Smart Cameras for Embedded Vision

Milan Jovanovic, Andreas Klausner, Markus Quaritsch, Bernhard Rinner, Allan Tengg
Institute for Technical Informatics, Graz University of Technology

Introduction

Image processing has been a very active research area over the last decades, and results of this research can now be found almost everywhere. The market for image processing technology is rapidly increasing; its applications range from machine inspection over security to consumer products.

Two trends emerge in recent image processing research: distributed computing and embedded processing. These trends are very well exemplified in smart cameras [1, 2] which combine image sensing, image processing and communication on a single embedded device. Smart cameras not only capture and compress the grabbed video stream, but also perform sophisticated, real-time, on-board video analysis of the captured scene. In this article we present a smart camera prototype and discuss its potential for (distributed) embedded vision applications. These cameras help to migrate the video processing from central base stations to the image sensors.

Performing vision applications on embedded computing platforms poses several challenges. First, there are stronger resource limitations on embedded platforms than on general-purpose computers. This is especially true for computing and memory capacities. Second, the software development process is more tedious due to limited availability of dedicated image processing libraries and software frameworks. Finally, software coding requires more experienced programmers due to the stronger dependencies on the underlying hardware. However, we strongly believe that embedded computing platforms are crucial for many future vision applications. They help to increase the reliability, to reduce the power consumption significantly and to simplify the deployment and maintenance of such systems.

In distributed video applications such as surveillance, smart cameras introduce several benefits. Communication bandwidth between camera and base station can be reduced because images are processed directly on the sensor. Systems comprised of smart cameras are more flexible and more scalable compared to centralized systems. This allows to deploy more surveillance tasks than with traditional cameras. Decentralizing the overall surveillance system hence improves the availability as well [2].

Related Work

There exist various implementations of embedded smart cameras ranging from high-performance cameras to tiny single chip solutions. Heyrman et al. describe in [3] the architecture of a smart camera which integrates a CMOS sensor, processor and a reconfigurable unit on a single chip. This camera is designed for high-speed image processing using dedicated parts such as sensors with massively parallel outputs and region-of-interest readout circuits.

Integrating video processing into embedded systems is also an active research topic. Fleck and Straßer present in [5] a particle based filter algorithm for tracking objects. They use a commercially available camera which is comprised of a CCD image sensor, a Xilinx FPGA for image processing and a Motorola PowerPC CPU. Rowe et al.[6] promote a low-cost embedded vision system where image processing is integrated into a small camera. Due to the very limited memory and computing resources, only low-level image processing such as threshold filtering is possible. Kleihorst et al. describe in [7] a wireless smart camera mote comprised of a single-instruction-multiple-data (SIMD) video-analysis processor and an 8051 microcontroller. Wireless communication is based on the IEEE802.15.4 protocol. Another wireless smart camera has been developed by Kim et al [8].

SmartCam Architecture

Hardware Architecture

Our smart camera (SmartCam) [1] has been designed as a low-power, high-performance embedded system. Figure 1 depicts the camera's architecture which is comprised of a sensing unit, a processing unit and a communication unit. A CMOS sensor is the heart of the sensing unit. It delivers color images up to VGA resolution at 25 frames per second to the processing unit via a FIFO memory. The processing unit is composed of a variable number of digital signal processors (DSPs) which are connected via a local PCI bus. This unit is intended to execute image processing on the acquired images in real-time. An ARM-based network processor controls the

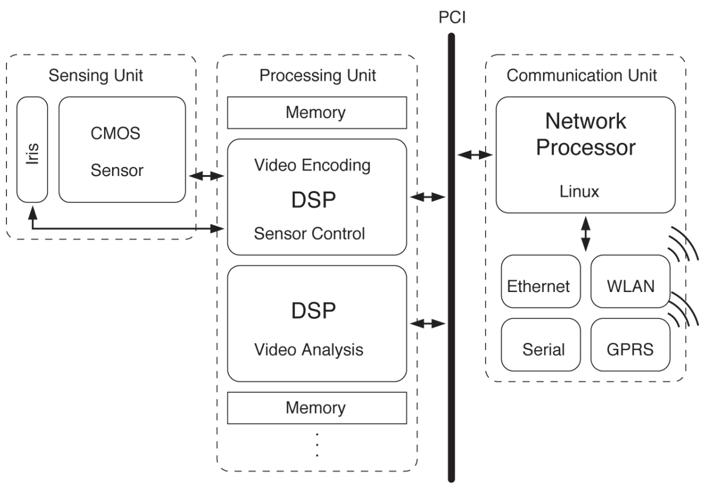

Fig.1: *SmartCam architecture*

communication unit which has two main tasks. First, it coordinates the internal communication among the DSPs as well as the DSPs and the network processor. Second, it provides communication channels to the outside world.

