surveillance System. In *Proceedings of the IEEE Intelligent Vehicles Symposium*. IEEE, Oct 2000.

[2] M. Bramberger, B. Rinner, and H. Schwabach. An Embedded Smart Camera on a Scalable Heterogeneous Multi-DSP System. In *Proceedings of the European DSP Education and Research Symposium (EDERS 2004)*, Nov 2004.

[3] M. Bramberger, B. Rinner, and H. Schwabach. Resource-aware dynamic task-allocation in clusters of embedded smart cameras by mobile agents. In *Proceedings of the IEE Intenational Workshop on Intelligent Environments*. IEE, June 2005.

[4] T. Ellis. Multi-camera video surveillance. In *Proceedings of the International Carnahan Conference on Security Technology*, pages 228–233. IEEE, 2002.

[5] T. Mullanix, D. Magdic, V. Wan, B. Lee, B. Cruickshank, A. Campbell, and Y. DeGraw. Reference Frameworks for eXpressDSP Software: RF5, An Extensive, High-Density System. Technical Report SPRA795A, Texas Instruments, April 2003.

[6] C. Regazzoni, V. Ramesh, and G. Foresti. Introduction of the special issue. *Proceedings of the IEEE*, 89(10), Oct 2001.

[7] P. Remagnino, J. Orwell, D. Greenhill, G. Jones, and L. Marchesotti. An agent society for scene interpretation. In G. Foresti, P.Mhnen, and C.S.Regazzoni, editors, *Multimedia Video Based Surveillance Systems*, pages 108–117. Kluwer Academic Publishers, 2001.

[8] C. Tomasi and T. Kanade. Detection and tracking of point features. Technical Report CMU-CS-91-132, Carnegie Mellon University, 1991.

[9] W. Wolf, B. Ozer, and T. Lv. Smart Cameras as Embedded Systems. *IEEE Computer*, 35(9):48–53, Sep 2002.