A Pervasive Smart Camera Network Architecture applied for Multi-Camera Object Classification

Wolfgang Schriebl, Thomas Winkler,
Andreas Starzacher and Bernhard Rinner

Pervasive Computing Group
Institute of Networked and Embedded Systems
Klagenfurt University, AUSTRIA
Email: {firstname.lastname}@uni-klu.ac.at

Abstract—**Visual sensor networks are an emerging research area with the goal of using cameras as pervasive and affordable sensing and processing devices. This paper presents a pervasive smart camera platform which is built from off-the-shelf hardware and software components. The hardware platform is comprised of an OMAP 3530 processor, 128 MB RAM and various interfaces for connecting sensors and peripherals. A dual-radio wireless network allows to trade communication performance for power consumption. The software architecture is built upon standard Linux and supports dataflow oriented application development by dynamically instantiating and connecting functions blocks. Data is transferred between blocks via shared memory for high throughput. We present a performance evaluation of our smart camera platform as well as a multi-camera object classification system to demonstrate the capabilities and applicability of our approach.**

I. INTRODUCTION AND MOTIVATION

Smart cameras, characterized by performing on-board video analysis, start to be adopted for real-world applications like assisted living [1]. Typically these systems rely on fixed infrastructure like Ethernet and mains power supply. Visual sensor networks represent an emerging trend in research, combining aspects of smart cameras and wireless sensor networks. A key factor for cameras to become truly pervasive sensors embedded in our environment, is a reduction of infrastructure requirements. The less fixed infrastructure is necessary and the less manual configuration is needed, the easier end users will be able to deploy such systems.

As power consumption is an important issue, visual sensor networking platforms often are specifically designed for a certain purpose, exposing only limited interface options as well as using low performance components. While those aspects are important when it comes to actual deployment, they are problematic when using such a system for research and development. One approach to overcome this issue is to rely on simulation during research but we believe it can not fully replace practical evaluations. For that reason we implemented a pervasive smart camera prototype based on standard, off-the-shelf components. While our approach is not as optimized for low power consumption as specifically designed systems are, it allows us to easily change parts or extend our system to evaluate new ideas.

As individual nodes in a visual sensor network only observe a limited area and are constrained in computational

capabilities, cooperation among cameras is required. Over time, different software frameworks have been proposed for distributed applications and wireless sensor networks. Many frameworks like CORBA, DCOM or RMI are designed to be used on workstations or other high performance systems and therefore are heavyweight or too complex for use in embedded environments. Sensor network middleware [2] [3] on the other hand, typically is designed for handling small amounts of data and does not scale well in the context of smart cameras where large image data has to be processed on the nodes and occasionally also exchanged between nodes. As existing middleware systems do not properly fit the domain of visual sensor networks, we propose in this work a simple and lightweight framework specifically addressing the needs of pervasive smart cameras.

The remainder of this paper is organized as follows: Section II presents selected platforms used in visual sensor networking as well as middleware systems targeted at smart cameras. In sections III and IV we present our prototype hardware platform and our software framework. Section VI presents details of a sample application we implemented and which is evaluated in section VII. Section VIII finally concludes the paper.

II. RELATED WORK

Several platforms for visual sensor networks have been developed. Rinner et al. [4] and Akyildiz et al. [5] present comprehensive overviews.

Citric [6] represents a more recent platform where a custom camera module featuring a PXA270 CPU at 624MHz and 64MB RAM is combined with a Telos Sky mote that provides 802.15.4 network connectivity. For application management, a client/server scheme is implemented where users can logon to nodes and assign tasks. The user that first logs on to a node becomes the manager and has full control until logout.

The design of middleware addressing the requirements of smart cameras has been discussed in [7] where the mobile agent paradigm is used for application development. An agent represent a specific task consisting of a low-level computer vision part executed on a DSP and a high-level part concerned with reasoning and decisions aspects. The agent paradigm allows applications to freely move from camera to camera following e.g. a tracked object. Aside from the split into the

agent logic and the image processing parts, applications are not further broken down into reusable sub-components. Initial versions of the agent system implemented in Java have been replaced by a C version for increased performance.

Doblander et al. [8] focus on the system-level software required for efficient streaming applications on single smart cameras as well as on networks of distributed smart cameras. Their software framework is based on a publisher-subscriber architecture and provides mechanisms for dynamically loading and unloading software components as well as for graceful degradation in case of software- and hardware-related faults. This framework has been implemented on multi-processor smart cameras.

The ASAP middleware [9] is a scalable distributed architecture for a multi-modal sensor network. Features of this architecture include the generation of prioritization cues that allow the infrastructure to pay selective attention to data streams of interest; the virtual sensor abstraction that allows easy integration of multi-modal sensing capabilities; and the dynamic redirection of sensor sources to distributed resources.

In [10] the HIVE middleware is presented that supports the formation of vision processing pipelines. Basic components or services are called drones which can be composed to form applications called swarms. Drones are implemented as processes which communicate with other drones in their swarm using an event model. Data produced by a drone is wrapped as an event which then is distributed using TCP/IP to the drones subscribed to that event. Directly sharing memory between drones on the same devices seems to be not supported.

