Distributed Multilevel Data Fusion for Networked Embedded Systems

Andreas Klausner, Allan Tengg, and Bernhard Rinner, *Senior Member, IEEE*

***Abstract*—Recently much research has been conducted in visual sensor networks. Compared to traditional sensor networks, vision networks differ in various aspects such as the amount of data to be processed and transmitted, the requirements on quality-of-service, and the level of collaboration among the sensor nodes.**

This paper deals with sensor fusion on visual sensor networks. We focus here on methods for fusing data from various distributed sensors and present a generic framework for fusion on embedded sensor nodes. This paper extends our previous work on distributed smart cameras and presents our approach toward the transformation of smart cameras into a distributed, embedded multisensor network.

Our generic fusion model has been completely implemented on a distributed embedded system. It provides a middleware which supports automatic mapping of our fusion model to the target hardware. This middleware features dynamic reconfiguration to support modification of the fusion application at runtime without loss of sensor data. The feasibility and reusability of the *I-SENSE* concept is demonstrated with experimental results of two case studies: vehicle classification and bulk good separation. Qualitative and quantitative benefits of multilevel information fusion are outlined in this article.

***Index Terms*—Distributed embedded systems, middleware, sensor fusion, vehicle classification.**

I. Introduction

PROGRESS in technology has facilitated the development of advanced distributed sensor networks. Recently much research has been conducted in visual sensor networks which perform image processing on distributed sensor nodes. Compared to traditional sensor networks visual sensor networks differ in various aspects such as i) the amount of data to be processed is much higher, ii) data is streamed through the network requiring specific quality-of-service (QoS), and iii) high-level collaboration among nodes is performed. By migrating resource intensive preprocessing tasks directly to the sensor nodes, the requirements concerning the communication bandwidth and delay may be relaxed compared to centralized architectures.

Manuscript received October 30, 2007; revised April 28, 2008. Current version published September 17, 2008. This work was supported in part by the Austrian Research Promotion Agency under Grants 812033 and 812204. The associate editor coordinating the review of this manuscript and approving it for publication was Dr. Wayne Wolf.

A. Klausner and A. Tengg are with the Institute for Technical Informatics, Graz University of Technology, A-8010 Graz, Austria (e-mail: andreas.klausner@tugraz.at; allan.tengg@tugraz.at).

B. Rinner is with the Pervasive Computing Group, Klagenfurt University, A-9020 Klagenfurt, Austria (e-mail: bernhard.rinner@uni-klu.ac.at).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/JSTSP.2008.925988

This paper deals with sensor fusion on visual sensor networks. Fusing data from various sensors helps to improve robustness and confidence, to extend spatial and temporal coverage as well as to reduce ambiguity and uncertainty of the processed data. We focus here on methods for fusing data from various distributed sensors and present a generic framework for sensor fusion on embedded systems. This paper extends our previous work on distributed smart cameras [1], [2] and presents our approach toward the transformation of smart cameras into a distributed, embedded multisensor network. Preliminary results of parts of this work have been presented at conferences ([3], [4]), however this paper comprehensively reports on this research for the first time.

There exist a large variety of multisensor fusion systems, but most of them are very application-specific (e.g., [5], [6]) or support only centralized data fusion (e.g., [7]). Our approach is focused on distributed sensor fusion performed in a *network of embedded sensor nodes*. However, our embedded nodes provide higher performance than typically found in sensor networks [8] but have tighter resource limitations than on general-purpose platforms.

The main contributions of this research can be summarized as follows:

- We introduce a generic fusion model—referred to as *I-SENSE*—which supports fusion at multiple levels, i.e., raw-data fusion, feature-based fusion and decision fusion. Our fusion model further considers the data flow in the sensor network as well as the resource restrictions on embedded systems. More specifically, the I-SENSE model accounts for data transfer costs in the distributed sensor network and provides a classifier dedicated for embedded systems, i.e., a least-square support vector machine with preselection of training data.
- The I-SENSE model represents the fusion application by a target hardware model and a software model. We have developed a light-weight middleware which supports the automatic mapping of our fusion model to the distributed embedded system. This middleware makes distributed processing transparent to the user and further features dynamic reconfiguration, i.e., the mapping of fusion tasks on the processing elements can be modified during runtime without loss of sensor data.
- We have evaluated our approach in two case studies, namely a traffic monitoring system and a bulk good separation. In these case studies we fuse visual data with other sensory data at multiple levels of data abstraction, distinguished by the amount of information they provide, and demonstrate the advantage of multisensor over

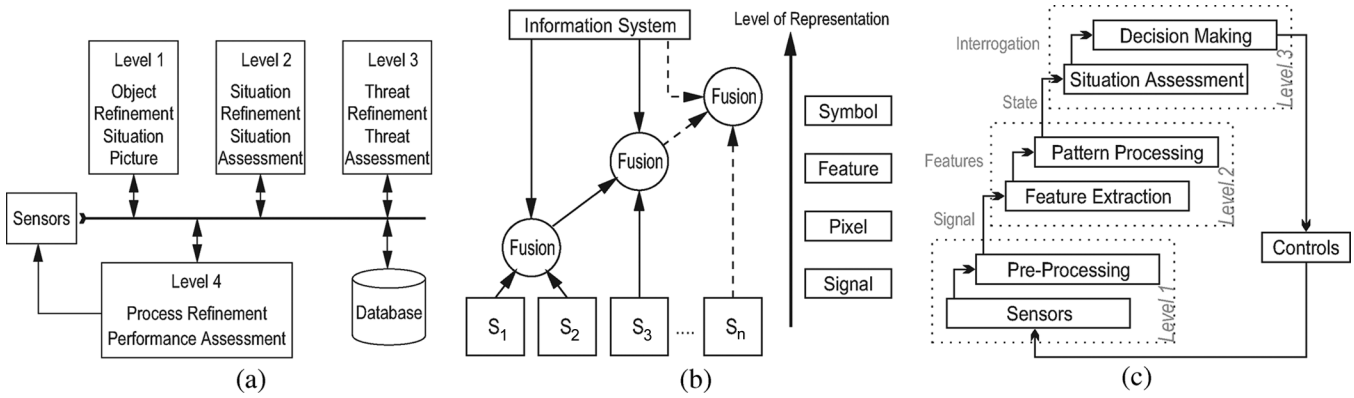

Fig. 1. Overview of the most popular models for data fusion. (a) JDL model; (b) Multi-sensor integration model; (c) Waterfall model.

single-sensor detection and classification. In our vehicle classification case study, this approach achieves an improvement from 90% to 96% compared to single sensor classification out of a data set of about 4000 vehicles. Applied to bulk good separation, our I-SENSE concept increases the overall classification accuracy from 86.8% to 98%.

The remainder of this paper is organized as follows: Section II reviews related work in the area of fusion models and frameworks. Section III presents our I-SENSE fusion model and briefly describes feature extraction, selection as well as fusion and decision modeling focusing on resource-constrained embedded systems. In Section IV, we present the I-SENSE middleware starting with an introduction of the available middleware services and some performance results. We then describe the specification of the hardware and software models as well as the applied method for optimizing the configuration on the distributed embedded system. Section V reports on the case studies of the I-SENSE framework to vehicle classification and bulk good separation. Section VI concludes the paper with a brief discussion and an outlook for future research.

II. Related Work

Over the last decades various data fusion models and frameworks have been developed—both in commercial as well as in research environments.

In the early years of data fusion the Joint Directors of Laboratories (JDL) within the US Department of Defense defined the *JDL data fusion framework* [9] (cp. Fig. 1(a)) which has been widely used. The main goal was to aid the developments in military applications. The JDL model describes a number of levels for data fusion. These levels include i) the location and identification of objects, ii) the construction of an image from incomplete information, iii) the provision of possible opportunities (i.e., prediction of effects on situations), and iv) the optimization of sensor allocations. A data management system for storage and human interaction is included as well.

In [10] an architecture for data fusion consisting of three modules, called *Thomopoulos architecture*, is proposed. These modules integrate data at three different levels, namely (i) *signal level fusion* where data correlation takes place through learning due to the lack of a mathematical model, (ii) *evidence level fusion* where data is combined based on a statistical model and the assessment required by the user and (iii) *dynamic level fusion* where fusion is performed with the aid of mathematical models.

[11] presents a *multisensor integration fusion model* (cp. Fig. 1(b)). In this system, data from various sources is combined in a hierarchical way within embedded fusion centers. A clear distinction between multisensor fusion and multisensor integration is stressed. Data collected at the sensor level is transferred to the fusion centers where the fusion process takes place. An information system, containing the relevant libraries and databases, facilitates the fusion process. The level of representation is increased from raw data to more abstract symbolic representations as the information is combined at the different fusion centers.

The *waterfall model* [12] (cp. Fig. 1(c)) is another example of a hierarchical architecture commonly used. The flow of data operates from the basic data level to the abstract decision making level. The system is therefore updated continuously with feedback information from the decision making model. These feedback elements advise the system on reconfiguration, recalibration and data gathering aspects. At the basic level information about the environment is gathered based on models of the sensors and whenever possible of the observed phenomena. Experimental analysis or physical laws are fundamental for those models. A symbolic level of inference about the data is obtained by means of feature extraction and accurate fusion. The aim of this stage is to minimize the data content while maximizing the delivered information. The output are estimates with associated probabilities of the observed objects. The highest level relates objects to events based on human interaction, databases and libraries.

Two interesting aspects regarding data fusion systems are given in the *distributed blackboard data fusion architecture* [13]. First, it assigns confidence levels to each sensor. Second, it refers to situations where conflicting sensor measurements occur. In this architecture, the sensors have a supervisor that controls the fusion process. The method for combining the statistical information provided by the sensor supervisors is taken from a database.