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.

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. The DSP framework supports dynamic loading and unloading of DSP applications and manages the available resources on each processor.

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 applications to communicate with the algorithms on the DSPs.

Mobile Agent Framework

Agents are the highest level of abstraction in our smart cameras. Each video processing task is represented by an instance of a mobile agent. A mobile agent framework [9] is executed on the network processor whereas each camera hosts an agency, the environment for mobile agents. The agents act autonomously and carry out the required actions in order to fulfill their mission. There are two different types of agents, namely DSP agents and SmartCam agents. DSP agents are used to represent video processing tasks and thus have a tight relation to the processing unit. The agent contains the DSP executable and is responsible for starting, initializing and stopping the DSP algorithm as required. The agent also knows how to interact with the DSP algorithm in order to obtain the information required for further actions. In contrast, SmartCam agents do not interact with the DSPs. Usually they perform control and management tasks. Exploiting mobility of agents allows to easily migrate video processing tasks from one smart camera to another and also encourages dynamic reconfiguration of the whole surveillance system.

Smart Camera Networks

In our research on *distributed smart cameras (DSC)*, we have implemented two case studies. The first study is on multi-camera object tracking focusing on the implementation of an autonomous tracker which follows an object of interest. The second study is on dynamic reconfiguration. We present our policy-based middleware used to reconfigure a network of smart cameras according to the current context as well as given objectives.

Multi-Camera Tracking

One application of our smart cameras is tracking the position of an object within a monitored area by exploiting collaboration of multiple cameras. The basic idea of our multi-camera tracking approach is to have a single tracking instance associated to an object of interest. This tracking instance follows the object from one camera to the next as the object moves within the monitored area. Hence, only the camera which currently observes the object of interest has to execute the tracking task while all other cameras can use their resources otherwise. Our multi-camera tracking solution is not intended as an exclusive task which is executed on each camera all the time but a tracking instance is started for a certain object of interest as an additional task.

A tracking instance is comprised of a mobile agent and a tracking algorithm. The tracking algorithm is responsible for extracting the position of an object from the video stream acquired by the camera. Because this is a computing intensive task, the tracking algorithm is executed on the processing unit. Only abstract information about the tracked object such as the current position and the trajectory is reported to the agent.

The mobile agent on the other hand contains the application logic and is responsible to follow the target from one camera to the next. The agent, therefore, uses the information provided by the tracking algorithm in order 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 continue tracking the object on the neighboring cameras.

Tracking Algorithm

The tracking algorithm is the foundation of our approach as it is responsible for detecting and extracting the position of the tracked object from the video stream. Our approach is basically independent of the concrete tracking algorithm used. The tracking algorithm, therefore, can be chosen to fit the individual needs, e.g., the type of object to track or the environmental conditions. However, the tracking algorithm has to cope with our highly dynamic tracking approach. Particularly, the algorithm must not suffer from long initialization times because it has to continue tracking as fast as possible after migrating to the next camera. The tracking algorithm must also be able to identify the same object on the next camera whereas the object may appear differently due to the position and orientation of the camera.

Taking these requirements into account, the color-based Continuously Adaptive Mean-shift algorithm (CamShift) [10] was chosen to demonstrate the feasibility of our approach. The CamShift algorithm is a color-based tracking algorithm which requires almost no initialization time after migration from one camera to another. When starting tracking an object, the algorithm extracts the color-distribution of the object of interest. Tracking the object in the video stream is done by finding an object with the same color-distribution within the search window. Output of the algorithm is the size and position of the object. CamShift is designed for dynamically changing distributions which occur when objects being tracked move so that the size and location of the probability color-distribution changes over time.