Scallop [11] is a framework for distributed vision sensor networks based on peer-to-peer concepts. The system is implemented using the .NET framework and is intended to be used on PCs and workstations. System configuration as well as messaging has been implemented based on XML. Consequently, binary data such as images is encapsulated in XML messages for transmission. Messages and events are delivered to consumer asynchronously using callbacks.

III. THE PSC HARDWARE ARCHITECTURE

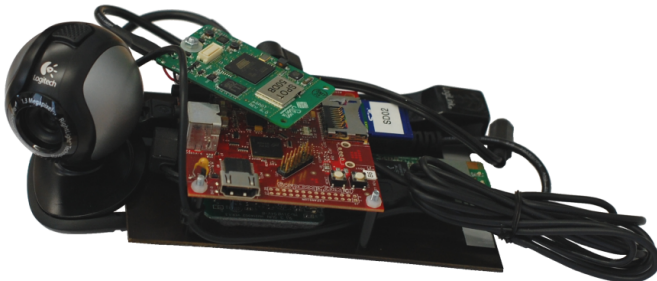Figure 1 shows our pervasive smart camera platform.

Fig. 1. A Pervasive Smart Camera prototype based on an embedded processing board, a webcam, an 802.11 radio and a mote for 802.15.4 wireless connectivity.

The platform is based on the Beagleboard[1] equipped with

[1]http://www.beagleboard.org (March 2009)

an OMAP 3530 processor from Texas Instruments. The processor is based on an ARM Cortex-A8 clocked at 480MHz plus an additional TMS320C64x+ digital signal processor running at 430MHz. The system provides 128MB RAM and 256MB NAND flash. Peripherals can be attached via USB, I2C, SPI, DVI as well as stereo in/out. In our setup, USB is used to connect a Logitech QuickCam S5500, an RA-Link 802.11b/g WiFi adapter as well as a SunSPOT mote providing 802.15.4 wireless connectivity. For development and debugging purposes, the nodes additionally are equipped with USB to Ethernet adapters as there is no Ethernet interface on-board.

In the field of visual sensor networks, power consumption is an important aspect. Using WiFi networking allows us to occasionally transmit video streams which can be useful to inspect and evaluate events reported by the camera network. During normal operation however, where only small amounts of data are exchanged between cameras for control and co-ordination, WiFi is too power intensive. For that reason, in [12] we have proposed the approach of equipping our cameras with an additional 802.15.4 radio. The resulting dual radio network allows us to trade communication performance for power consumption based on the actual requirements of the application. An example of such a dual-radio network is shown in figure 2. Since achievable communication distances for 802.15.4 networks typically are lower than those of 802.11, the network is augmented with nodes only equipped with 802.15.4 radios which are used for packet forwarding.
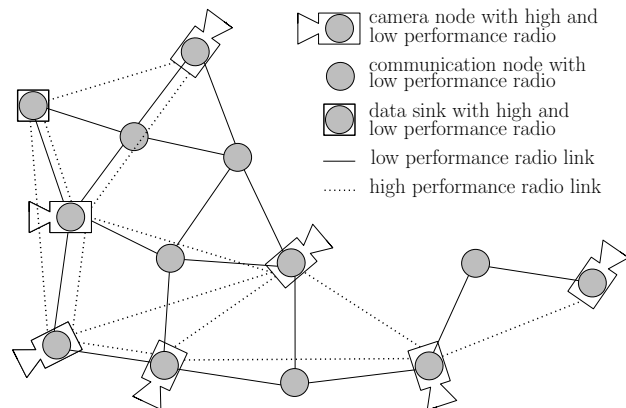

Fig. 2. PSC networking architecture. Camera nodes are equipped with two radios (high and low performance) while intermediate nodes are equipped with only a single, low-performance radio.

As an operating system a Debian GNU/Linux distribution compiled for the ARM platform together with an OMAP specific kernel is used. Without any optimizations, the system currently requires 500MB of the 8GB SD card it is stored on.

IV. THE PSC SOFTWARE ARCHITECTURE

To simplify application development and to allow re-use of components, a software framework has been designed that supports composition of applications from individual blocks which are instantiated and interconnected at runtime. The

selected approach for the middleware framework follows the concept of modeling the dataflow between the individual components.

Conceptually, every block has an output memory where its results can be accessed by subsequent blocks. To maintain consistency of the stored data, access to the memory is guarded by a lock that is passed between the producing and consuming block similar to a token. Blocks can form chains of arbitrary length where each pair of blocks is connected by a shared memory and a lock. In our implementation, a block as shown in figure 3, is realized as an individual operating system process expecting well-defined input data and generating output consumable by subsequent blocks. The shared memories are implemented as POSIX shared memory synchronized by an inter-process locking mechanism.
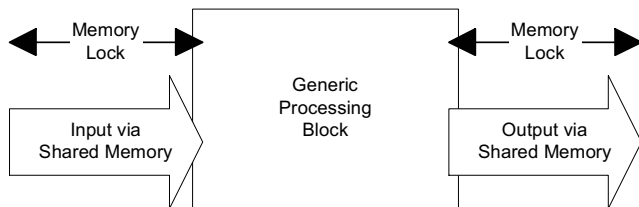

Fig. 3. In the PSC framework, applications are composed of single function blocks that communicate using shared memory.