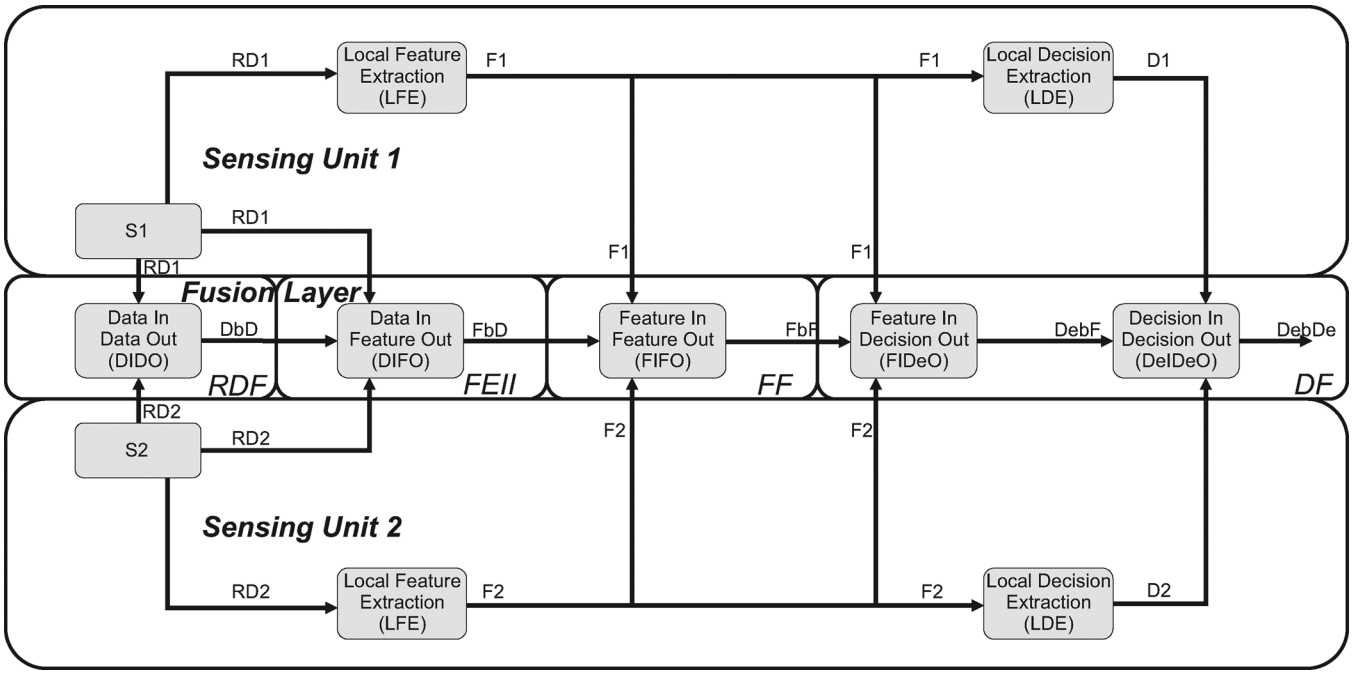

Fig. 2. The I-SENSE data-oriented fusion model. The functional units (blocks) and their input/output data are shown using two sensors $S1$ and $S2$.

The *Dasarathy model* [14] is based on fusion functions. These functions are characterized by the types of input and output data. Many researchers have identified the three main levels of abstraction during the data fusion process as decisions (symbols or belief values), features (intermediate-level information), and data (more specifically sensor data). Dasarathy pointed out that fusion not only occurs within these levels but also as a means of transformation between them.

The *Omnibus model* [15] is a hybrid model that overcomes some of the main limitations of the previous models while emphasizing on their advantages. The *Omnibus model* is used in two ways. First, it characterizes and subdivides the overall system aims to provide an ordered list of tasks. Second, the same structure may be used to organize the functional objectives of each such task. The cyclic nature of the data fusion process is made explicit. The constancy of representation expressed by the Waterfall model is incorporated into process tasks.

All presented models have one major drawback. There is no specification given how to handle factors such as the delay in the transmission of data, transmission errors as well the as spatial/temporal alignment of data to be fused. Our proposed approach supports data-fusion based on a light-weight middleware, specially designed to meet the needs of distributed data fusion applications on embedded systems based on *raw-data level*, *feature level* and *decision level*. The implementation of a specific data fusion application based on our architecture is simplified by providing methods for communication and configuration.

III. Fusion Framework

In this section we describe our multilevel fusion framework in detail. First we present the fusion model—specifying the dataflow and characterizing the software tasks based on the types of input and output data. Essential parts are identified as *fusion tasks*; a detailed description is given in the following sections.

A. Fusion Model

Fig. 2 presents the detailed, data-oriented software fusion model in our I-SENSE approach [16] for two physical sensors, labeled with $S1$ and $S2$ (e.g., an audio and a visual sensor). This model combines the ideas of the *JDL model* [9], [17], *Dasarathy's functional model* [14] and the *waterfall model* [12] to a generic and reusable model of a multilevel data-fusion process. Our model basically consists of three different layers: the *sensing unit*, the *fusion layer* and the *sensor control & management unit*. The first two units are shown in Fig. 2.

As the name implies the *sensor control & management unit* is responsible for the sensor identification as well as for providing the interface to other sensing nodes, human observers and actuators. Furthermore, this unit controls the overall fusion process and provides access to a database where resource requirements for the different fusion tasks are stored. This layer provides online refinement of the overall fusion process which is based on i) the generated output decisions and ii) the generated output features.

The *sensing units* represent the intelligent sensors which consist of physical sensors and a suitable data preprocessors (e.g., resolution based down-sampling, automatic gain control, etc.). A *local feature extraction unit (LFE)* is used to extract a single-source feature vector of an observed object. This means, that each sensor provides an estimate of the position of an object with extracted features based only on its own single source data. These individual feature vectors are input to a data fusion process, namely the *feature in feature out (FIFO)* process, in order to achieve a joint feature vector estimate based on multiple sensors. A *local decision extraction unit (LDE)* is

used to extract local decision from the individual objectives features (e.g., classification of objectives identity).

The heart of the framework is the *fusion layer* including the following five functional units:

- **Data in data out unit (DIDO)**. This functional unit is also called *raw-data fusion unit (RDF)* where raw uncorrelated data is fused from different and/or similar multiple sensors. These raw data streams are labeled *RDi*. For example, in our framework we apply wavelet based image fusion techniques for images from visual sensor and infrared spectral camera. The output of this unit is labelled *DbD*.
- **Data in feature out unit (DIFO)**. This is our so called *feature extraction II unit (FEII)*, where raw data from the individual sensors and/or fused raw-data is used to extract suitable features of the individual tracked objects. These features are identified by experimental analysis and/or physical modeling and are described in more detail in our case studies (Section V). The output data are feature vectors (FbD) for each detected object in the observed area.
- **Feature in feature out unit (FIFO)**. This is our so called *feature fusion unit (FF)*, where features are fused to an overall feature vector based on individual objects. Corresponding objects are found by simple computations of object overlaps for similar sensor types and time stamping for different sensors. The output data of this fusion process are fused feature vectors based on features (*FbF*) extracted by the *LFE* unit (*Fi*) or features extracted by the *DIFO* unit with an accurate feature selection stage.
- **Feature in decision out unit (FIDeO)**. This functional unit is part of our *decision fusion unit (DF)*, where a classifier based on *support vector machines* (SVM, cp. Section III-D) is trained with previously recorded and classified sequences. In the fusion step this SVM is used as a classifier to derive classification decisions based on previously extracted single source feature vectors or joint feature vectors from the *FIFO* unit. Decisions based on features and a probability interval of this decision serve as output of this stage (*DebF*).
- **Decision in decision out unit (DeIDeO)**. This functional unit is the second part of our *decision fusion unit*, where extracted decisions are fused from multiple sensors from the *LDE* unit (*Di*) with fused data from *FIDeO* based on Dempster-Shafer methods [18]. The output of this unit represents the overall output of our fusion model (*DebDe*).

B. Feature Extraction

Usually the raw data delivered by a sensor consists of much irrelevant information. By means of *feature extraction*, the input data is transformed into a reduced representation—the so called *feature vector*. If the appropriate set of features is chosen, the feature vector extracts the relevant information from the input data.

One of the main questions that arise is the aim of a feature extractor in the entire system. Usually a different set of features has to be used for object classification and object tracking. For example, if the task is to track a vehicle visually, color might be a very powerful feature. On the other hand, for vehicle classification color is often irrelevant.

C. Feature Selection—Feature Fusion

This section deals with the fusion of features from different sensors (cp. Sections V-A and V-C) and the selection of a suitable set out of a pool of candidate features. After feature generation often a very large number of candidate features must be reduced to a sufficiently small set as the SVM classifier can only handle a limited number of input features. Some of these candidate features may provide reliable class discriminatory information while others do not carry any relevant information and, hence, must be excluded as they could mislead the classifier. This task is not trivial since features that provide good classification information may only achieve little improvement when combined in a feature vector, due to high mutual correlation. In contrast, the combination of features with little class discriminatory abilities may achieve good results.

A genetic algorithm (GA) [19] is used as a search method. We provide two feature selection fitness algorithms based on (i) class separability criteria calculated for feature subsets and (ii) based on the classification result of the classifier itself.

1) Class Separability Measures: Different class separability measures have been developed as efficient feature selection criteria in various feature subset searching methods. The major drawback is that they do not always reflect the classifier behavior, and thus yield in only a suboptimal classification result. Better performance is usually achieved by including the classifier into the selection process and using the classification error rate directly as separability criterion. However, this step also includes a high computational cost. Therefore, separability measures are especially important when preselecting features out of a large set of candidate features. Two different measures are implemented in our model.

- **Bhattacharyya Distance**: The Bhattacharyya distance is derived from Bayes decision theory, which assumes a multivariate Gaussian distribution for the underlying probability densities.
- **Scatter Matrices**: Unlike the Bhattacharyya based measure, the scatter matrices criterion does not assume Gaussian probability distribution for individual features, but investigates how feature vector samples are scattered in the feature vector space.

2) SVM Classifier Error Measures: Our experiments have shown that the best way to select suitable class separation features from a set of candidate features is to use the classifier itself to obtain the classification error minimized by a GA. This method works as follows: For each feature vector combination, the classification error probability of the classifier is estimated and the one with minimum error is selected. That means increased complexity and computational demand for the feature selection process, but on the other hand direct inclusion of the classifier into the optimization process.