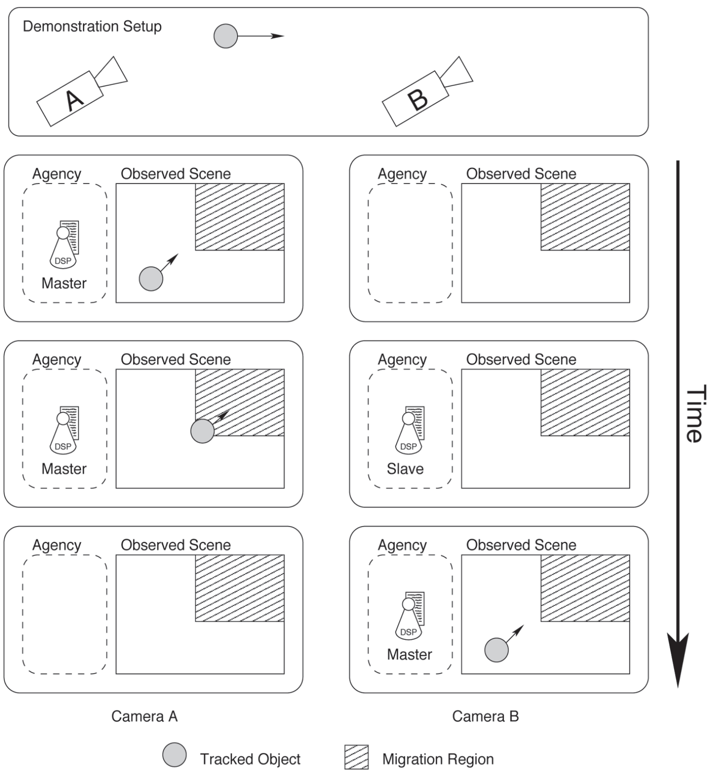

Fig.2: *Master/slave handover strategy*

Handover Mechanism

The tracking agent is responsible for following the target from one camera to the next. Thus, the handover mechanism is crucial for the presented autonomous, decentralized tracking method. Using our agent-based tracking approach, an algorithm designed for tracking an object in a single video stream can be extended for multi-camera tracking. 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 instance to the next camera
3. Initialize the tracking algorithm
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. Instead, 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 vector helps to distinguish among several smart cameras assigned to the same migration region.

The migration region and their assigned cameras represent the spatial relationship among the cameras. All information about the migration regions is managed locally on each camera by a dedicated agent. When the tracked object enters a migration region 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. Object discovery and continuing tracking is then executed by the migrated tracking algorithm.

The tracking agent may use different strategies for the handover [11]. For demonstrating our approach, we decided to implement the master/slave strategy. Figure 2 illustrates the handover procedure along with the instances of tracking agents for a sample scenario of two consecutive cameras. During the handover, there exist two tracking instances 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 agent also queries the 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 obtained 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 instances change. The slave now becomes the master as it observes and tracks the object now. The new master also notifies the old master that the target is now in its field of view, whereupon the old master terminates itself.