Each block is accompanied by a block description file containing information about the input the block expects, possible parameters including descriptions of legal values as well as a description of the format of the output that is generated by the block. This information is vital when blocks get instantiated and processing chains are formed. A block description example is given in figure 4.

Using separate processes instead of threads for the processing blocks offers a number of benefits. Blocks can be implemented in any programming language as long as there exists shared memory and locking support. This allows to e.g. use native code in places where performance is critical such as low-level data processing and to rely on higher level languages for e.g. statistics generation, system configuration and networking aspects. Moreover, separate processes allow to more easily implement watchdog functionality that monitors individual parts of the processing chain and restarts blocks as required.

Processing blocks do not directly support multiple consumers for their output memory. To overcome this limitation, a central entity running on every camera node called the *NodeManager* is introduced. Once a block has written its data to its output memory, it passes the lock to the *NodeManager* who in turn gives the lock to all registered consumer blocks which then can perform parallel read access to the memory. Once all consumer blocks have returned the lock, the *NodeManager* passes the lock for the shared memory to the producer block which can now fill it with new data. Note that the producer block does not necessarily have to be idle while not holding the memory lock but it can continue with

```
# General section with block name and class.
general = {
  'serviceName'   : 'CameraCapture',
  'serviceClass'  : blkconf.CLS_IMG_INPUT,
  'executable'    : 'camera_capture',
}

# Additional block configuration options
# not directly related to input or output.
config = {
  'nodeman'       : blkconf.BooleanType
}

# Capture block has no further inputs.
inputs = {
}

# Data format for output buffer.
outputOptions = {
  0 : {
    'resolution'  : blkconf.IMG_RES_640_480,
    'format'      : blkconf.IMG_FMT_RGB,
    'fps_max'     : 10,
    'outmem_size' : 1228800,
  },
  1 : {
    'resolution'  : blkconf.IMG_RES_320_240,
    'format'      : blkconf.IMG_FMT_RGB,
    'fps_max'     : 20,
    'outmem_size' : 307200,
  }
}

outputs = {
  # Block only supports one output buffer.
  0 : outputOptions
}
```

Fig. 4. A sample description of a block that captures images from a sensor: Blocks can be looked up and instantiated either by their name or the class they belong to. Additionally, the block description holds the name of the executable, configuration options as well as the input requirements and the output options to choose from upon block creation.

internal processing as required. As shown in figure 5, there exists exactly one *NodeManager* per camera. Per definition, the *NodeManager* not only is responsible for lock management but also is the only entity that creates new block instances. This allows it to keep track of running blocks and their connections. Additionally, the *NodeManager* monitors the available system resources and can decide whether creation of additional blocks is allowed or not.

In addition to block creation and lock management, the *NodeManager* offers the following additional services which are exposed by means of a remote procedure call interface.

Query Service: A client can query what blocks are currently running on a node and which other blocks are available and could be instantiated. The block description includes the resources required by a block, required inputs and parameters as well as its outputs.

Resource Monitoring: The *NodeManager* constantly monitors the available system resources. This information is re-
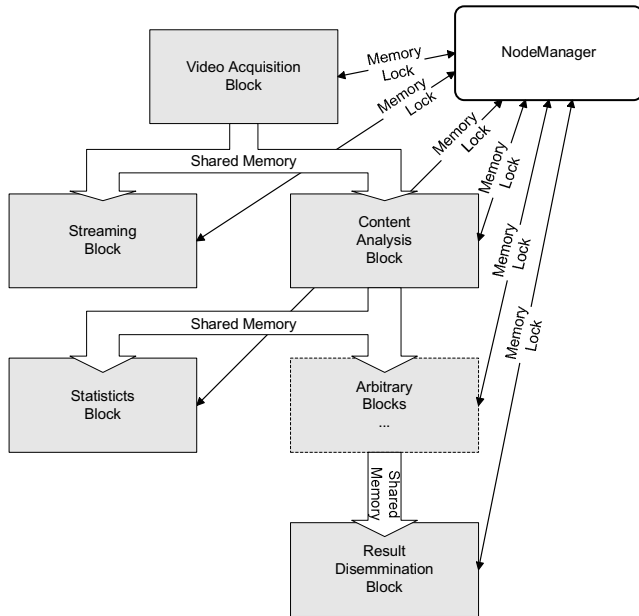
Fig. 5. The *NodeManager* is responsible for creating processing chains and lock management. The output of individual blocks is stored in a shared memory that can be accessed by one or more consumer blocks. Final results of processing chains typically are made available as a subscribable service which can be consumed by other nodes or client applications.

quired do determine if additional blocks can be instantiated on the node.

Processing Chain Management: In this framework, an actual application is composed of multiple blocks forming a processing chain. These chains are instantiate and managed by the *NodeManager* based on an application description containing the required blocks and their appropriate sequence. As interaction between processing blocks is based on cooperative behavior, the *NodeManager* should ensure that blocks e.g. return locks within a predefined timeframe. Additionally, blocks are expected to send a periodic life-beat to the *NodeManager* used by a watchdog to restart processes if required.