As the fitness function is minimized by the GA, the error rate of the classifier is interpreted as a reciprocal measure for the fitness. For each individual, the SVM is trained with a part of the database samples and then tested with the remaining part, yielding the error rate as fitness value. A single training and test run of the SVM does not necessarily lead to a reliable classification result, since feature data can be overlapped by noise

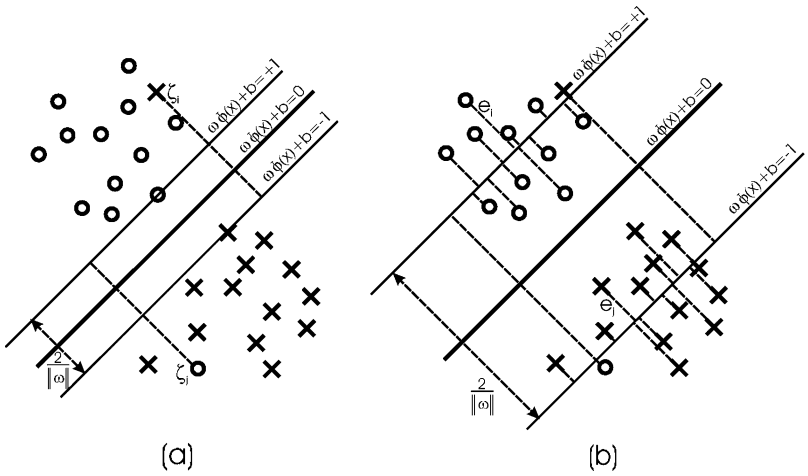

Fig. 3. Classification behavior in *feature space* of (a) standard SVM with the margin for the separating hyperplane and the misclassification measure ζ_i and (b) LS-SVM with two parallel hyperplanes and the error e_i attached to each point.

and the result highly depends on the selected training samples. Hence, the SVM must be trained and tested several times with randomly chosen samples from the database to ensure an accurate average result for the selected features. Using this method instead of class separability measures (CSM) leads to better results while boosting the training time by two orders of magnitude.

D. SVM—Feature Based Decision Modeling

The decision modeling process is provided as a generic software framework which allows online data fusion on a distributed embedded system with limited memory resources. In our multilevel data fusion framework *support vector machines (SVM)* [20] are used as classification method for feature based decision modeling. For $N > 4000$, where N is the number of training data sets, common SVM learning strategies are often not feasible, especially on our embedded platform (cp. Table II, available memory) because of their memory usage and required time. The memory requirement for common SVM learning strategies can be characterized by the standard measure *memory complexity*, which is given by $\mathcal{O}(N^2)$, caused by the storage of a Hessian matrix which is needed in the optimization process involved in training this kernel models.[1] Hence, a modified version of the original SVM, the so called Least Squares Support Vector Machine (LS-SVM) [21], [22] is used for decision modeling in the *I-SENSE* framework. The main characteristic of LS-SVMs is the lower computational complexity compared with original SVMs, without any quality loss in the classification results. A very attractive feature of SVM, namely the sparseness was lost by the LS-SVM formulation. In standard SVM a lot of the Lagrange multipliers are zero, leading to a smaller subset of learning data in order to build the decision boundary between the two involved classes. In LS-SVM almost all multipliers are non-zero, indicating that all training data sets will be used as support vectors. This fact implies slower classification computation and a higher demand of expensive non-volatile memory to store the support vectors. We present in the subsequent sections a method for a intelligent preselection of learning data, in order to reduce the training set and therefore reduce the number of support vectors which will be used by the LS-SVM classifier.

However, a training data set is given by $\{(x_i, y_i)\}_{i=1}^{N}$ with the inputs $x_i \in \mathbb{R}^d$ and class labels $y_i \in \{1, -1\}$. The idea of SVM classifier is to find the linear separating hyper surface $\omega^T\varphi(x)+b = 0$ in the feature space F that separates the mapped data $\{(\varphi(x_1), y_1), \ldots, (\varphi(x_N), y_N)\}$. According to statistical learning theory [20], [23] a good generalization is given if one demands that both classes are separated with a certain margin. The goal is to find the appropriate weight vector ω and the scalar bias term b, such that the relations hold $\forall i = \{1, \ldots, N\}$:

$$\begin{cases} \omega^T\varphi(x_i) + b \geq +1 & \text{if } y_i = +1 \\ \omega^T\varphi(x_i) + b \leq -1 & \text{if } y_i = -1. \end{cases} \tag{1}$$

Instead of building a single hyperplane as in standard SVM [cp. Fig. 3(a)], LS-SVM builds two parallel hyperplanes; one for the positive class and one for the negative class as it is indicated in Fig. 3(b). The distance between these hyperplanes $\omega^T\varphi(x) + b = +1$ and $\omega^T\varphi(x) + b = -1$ in the feature space is called the *separating margin*. Finding the separating hyperplane deals with the problem that this margin has to be maximized. Using Vapniks formalism [20], [23] from standard SVM, this will lead to a constraint quadratic programming (QP) problem. In order to avoid this sometimes hard to solve optimization problem LS-SVM uses equality constraints instead of inequality constraints to find the decision hyperplanes. The difference is compensated by adding an extra term to the cost function that penalizes the deviations from the two hyperplanes for each point of the learning data. The deviations are given by the

[1]For example, a simple calculation reveals that a training data set of 4100 samples—as used in one of our case studies—exceeds the memory capacity on our platform. For 4100 data points, the required memory is $(4.1 \cdot 10^3)^2 \cdot 8\ \text{Byte(for double precision)} = 134.5\ \text{M Byte}$, which exceeds the physical limits of the *I-SENSE* platform.

scalar error $e_i, \forall i \in \{1, \ldots, N\}$. The training problem is given by:

$$\begin{cases} \min_{\omega,e,b} \quad \frac{1}{2}||\omega||_2^2 + \frac{\gamma}{2}||e||_2^2 \\ \text{s.t} \quad y_i \left(\omega^T \varphi(x_i) + b\right) = 1 - e_i \quad \forall i \in \{1, \ldots, N\} \end{cases} \tag{2}$$

where γ plays the role of a regularization parameter between the two quadratic terms in the *primal problem formulation* [see (2)], and characterizes the relative importance of the terms. The first term aims to maximize the distance between the two hyperplanes, while the second term aims to minimize the slack variable e_i. This addition of the two quadratic terms is also responsible for the name least squares SVM. Since the dimension of the feature space is high, possibly infinite, this problem is difficult to solve. Constructing the Lagrangian and using the Karush–Kuhn–Tucker conditions yields in a *dual optimization problem*. This formulation is advantageous because the dimensionality of the optimization problem is equal to the number of data points, indicating that the training process is dependent neither on the dimension of the *feature space* nor on the dimension of the *input space*. Furthermore, these optimization problems are convex. After finding the optimal parameters the classifier is given in the form

$$f(x) = sign\left(\sum_{i=1}^{N} \alpha_i y_i k(x, x_i) + b\right). \tag{3}$$

Note that in LS-SVM all Lagrangian multipliers are non-zero, because all training data sets are used as support vectors for identifying the class separation surface. This main disadvantage is compensated in our approach as described in the following section.

1) LS-SVM Approach for Embedded Systems: In this section we describe a method for selecting vectors out of the training dataset which are likely to be support vectors in a LS-SVM and, therefore, describe best the individual classes. This is done by a data preselection algorithm based on a modified nearest neighbor technique, leading to a smaller set of samples which have to be stored for the classification task.

After this preselection, the remaining datasets are used as support vectors for a LS-SVM classifier to find the decision boundary between two classes in the learning process. Using our approach leads to a sparse LS-SVM classifier with good classification results and lower computational and memory requirements than standard SVM. In embedded systems, the memory resources are quite restricted and, therefore, the proposed approach is advantageous in comparison to a standard SVM.

The training data preselection algorithm (PTD) consists of three main stages as described in the following: A given training dataset T is given by the training samples s. The training samples can be divided into two subsets $A \in \{a_0, \ldots, a_n\}$ and $B \in \{b_0, \ldots, b_n\}$ characterizing the two involved classes.

- **Stage 1**: The goal of this stage is to find the samples nearest to the decision boundary which are needed to classify all samples in T. Therefore, for each sample out of A, the nearest neighbor sample from B and vice versa is identified by computing the Euclidean distance for all sample combinations until this distance is a minimum. These distance tuples are sorted and verified with the nearest neighbor rule in order to check if they are required to classify all samples in T. The resulting subset is called Ω_{nn}.
- **Stage 2**: The reduced nearest neighbor rule [24] is used to obtain the reduced subset Ω_{rnn} from Ω_{nn}. Therefore, each sample of Ω_{nn} is deleted and the nearest neighbor is used again to check if all samples are classified correctly in Ω_{nn} without the deleted ones. If all patterns are classified correctly the next pattern is removed. Otherwise the previous deletion is undone. The remaining subset Π is built by removing those samples from T.
- **Stage 3**: At the last stage the final preselection subset T_{ps} is computed to obtain the k-nearest samples from Π which are closest to the reduced subset Ω_{rnn}.

The value k influences the size of the required training data and the quality of the classification result. It can be set by the user, but our experiments have shown that 3% from the number of samples in T is a good initial value for k. The resulting subset of the training data, namely T_{ps}, is provided to the LS-SVM classifier. Fig. 4 shows an example random Gaussian distribution and the reduced subset Ω_{rnn} after Stage 2 of the proposed algorithm. Fig. 5 shows the overall results of the training data preselection algorithm with different values of k.

2) Performance of the LS-SVM With Preselection of Training Data: To demonstrate the performance of our PTD LS-SVM approach we present the obtained results of two different experiments. First, we generate two random Gaussian distributions, one for each individual class. The distributions show a tendency to a higher level of overlap as indicated in Fig. 4(a). In this experiment we compare the necessary training time, classification result and number of necessary support vectors of four different SVM based approaches under a constant amount of training samples: i) standard SVM, ii) least squares SVM, iii) LS-SVM with training data preselection (PTD-SVM), and iv) a sparse LS-SVM, called $\mathrm{LS}^2 - \mathrm{SVM}$ [25]. Second, we use again two random Gaussian distributions with increasing amount of data points for each class. We evaluate three different implementations in order to obtain a training time result. For all experiments we used a radial basis function (RBF) as kernel function. The results presented in the following are average values of a 20 times repeated experiment with random selection of training datasets.