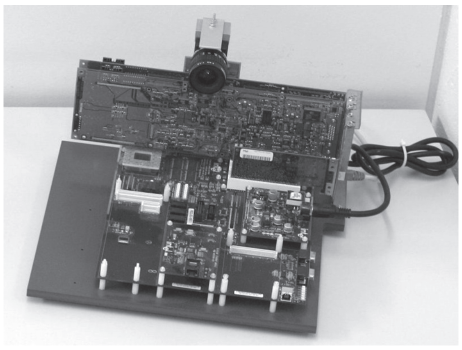
Fig.3: *SmartCam prototype*

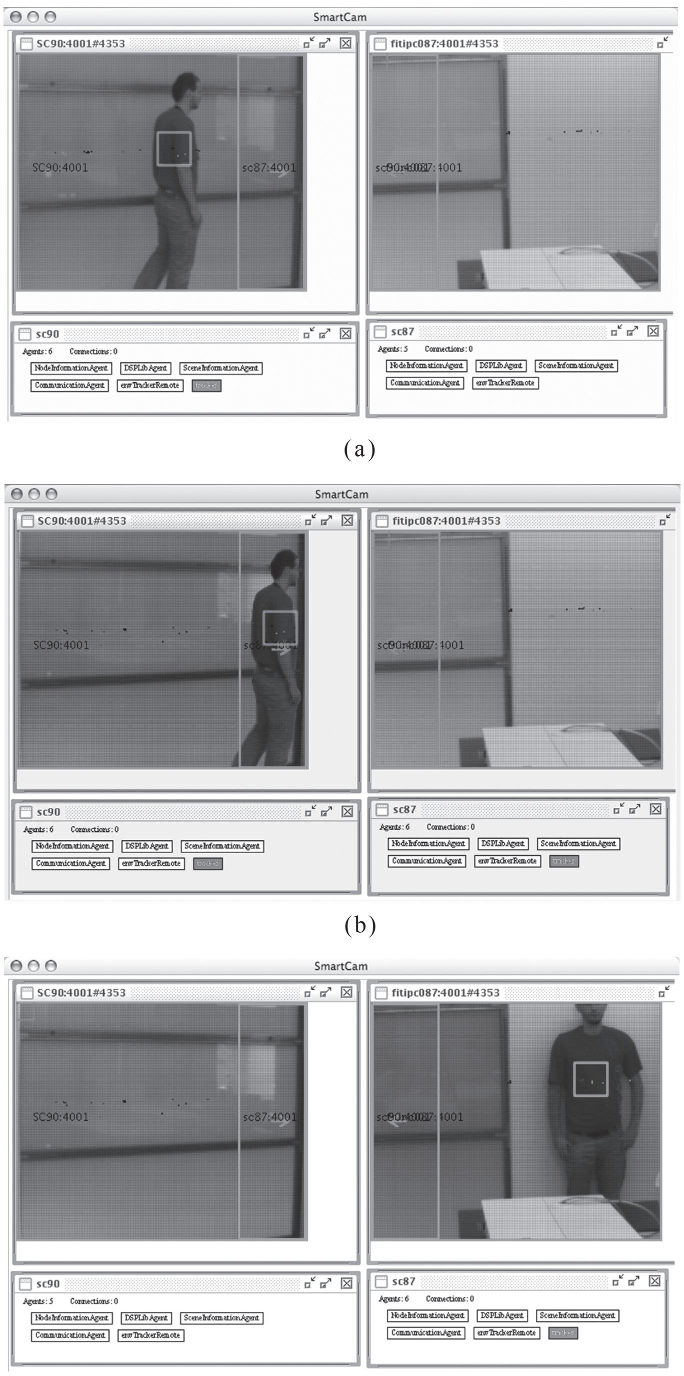

Fig.4: *Visualizer. The upper part of each screenshots shows the view of both cameras while the lower part illustrates the agency on each camera along with the current agents. The center of the tracked person is highlighted by the red square.*

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. This decentralized handover mechanism is also applicable for large scale surveillance systems because only neighboring cameras in a certain direction are involved.

Multi-Camera Tracking Demonstration

To demonstrate the feasibility of our multi-camera tracking, we have implemented the presented approach and tested on tracking persons in our laboratory. Therefore, we used two prototypes (cf. Figure 3) of our smart camera and tracked a

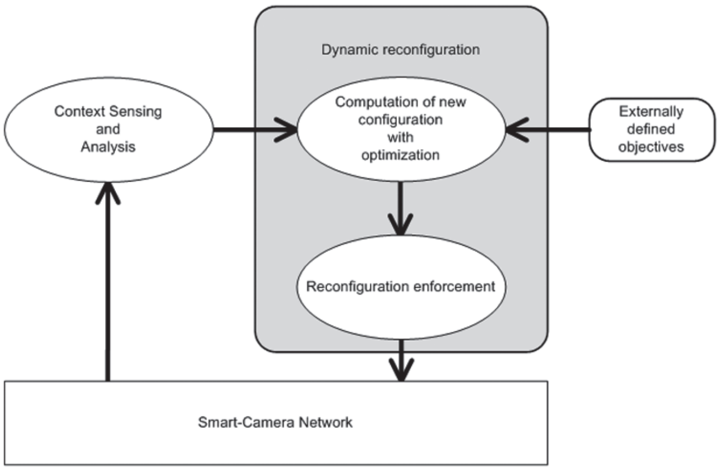
Dynamic reconfiguration
Context Sensing and Analysis
Computation of new configuration with optimization
Externally defined objectives
Reconfiguration enforcement
Smart-Camera Network

***Fig.5:** Dynamic reconfiguration loop*

person walking from the viewshed of one camera to the viewshed of the neighboring one. Figure 4 illustrates the handover procedure by screenshots of the visualizer. Tracking a person is started on the left camera by creating a tracking instance. The tracking algorithm first learns the description of the target and then tracks the position of the person (cf. Figure 4(a) – the tracking agent resides on the left camera, indicated by the highlighted rectangle in the lower part of the image). When the person walks out of the current field of view, i.e., it enters the migration region, the tracking agent creates a slave on the right camera, and the slave is waiting for the person to appear (cf. Figure 4(b) – the tracking agent is present on both cameras). When the person is discovered by the tracking instance on the right camera, the agent on left camera terminates itself (cf. Figure 4(c)).