Remote Subscription Service: Typically, output data produced by blocks is consumed locally on the same node the block is running. In case of a distributed processing scenario where intermediate results are required by other hosts or data can not be processed locally due to resource constraints, output shared memories can be accessed remotely. If a processing chain makes use of such remote data, the *NodeManager* of the system the producing block is located on, is responsible for copying the data via the network to a shared memory located at the host of the consumer block. To maintain synchronicity of the processing chain, the memory lock is also passed via the network and returned once the consumer block is finished. From the consumer blocks perspective, access to a remote shared memory is not different from accessing a local shared memory as memory copying is transparently managed by the involved *NodeManagers*.

V. FRAMEWORK EVALUATION

In this section we present an evaluation of the proposed framework. For that purpose, special performance testing blocks are used which behave as follows: A producer block acquires the lock for its output shared memory, copies the specified amount of data to the shared memory and releases the lock. A consumer block consumes the data of the shared memory by copying the shared memory to a local buffer and then returns the lock. As the copied data is not processed further and therefore no additional computing time is required, the performance testing blocks are well suited to evaluate the performance of the framework with respect to shared memory and lock management. Producer and consumer blocks are implemented in C using the C standard library and are running as native processes.

Four scenarios are chosen for evaluating different aspects of the framework, namely single consumer with and without *NodeManager*, multiple consumers and single remote consumer. To get comparative values for the framework performance, the measurements were also carried out using a recent PC platform. Figures 6, 7, 8 and 9 present the measurement results numerically and graphically.

1) Single Consumer without NodeManager: Figure 6 presents the results for two blocks directly connected together without intermediate *NodeManager*. The times involve acquiring the lock, copying data and releasing the lock for both blocks. It is dominated by the overhead for locking and unlocking with about $150\,\mu s$ for small shared memories, and grows linearly for memory sizes above $50\,\text{kB}$.

2) Single Consumer: Figure 7 presents the results for two blocks interconnected via *NodeManager*. Compared to the scenario without *NodeManager*, the times also involve lock transfers to and from the *NodeManager* as well as administrative overhead. On the camera platform this leads to an overhead between 200 and $400\,\mu s$ for the investigated memory sizes.

3) Multiple Consumers: Figure 8 presents the results for one producer block and 16 consumer blocks. Compared to scenario 2, the times involve lock handling and data consumption of 15 additional consumers. On the camera platform this leads to a runtime increase of factor 9 to 10 for all evaluated memory sizes.

4) Single Remote Consumer: Figure 9 presents the results for two blocks connected remotely via wireless network. The times involve the transfer of lock messages, unlock messages and shared memory via network. For 802.11, the constant overhead for locking dominates below $1\,\text{kB}$, with nearly increasing times above $10\,\text{kB}$. The times for 802.15.4 reflect the lower bandwidth and higher latency compared to 802.11, resulting in nearly linear growing transfer times for memory sizes above 100 Bytes. Compared to the local communication scenario, the overhead for locking and unlocking is about $1.5\,\text{ms}$ when using 802.11, and $17.1\,\text{ms}$ when using 802.15.4.

	1 Byte	1 kB	10 kB	50 kB	100 kB	1 MB
PSC	0.148	0.153	0.215	0.491	0.928	10.142
Desktop	0.038	0.040	0.042	0.058	0.076	1.356
	all values given in [ms]					

(a) Times for memory transfer including lock acquisition and release between two directly connected blocks. The PSC figures were measured on the camera platform, the Desktop figures on an 1.6 GHz Core2Duo machine.
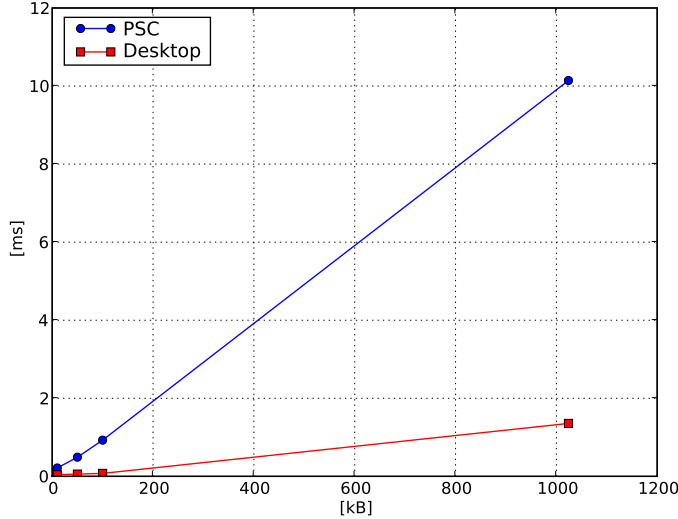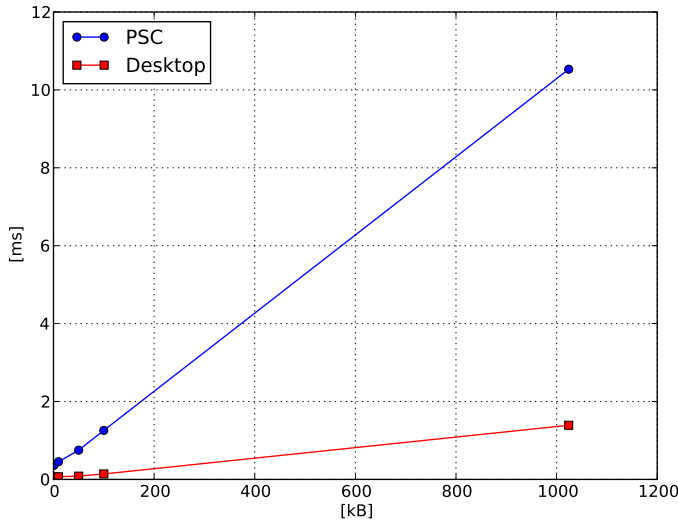

(b) Graphical representation of the times given in table 6(a).