Table I indicates that the standard SVM has the best training accuracy. Our proposed approach (PTD LS-SVM) deviates only about 3.1% from the standard SVM, while being 45% faster in training than the standard approach. The results presented in this table also show that the training accuracy is quite similar for all training approaches. The fastest training strategy is the LS-SVM approach, followed by the $\mathrm{LS}^2 - \mathrm{SVM}$ approach. Our proposed training data preselection LS-SVM approach is about 3 times slower than the fastest approach. The last column in this table shows that our algorithm has detected a smaller amount of support vectors (SV) even in comparison with the standard SVM approach. In comparison to the LS-SVM approach, which considers all training datasets as support vectors, our algorithm

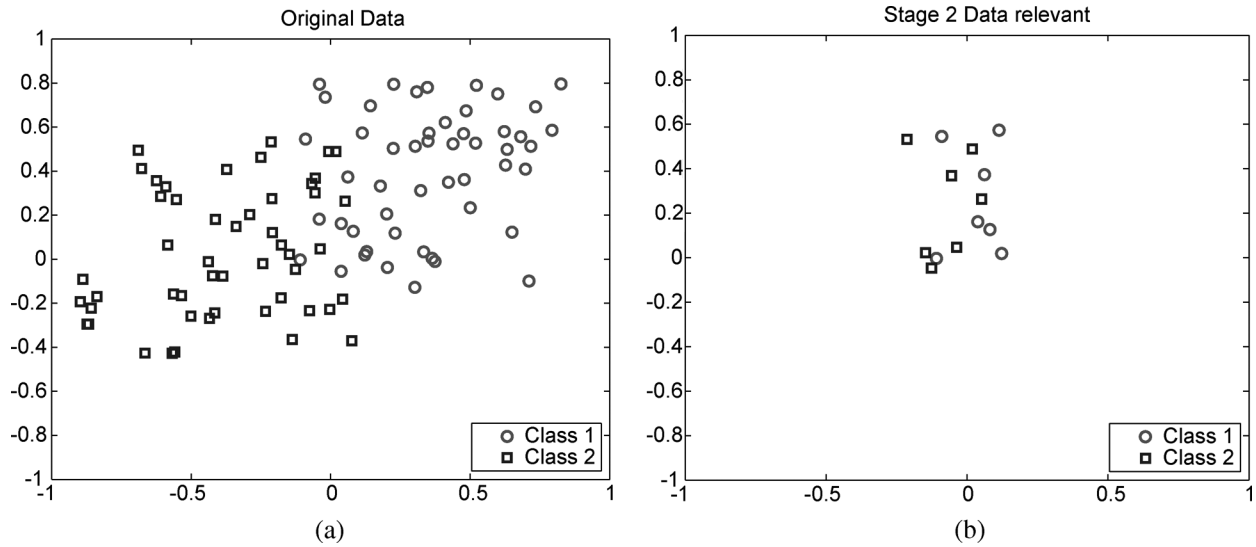

Fig. 4. Effect of our data preselection algorithm demonstrated on normalized, 2-dimensional random Gaussian distribution: (a) inital data set (100 samples) and (b) reduced remaining subset Ω_{rnn}.

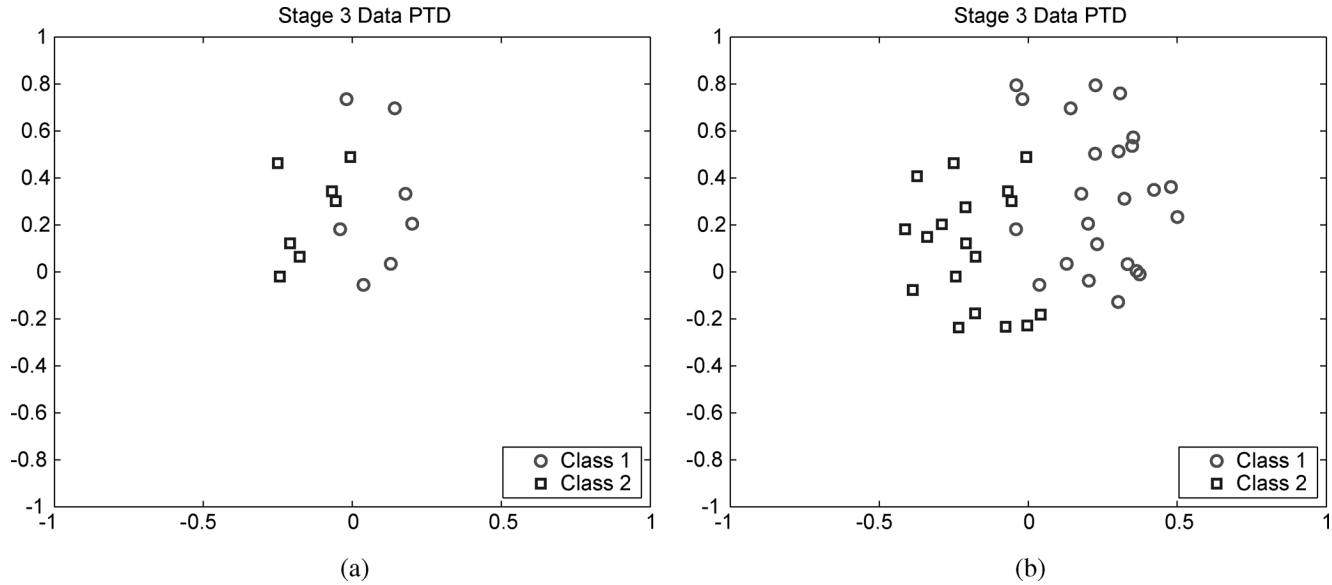

Fig. 5. Results of the training data preselection algorithm for the distribution shown in Fig. 4(a) with (a) $k = 1$ and (b) $k = 3$.

TABLE I
PERFORMANCE OF DIFFERENT SVM-BASED CLASSIFIERS WITH RESPECT TO TRAINING TIME, CLASSIFICATION QUALITY, NECESSARY SUPPORT VECTORS AND REQUIRED MEMORY DURING THE LEARNING PHASE

Algorithm	Training time (s)	Wrong classified (%)	Nb of SV	SV (%)	Memory
SVM	17.81 ± 1.25	12.4 ± 0.5	50.3 ± 3.6	62.9 ± 4.5	$612kB$
PTD LS-SVM	10.04 ± 1.11	15.3 ± 0.8	36.6 ± 4.4	45.8 ± 5.5	$141kB$
LS-SVM	3.85 ± 0.47	14.2 ± 0.5	80.0 ± 0.0	100.0 ± 0.0	$209kB$
LS^2-SVM	4.13 ± 0.64	12.8 ± 0.6	52.8 ± 4.2	66.0 ± 5.3	$237kB$

needed only 45% of support vectors for a quite similar classification result. According to Table I our PTD LS-SVM approach requires only 23% memory compared to the standard SVM during the learning process.

Fig. 6 shows the required training time as an averaged value of 10 experiments. The standard SVM approach is the slowest and is therefore not advisable for large training sets. The fastest approach is the LS-SVM approach, followed by our proposed approach. Both algorithms might be used for large training data sets, however our PTD-SVM approach requires much less support vectors (cp. Table I) which makes it superior for embedded systems with memory restrictions.

E. DS Combination—Decision Based Decision Modeling

This section deals with the fusion of decisions from individual sensors based on Dempster-Shafer (DS) theory of evidence [18].

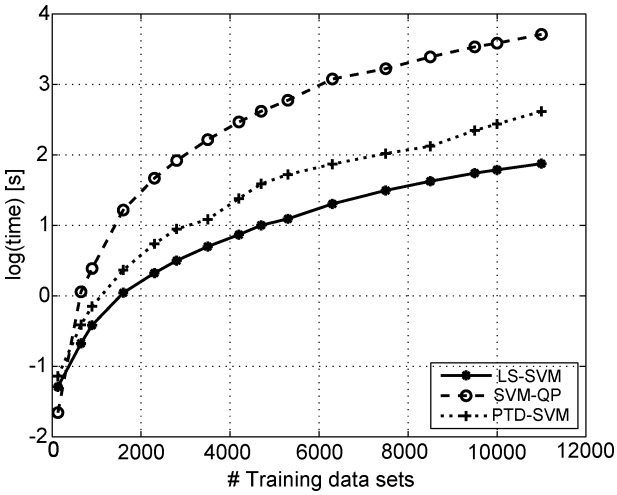

Fig. 6. Comparison of three different learning strategies with regard to required training time versus number of training samples: SVM-QP approach (standard SVM), LS-SVM approach (least squares SVM) and PTD-SVM approach (preselection of training data LS-SVM).

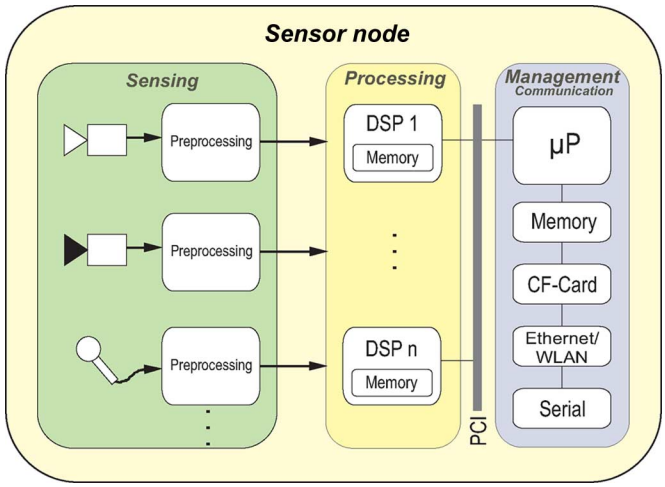

Fig. 7. Internal structure of a sensor node. Such nodes are connected via Ethernet among each other to form a distributed data fusion application.