Dynamic Reconfiguration

Dynamic reconfiguration refers to the modification of software tasks executed on nodes of the camera network. In order to support dynamic reconfiguration in our DSC system, we have extended the SmartCam-framework by an adaptive and reflective middleware system (ARMS) on top of the mobile agent system. Our ARMS provides the domain-specific services (1) task allocation, (2) dynamic reconfiguration, and (3) quality of service (QoS). Task allocation is performed before runtime. Its main objective is to compute an optimal mapping of tasks to the available resources. We have modeled the task allocation as a constraint satisfaction problem (CSP) [12]. A solution of the CSP corresponds then to an allocation of tasks to cameras. Dynamic reconfig-uration is performed online during the operation of the distributed system. The overall objective is to adapt the whole system to the changing needs, i.e., transfer the system into another configuration which better captures the current requirements. This typically involves starting and stopping specific services as well as moving tasks from one camera to another.

Dynamic reconfiguration requires some form of reasoning about the specific configuration. Thus, we need some information about the state of the current configuration as well as some objective for the next configuration. We have implemented dynamic reconfiguration in three steps. First, context sensing and analysis gathers information about the current state of the DSC network. The current state is retrieved using the monitoring service. Second, the computation of new configurations is done by an optimizer which uses the contextual information of the DSC network and the reconfiguration objective as input. In our implementation, the reconfiguration objective is specified by policies. Third, the reconfiguration enforcement performs the actual transfer from the current to the next configuration. This enforcement is implemented by our mobile agent system. All three steps are executed periodically in a so-called reconfiguration loop (cf. Figure 5).

Policy-based Middleware

The computation of new configurations is based on policies. Policies represent a collection of rules which have to be calculated to determine a potential new configuration. Thus, our policy-based approach can be separated into a policy-specification and a policy evaluation phase. As depicted in Figure 5, the policies are externally specified and serve as input to the reconfiguration loop. The evaluation of the policies has to be performed within the loop.
In order to get some output from the framework, the policies have to be evaluated. This evaluation is performed in a hierarchical way, from abstract objects (policies) over rules to conditions and actions. This evaluation chain is triggered whenever an event occurs and changes the corresponding parameters in the policy.
Policies can be changed online and enable different branching during the execution of scenarios. Scenarios represent successive reconfigurations of the DSC from one state to another. This enables the specification of more complex tasks in our DSC network. We have specified and tested several scenarios for various motion detection and tracking tasks.

Towards an Intelligent Multi-Sensor System

Fusing data from various sensors helps to improve the robustness and confidence, to extend the spatial and temporal coverage as well as to reduce ambiguity and uncertainty of the processed sensor data. In the I-SENSE project [13] we extend our SmartCam platform to embedded sensor fusion and demonstrate it on two case studies: In waste separation, sensor fusion helps to reduce the amount of non-reusable, disposable waste. In traffic surveillance, fusing sensor data from different sensors leads to increased detection rates which in turn can help to increase safety on the roads.
The goal of this project is to extend our SmartCam platform to a distributed high-performance multi-sensor architecture. Our multi-sensor architecture combines data from different sensors at two stages. First, intra-node fusion takes place at a single sensor node where raw sensor data or abstracted features are combined. Second, inter-node fusion combines abstracted data from various geographically distinct sensor nodes.
In our I-SENSE project not only methods for abstracting and fusing data but also architectural issues must be solved. These important architectural issues include both hardware and software features such as (i) computing and communication performance, (ii) scalability, autonomous operation and configuration, (iii) fault tolerance and graceful degradation as well as (iv) data abstraction and communication.

Multi-Sensor Data Fusion

In our I-SENSE project three basic categories (or levels) of data fusion are exploited. These fusion levels are differentiated according to the amount of information they provide. The most basic level is called *raw-data fusion*. At this level, only raw, uncorrelated data is provided to the user. In comparison, level two data fusion provides a higher level of inference and delivers additional interpretive meaning suggested from the raw data and data will be fused on feature level. Therefore, this level is called *feature-level fusion*. Level three data fusion is designed to make assessments and provide recommendations to the user and is called *decision fusion*. Thus, each jump between data fusion levels represents a corresponding leap in technological complexity to produce increasingly valuable informational detail.