Fig. 6. Direct block to block communication without *NodeManager*s.

	1 Byte	1 kB	10 kB	50 kB	100 kB	1 MB
PSC	0.359	0.368	0.454	0.751	1.258	10.530
Desktop	0.062	0.063	0.068	0.084	0.140	1.390
	all values given in [ms]					

(a) Times for memory transfer including lock acquisition and release between two blocks connected by an intermediate *NodeManager*. The PSC figures were measured on the camera platform, the Desktop figures on an 1.6 GHz Core2Duo machine.
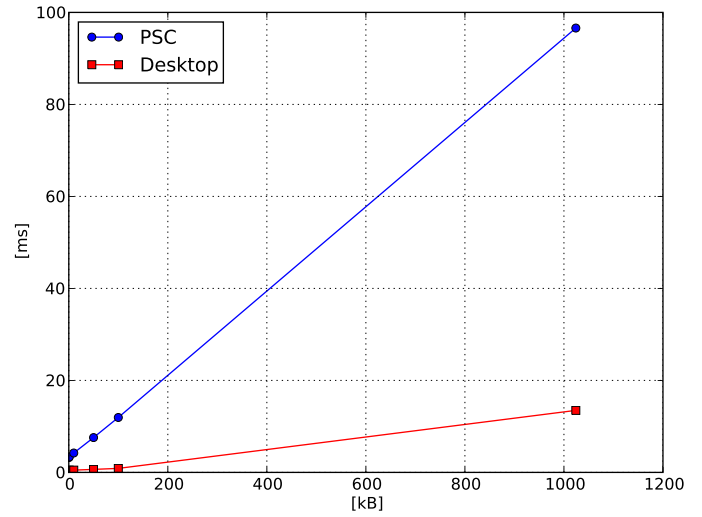

(b) Graphical representation of the times given in table 7(a).

Fig. 7. Block to block communication via *NodeManager*. Comparing with figure 6, results show that the overhead introduced by the intermediate *NodeManager* are about 1 ms on the camera platform.

	1 Byte	1 kB	10 kB	50 kB	100 kB	1 MB
PSC	3.216	3.364	4.215	7.572	11.940	96.599
Desktop	0.458	0.464	0.495	0.655	0.872	13.480
	all values given in [ms]					

(a) Times for memory transfer including lock acquisition and release between one producer block and 16 consumer blocks with an intermediate *NodeManager*. The PSC figures were measured on the camera platform, the Desktop figures on an 1.6 GHz Core2Duo machine.
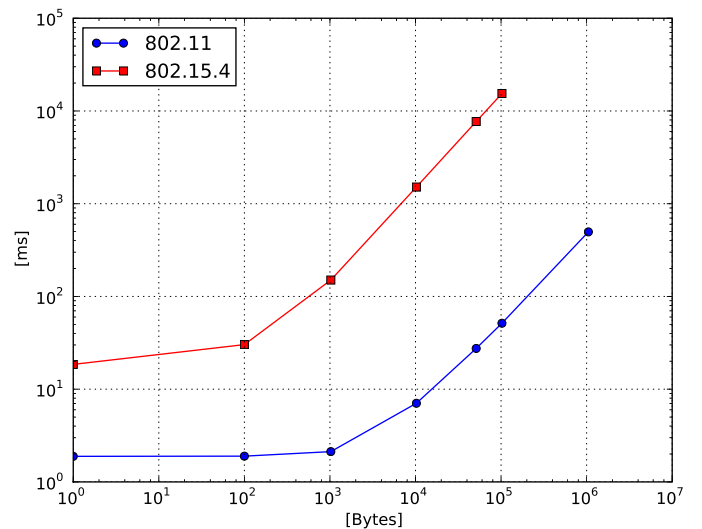

(b) Graphical representation of the times given in table 8(a).

Fig. 8. Communication between one producer and 16 consumers via an intermediate *NodeManager*.

	1 Byte	100 Bytes	1 kB	10 kB	50 kB	100 kB	1 MB
802.11	1.89	1.90	2.13	7.05	27.51	51.42	496.77
802.15.4	17.51	30.42	150.23	1.51 s	7.69 s	15.42 s	-
	all values given in [ms] if not stated otherwise						

(a) Times for memory transfer including lock acquisition and release between two blocks located on different cameras involving two *NodeManager*s. Communication performance was evaluated with 802.11b/g and 802.15.4 radios.

(b) Graphical representation of the times given in table 9(a).

Fig. 9. Communication between two blocks over the wireless network.

VI. Case Study: Cooperative Classification

The proposed software architecture is based on processing blocks which produce and consume data in a distributed manner. Although abstracting the network, applications must be designed with low-bandwidth networking in mind, therefore information should be abstracted in-node to reduce communication overhead inbetween nodes. To demonstrate and evaluate the in-network processing behavior of the architecture as well as local capabilities of the camera nodes in a real-world scenario, we implemented an application for cooperative person classification using multiple uncalibrated cameras.