The main objective is to use the complementary information from different single-source classifiers to fuse these classification results into a single decision or more precisely into a matrix of uncertainty intervals for each possible proposition—the so called "frame of discernment Θ." Here we use a distance mass function of our SVM based classifier as our DS belief function.

The common DS rule of combination implies that we trust all sensors equally. This approach can cause problems if the DS fusion system is not properly designed and is, therefore, suitable only for situations where all sensors have the same accuracy estimates or in situations where the basic belief assignments over the frame of discernment Θ can reflect the ignorance going with the observations. Due to building a generalizable sensor fusion framework working with sensors of different accuracy we introduce a weighted combination rule [4]. The basic idea is to exploit the knowledge of the sensor from similar situations, i.e., we can use the historical performance rates to decide how much we trust in a sensor's actual estimation. By using this approach we modify the original DS combination rule to handle cases of sensors with unequal confidence.

IV. I-SENSE Middleware

The main goal of the I-SENSE middleware is to provide services for i) mapping and executing a fusion application on a distributed embedded system, ii) optimizing the allocation of fusion tasks onto processing elements, and iii) modifying the task allocation during runtime without loosing sensor data during the reconfiguration process. Much care has been taken on an open and portable design of this middleware. As a result, the I-SENSE middleware can be executed on various distributed embedded platforms with sufficient computation and communication power [26].

Fig. 7 presents the internal structure of our sensor nodes. For our experiments this node consists of an "ePCI 101" embedded computing platform equipped with an Intel Pentium M processor running at 1.6 GHz. This platform can be extended by several DSP boards equipped with TMS320C6000 DSPs from Texas Instruments. The DSP boards are connected with the embedded platform via PCI.

As operating system Windows XP Embedded has been chosen, simply because of the premium driver support and its plug and play features. However, the core functionality of the middleware is independent of the operating system and, due to the layered concept, it should be easy to port it to any other operating system or hardware.

The sensor nodes used for the I-SENSE system do not require user interactions, display capabilities or a permanent storage device. However, there has to be one extraordinary node, the so called *master node* where the user of the system can specify and change the functionality of the entire network. The master node is in charge of finding and loading a configuration onto the network, doing reconfigurations during runtime and handling problems and exceptions in the system. A repository of all *fusion tasks* must be installed on this node.

Note that the data fusion application runs on the distributed embedded fusion nodes. Only the configuration is handled by the centralized master node.

A. Services of the Middleware

During initialization, the middleware first scans the PCI bus for supported DSP boards and installs the DSP-based part of the middleware on them. After all instances have been initialized, the node is ready to accept commands and execute fusion tasks.

In Fig. 8 the internal structures of the I-SENSE middleware is illustrated. The *message router* is responsible for a correct and efficient data transfer from one fusion task to another, either on the same processor via *shared memory*, the same node via PCI or on a distant node via Ethernet. Furthermore, the message router supports message forwarding if a fusion task has been migrated to another processor.

Each processor in the I-SENSE network offers a service called *task loader*. It accepts requests to load, start, stop, migrate and remove fusion tasks. Loading a task involves basically the following steps: First, the fusion controller transfers the image of the fusion task, if it is not yet available at the node. After that the task environment is transferred and installed. In

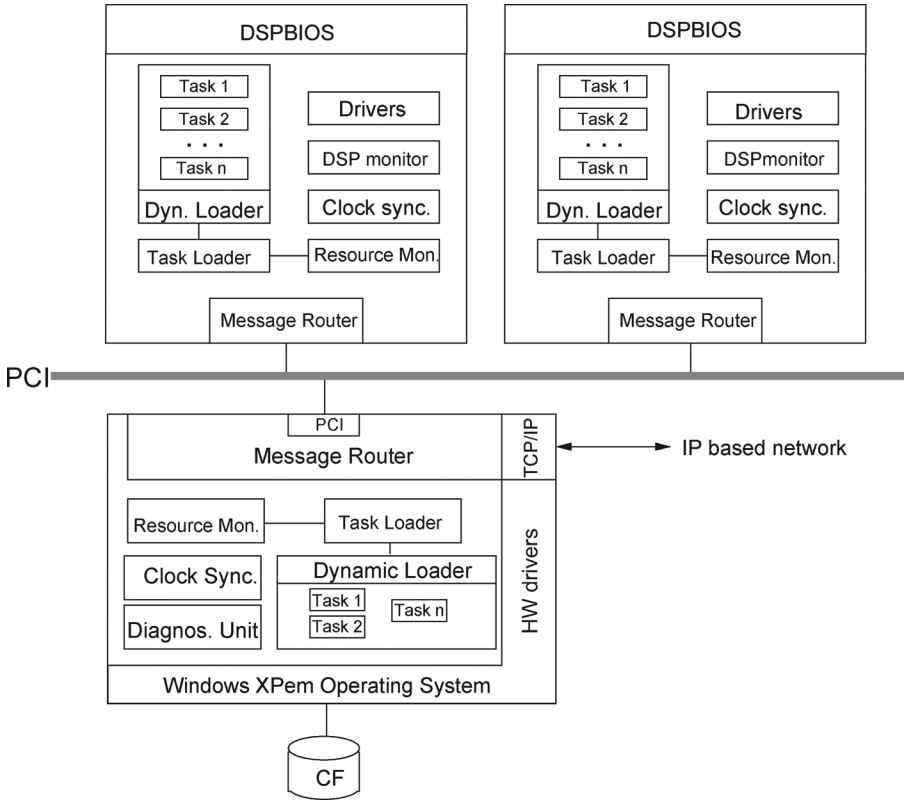

Fig. 8. Internal structure and main services of the I-SENSE middleware.

TABLE II
FOOTPRINT SIZE OF MIDDLEWARE AND AVAILABLE MEMORY ON (a) DSP (TMS320DM642) AND (b) PENTIUM-M

	On-Chip	Off-Chip
Code	20 kB	24kB
Data	36 kB	4 MB
Available	96 kB	124 MB

(a)

	Memory
Code	500 kB
Data	36 kB + 4MB
Available	268 MB

(b)

the next step the communication links are established and registered. If all previous steps have been completed successfully, the task's main routine is started in an own thread.

It is the responsibility of the *resource monitor* to keep record of all consumed resources by the tasks (memory blocks, DMA channels, etc.). This is required for freeing the resources after a fusion tasks has been removed.

Distributed sensor data fusion implies a uniform timebase for all nodes. Without a system wide synchronized clock, it would be impossible to combine results from different sensors. Therefore, each processor has its own task which keeps the local clock synchronized with the system time.

To detect software- and hardware-failures, each node periodically checks its state, and the connection to its neighbor nodes. This functionality is summarized in the *DSP Monitor* and *diagnosis unit* blocks, respectively.

In Table II the memory requirements of the middleware are shown. On-chip memory refers to the fast internal memory of the signal processor while Off-Chip memory refers to the external connected DRAM memory.

The buffer sizes for the communication are adjustable; we currently use a total of 4 MB for these buffers. The communication buffers are dynamically organized and are required for inter-task communication as well as for storing the sensor data during the reconfiguration process. Thus, the size of the buffers influences the maximum time required for the reconfiguration (such that no data is lost). When porting to other platforms, these buffers must be adapted to the actual requirements and can be dimensioned much smaller.

B. Performance of the Middleware

To test the performance of the middleware and estimate the time that it takes to configure and reconfigure the system, we constructed some simple software models consisting of at most four tasks. Table III presents the time required to load specific tasks. In this experiment all code and data have been completely loaded over the network; no local caching has been performed.

In a second test, we measured the time required to migrate a task from one processor to another. The results of this experiment can be seen in Table IV. In principal two different scenarios have to be considered. Either the task migrates from one sensor node to another via Ethernet (*remote destination*) or the task just moves between processors on the same node via PCI (*local destination*).

C. Definition of Fusion Tasks

There are two parts required to describe a fusion task: The first part provides the functionality of the task which is a dynamic loadable library written in C/C++ and has access to the I-SENSE API. The second part provides meta-information

TABLE III
TIME REQUIRED TO LOAD A TASK ON THE SYSTEM WITHOUT CACHING

Task Name	CPU Type	Code Size	Environment	Time
Image viewer	Pentium	154 kB	256 Byte	139.7 ms
Motion detector	Pentium	162 kB	512 kB	273.6 ms
Camera driver	DSP	5 kB	256 Byte	23.3 ms
Motion detector	DSP	7 kB	512 kB	143.6 ms

TABLE IV
TIME REQUIRED TO MIGRATE A TASK BETWEEN PROCESSORS

Task Name	Source	Code Size	Environment	Destination CPU	Time
Image viewer	Pentium	154 kB	256 Byte	remote Pentium	508 ms
Camera driver	DSP	5 kB	256 Byte	local DSP	436 ms
Camera driver	DSP	5 kB	256 Byte	remote DSP	475 ms
Motion detector	Pentium	7 kB	512 kB	local DSP	520 ms
Motion detector	Pentium	7 kB	512 kB	remote DSP	552 ms
Motion detector	DSP	162 kB	512 kB	local Pentium	613 ms
Motion detector	DSP	7 kB	512 kB	local DSP	395 ms
Motion detector	DSP	162 kB	512 kB	remote Pentium	623 ms

about the task specified in a separate XML file. This meta-information is required for two reasons: First, the automatic task assignment module needs to know the resources and the runtime of each task to find an optimal mapping of tasks onto CPUs. Second, when this configuration is loaded onto the distributed system, this meta-information is used to initialize the communication buffers and memory segments for every task.

To build an overall fusion application, individual fusion tasks have to be connected. Thus, each fusion task has a number of ports where it can be connected to other tasks, as defined in the *software model*. These communication links are bidirectional. The number of available ports and the number of available message slots as well as the size of the message slots for outgoing and incoming messages have to be declared in the task's metadata.

In addition to a simple message passing system, the I-SENSE API provides other very useful functions to ease the development of distributed fusion applications. Via the *timebase* module, tasks can query the system time—which is synchronized over the entire system—whenever they want. They can fork new threads by using the *scheduler* module. The *memory management* module standardizes and encapsulates the hardware dependent memory management functions of the underlying operating system. A *DMAmanager* provides a variety of functions to ease the programming of DMA transfers on DSPs.