Data fusion is performed on the individual sensor nodes using data from the local sensors as well as abstracted sensor data from the adjacent sensor nodes. In order to realize a flexible and scalable solution our multi-level fusion is based on the following constraints: (i) Sensor fusion is performed distributed in our I-SENSE platform, i.e., there is no (single) central fusion node. (ii) Communication of the abstracted sensor data is managed by the I-SENSE runtime environment over the entire network, transparent for the individual sensor node. (iii) The sensor nodes have no global knowledge about the network topology; they require only knowledge about their neighborhood.

Multi-Level Fusion Architecture

The I-SENSE platform consists of a two-level architecture which is an extension of our SmartCam architecture (cf. section „SmartCam Architecture“). The top-level is composed of geographically distributed sensor nodes which are connected via a common communication medium (cf. Figure 6). The sensor nodes represent the bottom-level of our I-SENSE platform. They are equipped with high-performance processing and communication elements and several heterogeneous and homogeneous sensors can be attached to them. Thus, the sensor nodes are the main processing components of our I-SENSE platform.

Scalability is supported in several ways in our I- SENSE project. First, the number and type of sensors can be adapted at the individual sensor nodes. Second, the processing and communication performance of each sensor node can be easily modified. Third, communication can be performed via wired and wireless connections and finally, the number of sensor nodes can be easily adapted.

The fusion nodes are logical entities on the sensor node that perform the sensor fusion algorithms at the individual fusion levels. These nodes receive (raw) data from the sensors and abstracted data from other fusion nodes, and provide data for other sensor nodes and data consumers. Data consumers are remote fusion nodes as well as visualization nodes where the combined sensor data can be monitored. The presence of multiple data sources and fusion nodes provides many choices for the architecture, i.e., how the sensors or data sources report to each fusion node and the connectivity among the nodes (cf. Figure 8).

In order to develop a generic HW/SW architecture for embedded data fusion the following challenges can be identified: (i) simple and expressive specification of the configuration problem like, (ii) effective online optimization, (iii) lightweight runtime environment on the distributed embedded platform supporting efficient synthesis and communication, and (iv) synchronization of intra- and inter-node sensor data. The first three challenges are general for distributed embedded systems whereas the last one is specific for distributed data fusion.

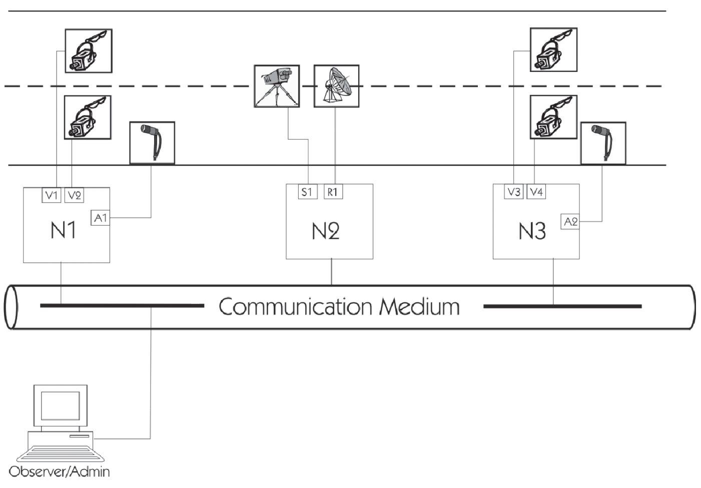

Fig.6: *The I-SENSE platform as a scalable, distributed embedded system. (N1 . . . N3: Sensor nodes, V1 . . . V4: Video interfaces, A1 . . . A2: Audio ports, R1: Radar interface and S1: Spectral imaging)*

The multi-level data fusion architecture for the I-SENSE project is presented in Figure 7. The three fusion methods (i) raw-data fusion, (ii) feature-level fusion and (iii) decision fusion are the heart of this architecture. The output of these fusion strategies is then combined to derive the current state of the fusion process.