The goal of multi-camera person classification is to improve the detection rate achieved by a single smart camera using object information from multiple views. The level of abstraction at which data is distributed among nodes should be high to keep communication efforts low and processing distributed. The application for this case study uses decision level as object abstraction. Decisions about the class of a person are generated locally by each camera, and these decisions are fused at a single node to come to a final classification result.

The application shows two aspects typically found in the area of pervasive smart cameras: (1) In-node image processing and fusion of local features with high demands in processing and memory capabilities and (2) in-network processing on a high level of data abstraction with demands in inter-node communication capabilities.
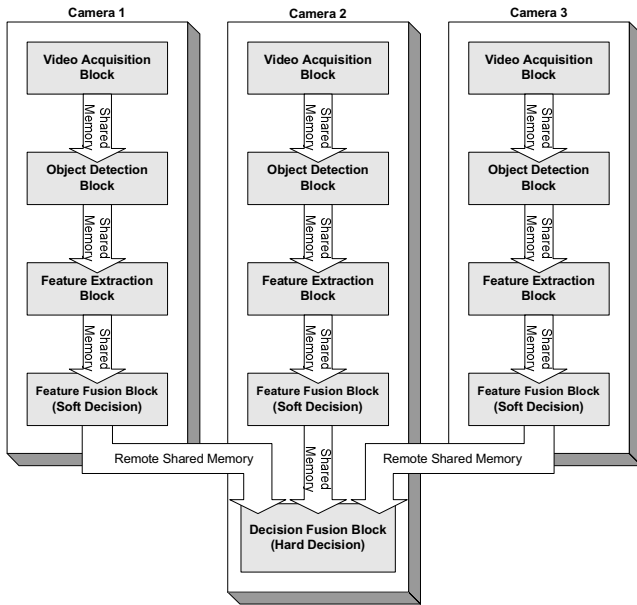
Architecture and Implementation

Fig. 10. This figure shows the application structure used in the person classification evaluation scenario. The setup consists of three cameras each running a chain of video acquisition, object detection, feature extraction and soft decision blocks. The camera tasked with the decision fusion is running an additional block that remotely access the decision output from the other cameras. To simplify the figure, the *NodeManagers* have been omitted.

The mapping of functions to building blocks of the software architecture is not strict, but is a trade-off between good reusability of blocks running simple functions and overhead caused by every block. Figure 10 gives an overview of the application and how the image processing and classification chain is mapped onto blocks. All blocks are implemented in C or C++, using the OpenCV [13] image processing library for image processing and classification.

The *video acquisition block* grabs images from the video sensor in YCrCb format and forwards the frames unprocessed to subsequent blocks.

The *object detection block* uses frame differencing with a color and lighting based adaptive background model to extract pixels of interest. After postprocessing the resulting foreground segments using median, erode and dilate, blobs larger than a threshold are labeled as object of interest. Only the largest object of interest is forwarded as person. The block does not distinguish between persons and other moving objects, and does not forward more than one person in the field-of-view at any given time, as those scenarios are not considered by the case study.

The *feature extraction block* extracts features distinctive for different persons. As an identical classifier should be able to run on different uncalibrated cameras in the network, the type of features must be tolerant to perspective and lighting as well as to scale and orientation of the objects. We have chosen the color distribution represented by the 2D histogram spanned by the chrominance channels of the object as features. Choosing a histogram resolution of 4 for each channel results in a feature vector of 16, which proved well-suited for our application with respect to accuracy and complexity.

The *feature fusion block* classifies a person resulting in a soft decision, which is the class the object most likely belongs to. Two different variants of this block, based on different classifier models are available in the system, namely *naive bayes* and *artificial neural network*. The classifier models are trained offline using features extracted from one or more cameras with an overlapping field of view. Assuming the use of features that do not depend on e.g. lighting conditions or camera position and calibration, a classifier trained with samples from a set of cameras can be used as common classifier for several cameras.

The *decision fusion block* computes a final hard decision about the class of an object. The soft decisions generated by each node are transmitted to this block running on a predetermined node, and a final hard decision is generated using majority voting. To further improve this approach, not only the detected classes but also the likelihoods for the soft decisions or a confidence weight for the cameras could be taken into account.

VII. Case Study Evaluation

At our institute, we have deployed a testbed consisting of several camera nodes as described in section III. The wall-mounted cameras are powered via Power-Over-Ethernet from a central switch. From there the nodes can be individually turned on and off. The Ethernet is not only used for power

supply but also for control and management to the camera nodes.

For the person classification application outlined in section VI, three cameras overlooking a common area have been selected. The cameras have not been calibrated. The first row of blocks in figure 11 presents images acquired by each of the three cameras with the same person present in the field of view of every camera. The second row of images shows the result of the object detection performed locally on every camera. The detected objects are then passed to the feature extraction blocks with their results shown in the third row of figure 11. In the last processing step that is performed locally on each camera, soft decisions are computed. The soft decisions are then transmitted via 802.15.4 to the designated node tasked with decision fusion, finally producing a hard decision as shown in the last row of figure 11. Note that the data presented in figure 11 corresponds to the outputs of the processing blocks of figure 10.