D. Configuration Method

Whenever the *software model* or the *hardware model* is changed, a reconfiguration process is triggered. Both models are the inputs of the so called *optimizer* which tries to find a suitable mapping of the fusion tasks onto the processors. The optimizer distributes the load which has been balanced by a genetic optimization algorithm [27]. Constraints help to enforce the mapping of an individual fusion task onto a dedicated processor. As soon as a valid configuration is found, the *configuration synthesizer* distributes and runs the fusion tasks on the network of distributed embedded platforms.

There are three possible situations that require the master node to trigger a reconfiguration: i) the user modifies the hardware or software model, ii) a fusion task detects a relevant event and decides to adapt the software model to better capture this event, or iii) a hardware failure has been detected.

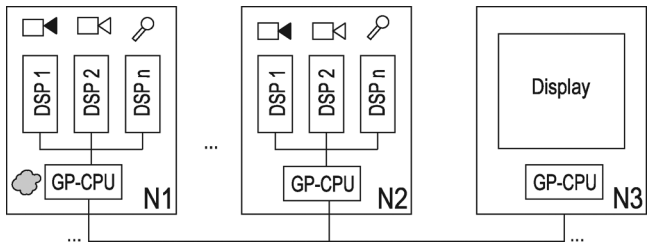

Fig. 9. Hardware topology of an I-SENSE network.

1) Hardware Model: The hardware model describes the distributed embedded system where the fusion application should run on. In our case it consists of a set of connected hardware nodes ($N1 \ldots N3$, cp. Fig. 9). Each hardware node has at least one general purpose CPU (parent) and optionally some digital signal processors (children) coupled via PCI, and various ports to interface sensors.

Every processing node allows to query and use its free resources (i.e., computing power, on/off chip memory, different sensors, etc.) for fusion tasks. A middleware module explores the embedded system automatically. This has two advantages: i) faulty or missing hardware nodes can be found during start up and ii) the hardware model can be built and parameterized during the initialization process.

2) Software Model: The software model describes the functionality of the distributed fusion application. It is obtained from the fusion model (cp. Fig. 2) by increasing the level of detail up to a set of communicating tasks which may be represented as a task graph $G = (N, E)$. It is assumed to be a weighted directed acyclic graph, consisting of nodes $N = (n_1, n_2, \ldots, n_m)$ which represent the fusion tasks and the edges $E = (e_{12}, e_{13}, \ldots, e_{nm})$ which represent the data flow between those tasks.

Each node of the graph has some properties, describing the (hardware/resource-) requirements of a task. The weights of each edge from node u to node v (e_{uv}) indicates the required

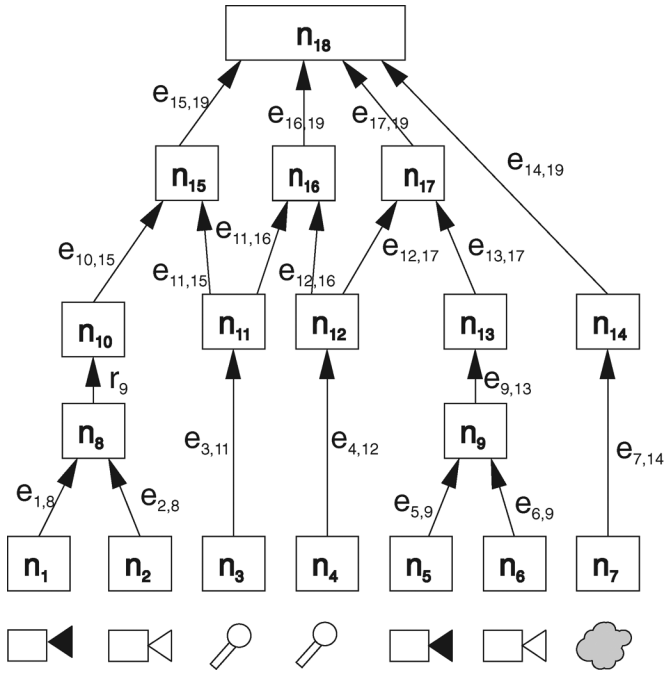

Fig. 10. Simple software model.

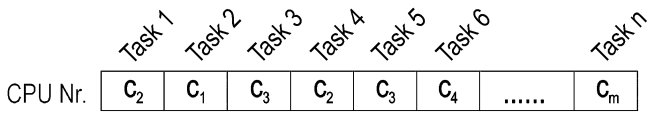

Fig. 11. Mapping of tasks on processors.

communication bandwidth between those two tasks. A quite simple example of a software model is shown in Fig. 10.

E. Genetic Algorithm for Task Allocation

As mentioned before, a genetic algorithm is used to allocate n tasks and their interconnections, represented by the software model $G = (N, E)$, on a distributed embedded system consisting of m heterogeneous processors, described by the hardware model.

1) Encoding of Chromosomes: To solve a problem by genetic algorithms, it is necessary to find a mapping of a potential candidate for a solution onto a sequence of binary digits, the so called chromosome. In our case, however, it is more suitable and efficient to represent chromosomes as strings of integers. The length of the chromosomes is given by the number of tasks n that should be allocated. Every gene in the chromosome represents the processor m where the task is running on. Fig. 11 presents an example mapping of n tasks on m processors.

2) Fitness Function Definition: A problem with a straight forward implementation of a genetic algorithm in our case is that in a randomly generated initial population almost all combinations are invalid. Either the CPU utilization of any processor in the system is exceeded or a communication link is overloaded. As proposed in [28], a penalty in the fitness score for those violations has proved to be a good approach in our work.

The fitness score of a CPU is calculated by a polynomial function of second order. A processor work load of 50% gets the highest possible fitness score. More or less utilization causes the fitness score to decrease. If a processor is overloaded (more than 90% usage) the chromosome gets a punishment value that is higher than the highest possible score of a perfect chromosome. If more than one CPU in the chromosome is overloaded, this punishment value is subtracted for each violation. The fitness of the communication is calculated in a similar way. The main difference is that a communication utilization of 35% results in the best possible fitness score. The settings of a preferred 35% network load and 50% CPU load have been determined empirically.[2] The overall fitness score of a chromosome is simply a weighted sum of the processor- and communication-fitness.

TABLE V
FITNESS FUNCTION EVALUATIONS REQUIRED TO FIND A VALID CONFIGURATION

Tasks	CPUs	Complexity	Iterations
15	6	easy	107.2
15	10	easy	267.2
15	14	medium	428.0
26	10	medium	848.8
26	14	hard	4038.4
20	14	hard	5424.8
20	12	very hard	7068.0

Table V presents the number of fitness function evaluations necessary to find a valid configuration for many different software and hardware models. All presented values have been averaged over 200 runs. The population size was fixed at 80 elements. All examples have been classified by its complexity. Problems classified as 'easy' have a large number of valid solutions while in "hard" problems the fraction of valid combinations is very small. For the example classified as "very hard," a special hardware model has been constructed which utilizes all processors close at their limit.

The presented Table V reflects the assumed cluster size of several dozen computation nodes. Currently a single node is responsible for managing a moderate number of slaves. Adopting a hierarchical network approach, similar to [29] might be a solution for managing reconfigurations for larger networks (several hundred to many thousand nodes).

The quotation of fitness function evaluations is quite popular when dealing with genetic algorithms since it is independent of the actual implementation and used processor. However, it says nothing about the expected execution time. On our development computer (Intel Pentium 4, 1.9 GHz) it takes approximately 0.3 s to find a solution for the problem specified as "very hard."

F. Applying Model Changes at Runtime

A special feature of I-SENSE is the capability to apply small and medium changes in the software model or in the hardware model while the system is running. Therefore, the light-weight I-SENSE middleware supports loading, unloading as well as migration of tasks and the update of communication links during runtime. So it is possible to change an existing configuration gradually into a modified one by applying those four operations. Similar to the famous *fifteen puzzle* where squares are moved

[2]This is a pragmatic approach to keep these system resources available for other applications and to allow the computation and the deployment of new configurations in the background.

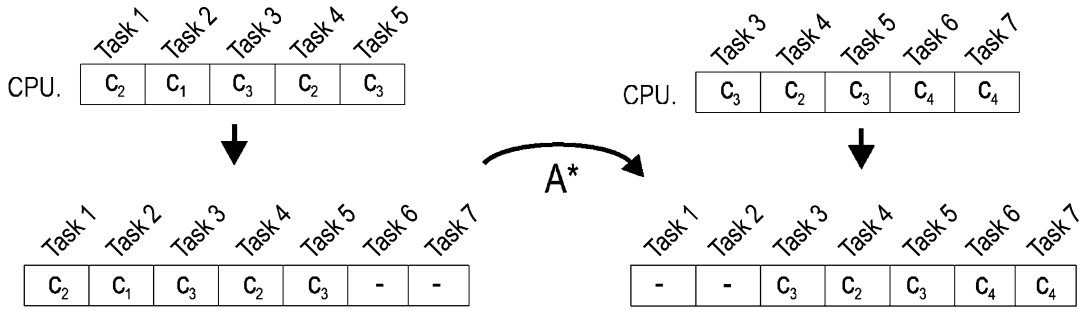

Fig. 12. Example for a transfer in the common solution-space.

around to obtain a specific pattern, we move tasks on the distributed system to transform an existing configuration into the desired modified one.

The number of sequential task migrations directly correlates with the time required for the reconfiguration. To keep this number low, paying attention to the similarity of the new configuration is essential. Therefore, in case of a reconfiguration, the fitness score of a gene is extended by a third weight which expresses the difference between the existing configuration and the potential new one. Thus, similar configurations are preferred by the genetic algorithm and the search for a reconfiguration sequence is eased.

Since a changed configuration may differ in the number of tasks and the number of processing nodes, both configurations must be transferred into a common solution space before a path search algorithm can be invoked (cp. Fig. 12).