The two bottom level methods combine data from sensor directly attached to the fusion node; the top level method uses state information from remote fusion nodes. In order to realize a flexible solution our multi-level fusion is based on the following functionalities:

- **Raw-Data Level Fusion:** Each sensor performs a single-source positional estimate in the sensor state space. These estimates are then combined to an aggregate estimate. Since raw-data fusion is performed in the raw-data space of the sensor, it can only be applied for similar sensor types (e.g., visual sensor and infrared visual sensor).
- **Feature Level Fusion:** Each sensor extracts a single-source state vector. This means, each sensor provides an estimate of the position and velocity of an object (cars, cargo, persons, etc.) or an observed situation, based only on its own single source data. These

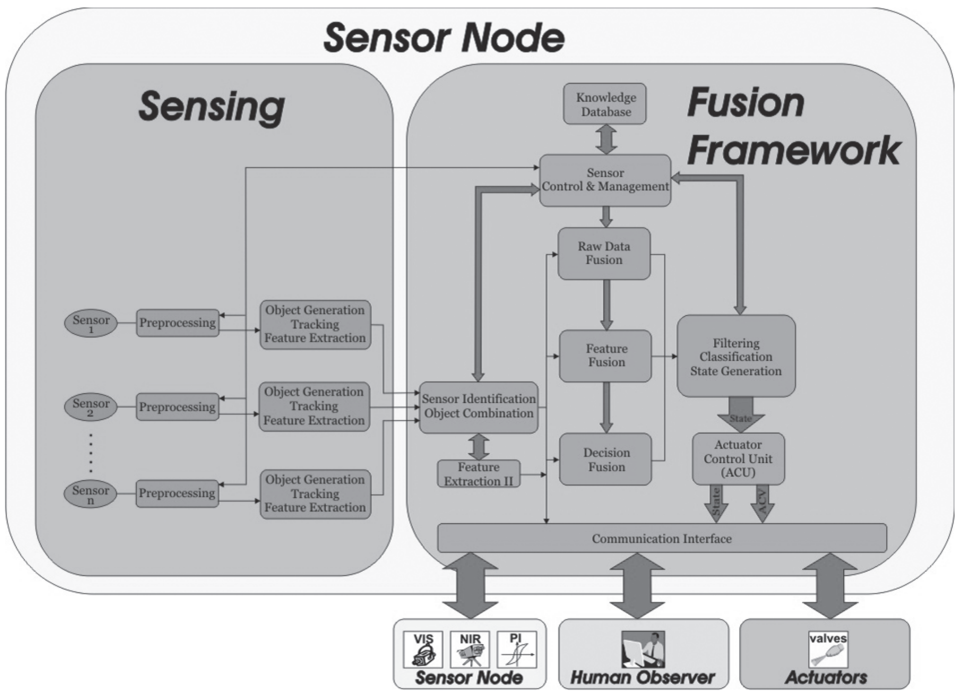
Sensor Node
Sensing
Fusion Framework
Knowledge Database
Sensor Control & Management
Raw Data Fusion
Feature Fusion
Decision Fusion
Filtering Classification State Generation
State
Actuator Control Unit (ACU)
State
ACU
Sensor 1
Sensor 2
Sensor n
Preprocessing
Preprocessing
Preprocessing
Object Generation Tracking Feature Extraction
Object Generation Tracking Feature Extraction
Object Generation Tracking Feature Extraction
Sensor Identification Object Combination
Feature Extraction II
Communication Interface
VIS
NIR
PI
Sensor Node
Human Observer
valves
Actuators

Fig.7: *Multi-level Data fusion architecture*

state vectors are input to a data fusion process to achieve a joint state vector estimate based on multiple sensors. A classifier, based on support vector machines, is trained with previously recorded and classified traffic sequences and furthermore used to derive a decision from previously fused feature vectors.

- **Decision Fusion:** At this level the derived states of different sensor nodes are combined. A state of a sensor node corresponds to a high-level assessment of the sensor's observed area. Examples for such an assessment in traffic surveillance include the detection of a traffic jam, the identification of a specific vehicle, or the detection of lost cargo. Both, the assessment of states as well as their combination requires an interpretation of the lower level fusion outputs and a lot of domain knowledge. In our research project we investigate two methods for combining the states. In the first method, the procedure for combining states is specified by domain-dependent rules. In the second method, we apply Dempster-Shafer based statistical decision fusion.