As the amount of exchanged data is very small, all inter-camera communication in this application is performed via 802.15.4. Specifically, the transmitted soft decisions easily fit into a packet payload of a few bytes containing a node identifier and the class of the detected object. This data can easily be transmitted in a single radio packet of the SunSpots providing a maximum payload of 95 bytes per packet. Transmission of such a single packet including involved memory locks, was measured to take $19\,ms$.
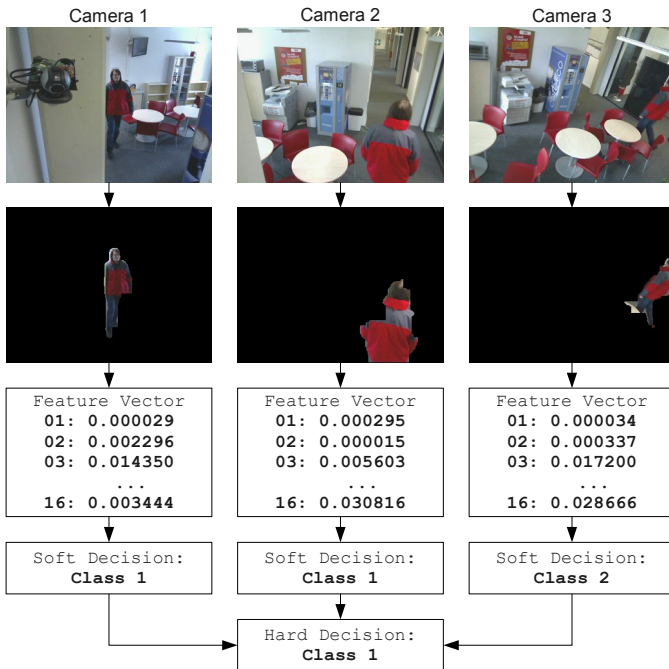

Fig. 11. The output data of the individual processing steps: Image acquisition, object detection, feature extraction and classification (soft decisions). In the last step, the soft decisions are transmitted to a predetermined node that fuses them into a common hard decision. The shown data corresponds to the output of the processing blocks of figure 10.

For classification, a Naive Bayes Classifier (NBC) and an Artificial Neural Network (ANN) are used. The classifiers are trained offline for three classes using a training set of 50 records per class for every camera. As an alternative to using local classifiers for every camera, a common classifier has been trained using a training set of 450 feature vectors (50 per camera per class). As the features used in this application have been chosen to be independent from lighting, perspective and scale of the observed objects, the common classifier should be usable for all cameras. This considerably increases the flexibility of the system as it allows to add cameras without requiring training of a specific classifier.

Figure 12 presents the classification results for a test set of 150 records. Local classification on a camera node, producing a soft decision for an incoming feature vector, takes $396\,\mu s$ using NBC and $490\,\mu s$ for ANN respectively.

	Naive Bayes	Neural Network
Camera 1	90.00 %	90.66 %
Camera 2	98.66 %	98.66 %
Camera 3	92.66 %	98.00 %
Fused Result	98.66 %	100.00 %

Fig. 12. Classification rates using a **local classifier** for each camera. The fused result represents the classification rate for the final hard decisions.

Figure 13 presents the classification results for the same test set of 150 records. Contrary to the previous case, a common classifier is used for all cameras instead of individually trained classifiers.

	Naive Bayes	Neural Network
Camera 1	93.33 %	90.66 %
Camera 2	92.66 %	95.33 %
Camera 3	88.66 %	98.00 %
Fused Result	98.00 %	99.33 %

Fig. 13. Classification rates with a **common classifier** used for all cameras. The fused result represents the classification rate for the final hard decisions.

To provide an overview of the times required for the individual steps of the processing chain, table 14 summarizes the results. Additionally, figures of the memory consumption for each step are given. These include the output shared memories of the blocks as well as estimates of the memory allocated internally by the blocks. Note that those buffers are allocated only once when the blocks are instantiated. In total, the propagation of one frame through the processing chain until a hard decision is reached is below $225\,ms$.

VIII. Conclusion and Future Work

In this work we presented a pervasive smart camera system built from off-the-shelf components, providing a flexible and expendable platform for research and development. Complementary to the hardware platform we designed and implemented a software framework supporting composition of applications based on well defined, reusable components. To demonstrate the feasibility of our approach we implemented and evaluated a multi-camera person classification application.

	Time	Memory
Image Acquisition	4.5 ms	545 kB
Object Detection	165.0 ms	5005 kB
Feature Extraction	30.0 ms	250 kB
NBC Feature Fusion	0.4 ms	332 kB
ANN Feature Fusion	0.5 ms	310 kB
802.15.4 Transmission	19.0 ms	n/a
Decisions Fusion	0.1 ms	15 kB
Node Manager	5.0 ms	2950 kB
Total	<225.0 ms	<9500 kB

Fig. 14. The figures show the time consumed by the individual components running on the camera platform for one frame passing through the processing pipeline. Within a processing chain either NBC or ANN are performed. Memory is not allocated for every frame but initially when the blocks are created. The time consumed by the *NodeManager* covers all shared memory and lock management operations. Since the *NodeManager* is implemented in Python, its memory footprint also includes the Python runtime environment.