Finally, an A* search algorithm [30] tries to find a sequence of valid configurations to transform the current configuration into the new one. The two heuristic functions f and g for the A* algorithm are currently the following:

$$f = \text{number of already moved tasks}$$
$$g = \text{number of tasks still on the wrong CPU}$$

In cases where a valid sequence for an online reconfiguration can not be found in an adequate time, the system reverts to a offline reconfiguration process automatically. This means all fusion tasks may be halted during the reconfiguration process and the data acquisition from the environment is suspended for this time.

V. Case Studies on Vehicle Classification and Bulk Good Separation

In this section we present the case studies which focus on vehicle classification and granulated material separation. Vehicle classification is performed by exploiting visual and acoustic data. In our experiments we collected a database consisting of about 4100 vehicles which are mainly assigned to three different classes: large trucks, small trucks and cars. Further vehicle classes (motorcycles and buses) are possible but the number of samples in these classes are rather low in comparison to the other classes and we, therefore, decided to use only these three classes to demonstrate the feasibility of our multilevel data fusion approach. A screenshot of the I-SENSE user interface is depicted in Fig. 14. Bulk good separation is focused on visual data extended by infrared spectral imaging data. In the subsequent sections we focus on feature extraction for these case studies.

A. Feature Extraction for Vehicle Classification

For our visual feature extractor we adopted the ideas of Viola and Jones [31] to build a multiclass extractor and improved it by using *RealBoost* [32]. The feature set is built by Haar-features and additional gradient-based information which are calculated in real-time on an embedded platform. The boosting approach is mainly used to extract the most powerful features. Further details are presented in [3].

For our acoustic feature extractor various signal processing algorithms have been implemented in order to collect a pool of candidate features able to distinguish between vehicle categories. Each of the algorithms extracts several features from the raw input data. Features in time domain are generated from short time energies, zero crossing rates and correlation analysis algorithms. Spectral features include signal attributes that describe average energies, positions and spreads in frequency domain, such as the spectral centroid, signal bandwidth, spectral flux, or band energy ratios. Cepstral coefficients are popular feature candidates as they provide very good information packing properties: Low order coefficients capture information about the slowly varying properties of the spectrum, also referred to as spectral envelope. A more comprehensive overview is presented in [33].

B. Audio–Visual Vehicle Classification Results

We have implemented a simple multiclass classifier by applying the One-against-All technique. 30% of the vehicle database is selected for training and the other 70% is used for evaluation purposes. The results presented in the following are average values from 20 runs of our LS-SVM with support vector preselection and a radial basis function (RBF) as the kernel function.

First, we demonstrate the vehicle classification results based on a single sensor estimation only. The box plots shown in Fig. 13(a) indicate that class separation with acoustic features only is quite difficult. Using a confusion matrix (cp. Table VI—audio features only) for this experiment reveals that the classification system achieves a quite reliable distinction between cars and other vehicles but has serious problems in distinguishing between the two types of trucks.

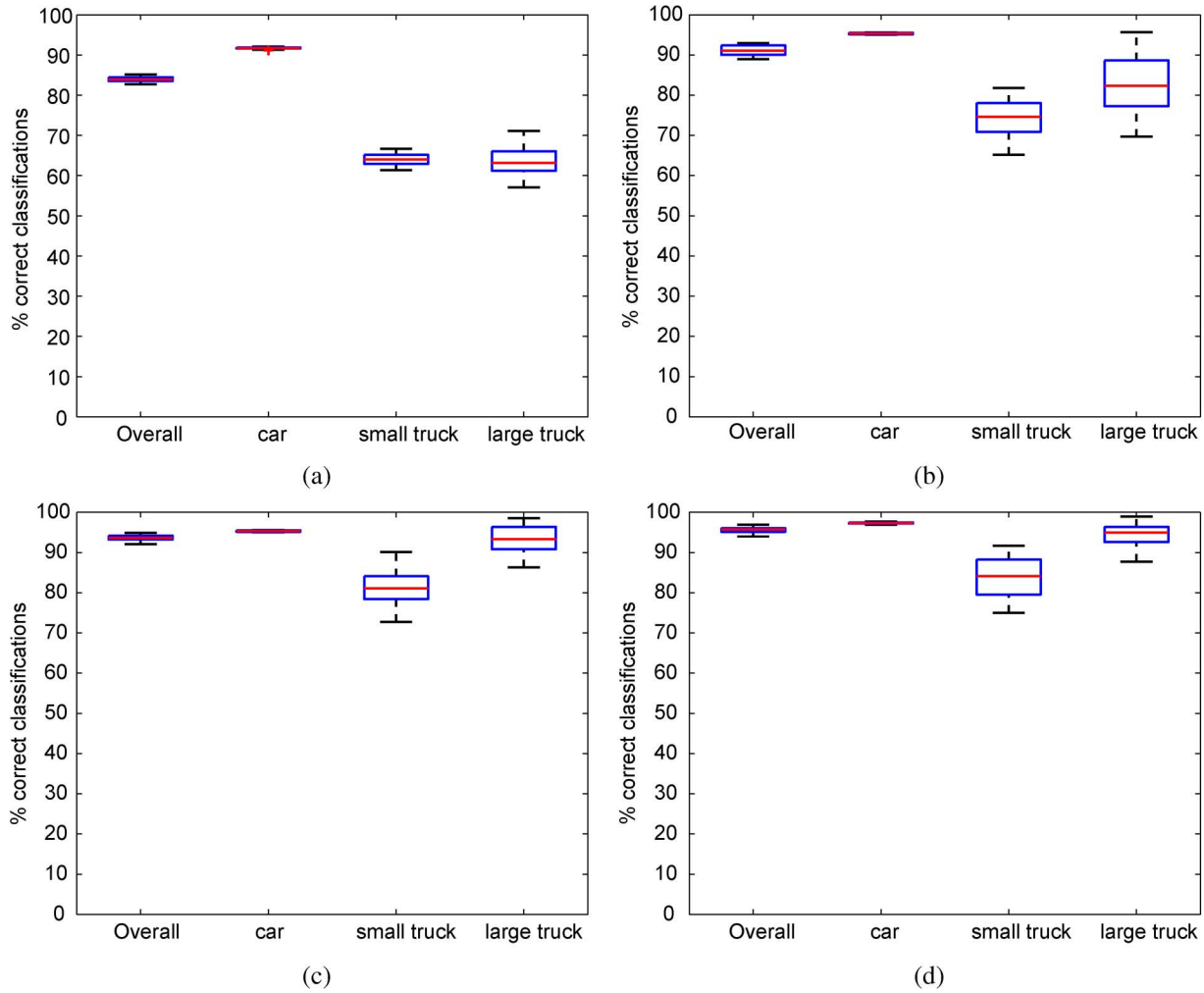

Fig. 13. Classification result with PTD LS-SVM based on (a) acoustic features only, (b) visual features only, (c) DS fused decisions from individual sensors and (d) fused features from both sensors and accurate feature selection. The lines indicate the lower, median and upper quartile values; whiskers show the extent of the rest of the data.

Fig. 13(b) demonstrates quite the same classification performance for vision-only sensor data as for a acoustic-only classification. Higher absolute classification rates are the main difference between the two single sensor classifiers.

In Fig. 13(c) and Table VI (cp. decision level fusion), we show that our approach for fusing data at the decision level is advantageous in comparison to single sensor classification in the overall classification result as well as the individual class separation abilities. In our case study we use a weight $w_1 = w_2$ in order to trust in both sensors equally.

According to Fig. 13(d) and Table VI (cp. feature level fusion), fusing data at feature level is superior to the Dempster–Shafer approach, discussed in Section III-E. Note, that classification based on single sensor decisions needs less memory and communication requirements than using feature based classification. Therefore, both approaches are suitable in an multilevel sensor fusion framework—depending on the current situation and the available computational and memory resources.

While Fig. 13 visualizes the overall performance of our PTD LS-SVM classifier, the confusion matrix (cp. Table VI) shows detailed information about the actual and predicted classifications of our presented classifier. A comparison between the vision-only classification [cp. Fig. 13(a)] and the classification based on fused features from audio and visual sensors [cp. Figs. 13(c) and 13(d)] can be interpreted as follows: The vision-only classifier predicts cars very well (95.3%), but it tends to have problems in distinguishing between small trucks (74.6%) and large trucks (82.3%). Quite similar behavior is achieved by using acoustic features only (91.7%, 64.0%, and 63.2%, respectively). However, fusing data from both sensors either at the feature level or the decision level leads to good classification performance for all three vehicle classes (97.3%, 84.1%, and 95.0%, respectively) while decreasing the false-positive rates (by a factor of 3.3 for cars, by a factor of 2.1 for small trucks and by a factor of 1.6 for large trucks).

C. *Feature Extraction for Bulk Good Separation*

This section presents the feature extraction for the bulk good separation case study (cp. Fig. 15) which focuses on granulated material classification (e.g., rocks, minerals, glasses, etc.).

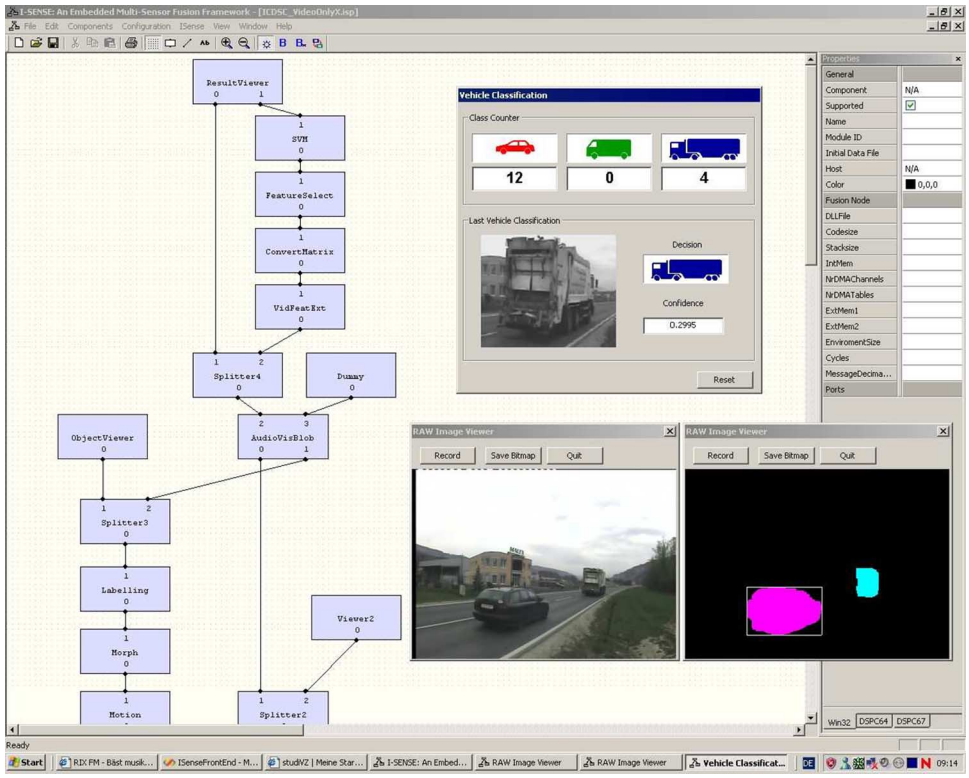

Fig. 14. Screenshot of our online vehicle classification build on the I-SENSE middleware.