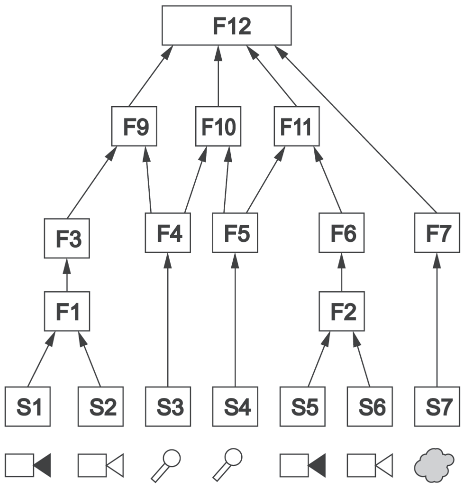

Fig.8: *Example of a simple fusion configuration*

Conclusion

Our SmartCam prototype is a major step in the development of a distributed embedded vision system. By migrating most of the computation from a central server or workstation to the smart embedded cameras, the system architecture becomes more flexible and scalable, the overall communication bandwidth is reduced and the vision system is able to autonomously and dynamically react to detected events in the supervised scenes. This novel vision architecture poses, however, strong requirements on the camera's hard- and software. Recent technological advances support this trend towards smart cameras. In a subsequent step we currently augment the smart cameras with additional sensors transforming them into a high-performance multi-sensor system. By combining visual, acoustic, tactile or location-based information, the smart cameras become more „sensitive" and are able to deliver more accurate results which make them even wider applicable. Embedded smart cameras have the potential for the successful deployment in many different applications such as smart environments, intelligent infrastructures and pervasive computing. We continue our work in this exiting field and are therefore looking for students interested in pursuing a project or thesis work on embedded vision.

References

[1] M. Bramberger, A. Doblander, A. Maier, B. Rinner and H. Schwabach. Distributed Embedded Smart Cameras for Surveillance Applications. IEEE Computer, vol 39, no. 2, pp 68-75, 2006

[2] W. Wolf, B. Ozer, and T. Lv. Smart Cameras as Embedded Systems. IEEE Computer, 35, no. 9, pp 48-53, September 2002.

[3] 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

[5] S. Fleck and W. Straßer, „Adaptive Probabilistic Tracking Embedded in a Smart Camera", in Proceedings of IEEE Embedded Computer Vision Workshop (ECVW) in conjunction with IEEE CVPR 2005, pp. 134-134.

[6] 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) in conjunction with IEEE CVPR 2005, vol 3, 2005, pp. 136-136.

[7] R. Kleihorst , B. Schueler, A. Danilin, M. Heijligers, „Smart Camera Mote With High Performance Vision System" In Proceedings of the Workshop on Distributed Smart Cameras (DSC-06), pp 17-21, Boulder, CO, USA, 2006

[8] I. Kim , S. Shim, J. Schlessman, W. Wolf, „Remote Wireless Face Recognition Employing ZigBee" In Proceedings of the Workshop on Distributed Smart Cameras (DSC-06), pp 125-129, Boulder, CO, USA, 2006

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

[10] G.R. Bradski, „Computer vision face tracking for use in a perceptual user interface", Intel Technology Journal, no. Q2, p15, 1998

[11] M. Bramberger, M. Quarisch, T. Winkler, B. Rinner, H. Schwabach, „Integrating Multi-Camera Tracking into a Dynamic Task Allocation System for Smart Cameras". In Proceedings of the IEEE International Conferece on Advanced Video and Signal Based Surveillance, Como, Italy, September 2005.

[12] M. Bramberger, B. Rinner, and H. Schwabach. A Method for Dynamic Allocation of Tasks in Clusters of Embedded Smart Cameras. In Proc. IEEE Conference of Systems, Man and Cybernetics, pp 2595-2600 Hawaii, U.S.A., Oct. 2005.

[13] Andreas Klausner, Bernhard Rinner, Allan Tengg. I-SENSE: Intelligent Embedded Multi-Sensor Fusion. In Proceedings of the 4th IEEE International on Intelligent Solutions in Embedded Systems (WISES 2006). Vienna, Austria. June 2006. ■

Milan Jovanovic, Andreas Klausner, Markus Quaritsch, Bernhard Rinner, Allan Tengg, Institute for Technical Informatics, Graz University of Technology, Austria, Email: {jovanovic, klausner, quaritsch, rinner, tengg}@iti.tugraz.at.