While the system presented in this work forms a solid basis, there are numerous directions for ongoing research including:

System Autonomy: In the demonstrated application, the classifiers are trained offline and the node performing decision fusion is predetermined. Ideally the system should be able to dynamically determine the roles of the nodes at runtime as well as provide basic online learning support to be able to adapt to a changing environment.

Asynchronous Communication: In the proposed framework, processing chains are designed to work fully synchronous. While this is a desired feature for many applications, there are situations where asynchronous behavior is preferred. In cases where a block e.g. has multiple blocks consuming its output, the producer can not provide new data until all consumers have completed their task. The overall performance is dictated by the slowest block. An asynchronous inter-block communication mechanism should be added that allows blocks to communicate without blocking each other.

Power Consumption Evaluation: In this work, the important issue of power consumption is not addressed. In ongoing work, the power consumption of the system under different load conditions as well as the communication power requirements will be evaluated.

Performance Enhancements: Up to now, all computations are performed on the ARM core. The presented results could be improved by taking advantage of the NEON floating point unit which could considerably improve OpenCV based image processing which heavily used floating point operations.

Additionally, the DSP core that is part of the Cortex-A8 is not yet used. One approach would be to move certain tasks like low level image processing to the DSP. The component based system architecture should allow to easily incorporate DSP blocks into the processing chains.

References

[1] S. Fleck and W. Strasser, "Smart Camera Based Monitoring System and Its Application to Assisted Living," *Proc. IEEE*, vol. 96, no. 10, pp. 1698–1714, Oct. 2008.

[2] M. Molla and S. Ahamed, "A Survey of Middleware for Sensor Network and Challenges," in *Proc. of the 35th International Conference on Parallel Processing Workshops (ICPPW)*, Columbus, Ohio, USA, Aug. 2006, pp. 1–6.

[3] I. Chatzigiannakis, G. Mylonas, and S. Nikoletseas, "50 Ways to build your Application: A Survey of Middleware and Systems for Wireless Sensor Networks," in *Proc. of the IEEE Conference on Emerging Technologies & Factory Automation (ETFA)*, Patras, Greece, Sep. 2007, pp. 466–473.

[4] B. Rinner, T. Winkler, W. Schriebl, M. Quaritsch, and W. Wolf, "The Evolution from Single to Pervasive Smart Cameras," in *Proc. of the 2nd ACM/IEEE International Conference on Distributed Smart Cameras (ICDSC)*, Stanford, California, USA, Sep. 2008, pp. 1–10.

[5] I. F. Akyildiz, T. Melodia, and K. R. Chowdhury, "Wireless Multimedia Sensor Networks: Applications and Testbeds," *Proc. IEEE*, vol. 96, no. 10, pp. 1588–1605, Oct. 2008.

[6] P. Chen, P. Ahammad, C. Boyer, S.-I. Huang, L. Lin, E. Lobaton, M. Meingast, S. Oh, S. Wang, P. Yan, A. Y. Yang, C. Yeo, L.-C. Chang, J. Tygar, and S. S. Sastry, "CITRIC: A Low-Bandwidth Wireless Camera Network Platform," in *Proc. of the 2nd ACM/IEEE International Conference on Distributed Smart Cameras (ICDSC)*, Stanford, California, USA, Sep. 2008, pp. 1–10.

[7] M. Quaritsch, B. Rinner, and B. Strobl, "Improved Agent-Oriented Middleware for Distributed Smart Cameras," in *Proc. of the 1st ACM/IEEE International Conference on Distributed Smart Cameras (ICDSC)*, Vienna, Austria, Sep. 2007, pp. 297–304.

[8] A. Doblander, A. Zoufal, and B. Rinner, "A Novel Software Framwork for Embedded Multiprocessor Smart Cameras," *ACM Transactions on Embedded Computing Systems*, vol. 8, no. 3, pp. 1–30, Apr. 2009.

[9] J. Shin, R. Kumar, D. Mohapatra, U. Ramachandran, and M. Ammar, "ASAP: A Camera Sensor Network for Situation Awareness," in *Proc. of the 11th International Conference on Principles of Distributed Systems (OPODIS)*, Guadalupe, French West Indies, Dec. 2007, pp. 31–47.

[10] A. Afrah, G. Miller, D. Parks, M. Finke, and S. Fels, "HIVE: A Distributed System for Vision Processing," in *Proc. of the 2nd ACM/IEEE International Conference on Distributed Smart Cameras (ICDSC)*, Stanford, California, USA, Sep. 2008, pp. 1–9.

[11] P. Saastamoinen, S. Huttunen, V. Takala, M. Heikkilae, and J. Heikkilae, "SCALLOP: An Open Peer-to-Peer Framework for Distributed Sensor Networks," in *Proc. of the 2nd ACM/IEEE International Conference on Distributed Smart Cameras (ICDSC)*, Stanford, California, USA, Sep. 2008, pp. 1–9.

[12] T. Winkler and B. Rinner, "Pervasive Smart Camera Networks exploiting heterogeneous wireless Channels," in *Proc. of the IEEE International Conference on Pervasive Computing and Communications (PerCom)*, Galveston, Texas, USA, Mar. 2009, pp. 296–299.

[13] G. R. Bradski and A. Kaehler, *Learning OpenCV - Computer Vision with the OpenCV Library*, 1st ed. O'Reilly, Oct. 2008.