TABLE VI
CONFUSION MATRICES OF THE VEHICLE CLASSIFICATION BASED ON I) ACOUSTIC FEATURES, II) VISUAL FEATURES, III) DECISION FUSION, AND IV) FEATURE FUSION

	audio features only			video features only		
	car	s. truck	l. truck	car	s. truck	l. truck
car	1377	82	39	1426	48	24
s. truck	13	117	55	25	140	20
l. truck	25	99	264	3	26	359
	decision level fusion			feature level fusion		
	car	s. truck	l. truck	car	s. truck	l. truck
car	1431	42	25	1447	33	18
s. truck	11	153	21	5	170	10
l. truck	1	9	378	2	2	384

D. Visual Feature Extraction

Fundamental properties of granulated materials are size, shape, texture and physical composition. Color properties can provide useful information about the composition of materials. Surface texture gives clues to its crystal content. These properties are characterized from images of granulated material, summarized in the following paragraphs.

a) Color features: The most common colors of granulated material used in this case study are red, brown and yellow which are typically due to the presence of ferric oxide cement, gray–black which reflects the presence of carbonaceous material, and colorless, such as quartz, which contains neither ferric oxide nor free carbon. Color is an especially important characteristic for shale. The implemented algorithm in the *I-SENSE* framework is to normalize each intensity image (taken from granulated materials on a conveyor belt) to zero mean and unit variance for pixel intensities to help to account for changes in camera parameters when the image was taken. Scenes with highly directional lighting cause highlights and shadows, the computed intensity variance for a given material is larger than the true value, compensated by an adaptive histogram equalization.

The simplest method of extracting suitable features is characterizing granulated material albedo or color which involves two statistical measures: mean and variance. The mean pixel intensity represents the reflectivity of the material, while the variance in intensity provides a measure of how uniform the material reflectivity is. In the *I-SENSE* feature extraction unit two statistical features are obtained based on mean and variance of the objects intensity.

The second method is focused on a histogram of intensities, a method which gives a more complete representation of the reflectivity of a material. Often a granulated material will consist of regions of different reflectivity, which intensity mean and variance alone cannot accurately characterize. Pixel intensity ranges from zero to one. In the *I-SENSE* framework this range is divided into eight bins to compute the histogram and then normalize so that the elements of the resulting vector sum to one. The values of the eight bins serve as eight numerical features in the *I-SENSE* feature extractor.

The third method is focused on the color representation of a material, a method which involves determining the reflectivity at all wavelengths. However, the *I-SENSE* project is dealing with images of granulated materials which provide the intensity for

(a) (b)

Fig. 15. Example materials for granulated material separation: (a) precious rubble (referred to class 1) and (b) worthless rubble (referred to class 2).

each pixel at three different wavelengths. Pixel color is commonly represented in Hue Saturation Value (HSV) [34]. However, neither the RGB or HSV color space is uniform in that the numerical distance between colors does not correspond to the distance perceived by humans. However, to characterize granulated material, the mean and variance over each color channel are computed, as was previously done for intensity. For a more complete color representation, a color histogram is used. The previously explained method of intensity histograms can be applied to each color channel, resulting in a 2-D histogram and 24 color features.

b) Texture features: The surface texture is the size, shape and arrangement of the component elements of granulated materials as well as surface markings such as polish, striations and pits. Properties of crystals within the material, such as grain-size, distribution, sorting, permeability, shape and orientation are also important characteristics of materials identity. In the *I-SENSE* framework two methods for obtaining suitable features based on object texture analysis are provided.

First, features from co-occurrence statistics are obtained. The *gray-level co-occurrence matrix* (GLCM) measures spatial relationships of pixels in an object image. The matrix is defined as follows:

$$GLCM_{d,\alpha}(i,j) = |\{((r,s),(t,v)) : I(r,s) = i, I(t,v) = j\}| \tag{4}$$

where d is the distance at an angle α between pixels of intensities i and j and $|.|$ is the cardinality of a set. In other words, entry (i,j) is the number of occurrences of the pair of gray levels i and j at a distance d and angle α apart. This is computed for $\alpha = 0°$, $45°$, $90°$, $135°$ and d =1 to 5, averaged over these values for pixel intensities divided into eight bins. From this, the following features are computed:

$$Contrast = \sum_{(i,j)} |i-j|^2 GLCM(i,j) \tag{5}$$

$$Correlation = \sum_{(i,j)} \frac{(i-\mu_i)(j-\mu_j)GLCM(i,j)}{\sigma_i \sigma_j} \tag{6}$$

$$Energy = \sum_{(i,j)} GLCM(i,j)^2 \tag{7}$$

$$Homogeneity = \sum_{(i,j)} \frac{GLCM(i,j)}{1+|i-j|}. \tag{8}$$

These features are computed in the *I-SENSE* framework and form a vector to represent the texture.

The second method for feature extraction based on texture involve the most common approach to texture analysis in computer vision, a method convolving the texture with a set of filters and clustering the responses to form textons. In this work the Maximum Response 8 (MR8) filter bank is used. The MR8 filter bank, originally introduced by Varma and Zisserman in [35], is based on an edge filter (first derivative of a Gaussian) and a bar filter (second derivative of a Gaussian) at six orientations and three scales, and two spot filters (a Gaussian and a Laplacian of a Gaussian). The unique trick with this approach is that for each filter, the maximum response is taken across the six orientations. This reduces the response vector down to eight dimensions. Each response vector is normalized according to

$$R_i \leftarrow R_i \frac{log\,(1+||R||/0.03)}{||R||} \tag{9}$$

where R is the vector and R_i is each element. Once the filter bank is convolved with the texture to form a response vector for each pixel using MR8 filter bank, a set of textons is computed. The response vectors from all textures in the set are aggregated into a matrix of size $M \times N$ where M is the total number of pixels in the images and N is the dimensionality of the response vector. In the *I-SENSE* approach the response vectors are clustered using k-means [36].[3] Each texton is displayed as the linear combination of the filters and represents a form of primitive structure in the set of textured object images. From this the closest texton for the response vector is computed at each image pixel, forming a texton map. This distribution of textons within an granulated material is represented with a histogram, counting the number of occurrences of each texton in the image. Texture features are extracted based on these representative histograms.

1) Spectral Imaging Feature Extraction: Features based on spectral imaging can provide useful information about the composition of granulated materials. The overall aim is to extract features with sufficient class discriminatory abilities

[3]In this work 32 clusters are used, with each cluster representing a texton.

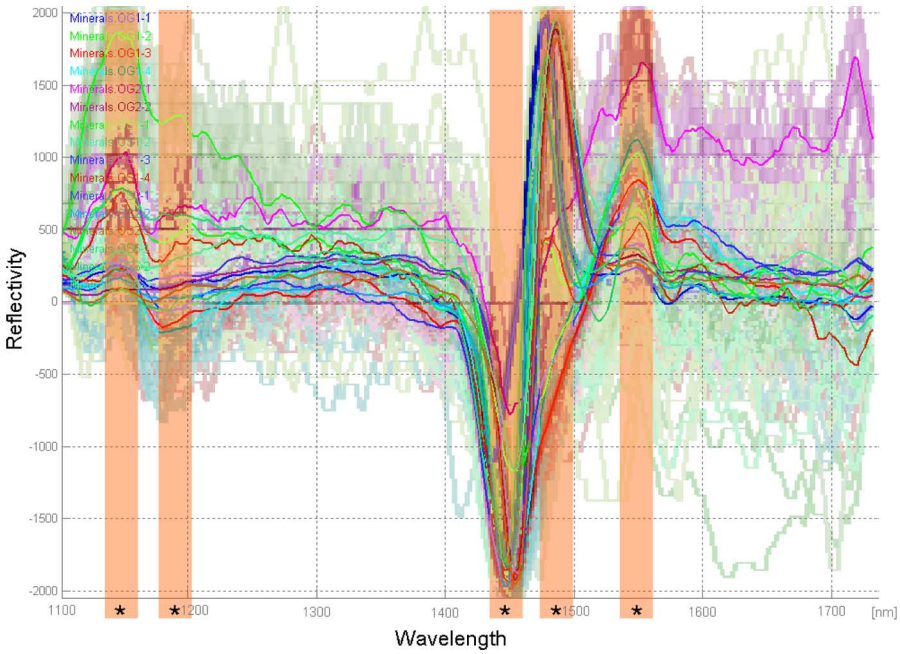

Fig. 16. Infrared spectra (range: 1100 nm–1700 nm) of different minerals with defined regions of interest (indicated by "*"), where statistical features are extracted.

from infrared spectra of the objects of interests. Therefore, in the *I-SENSE* framework various materials were analyzed heuristically for the specific application and regions of interest (ROI) were extracted during the evaluation stage (cp. Fig. 16). The method of extracting suitable features is based on two statistical measures (mean and variance) for each obtained pixel, in order to characterize the ROI of granulated material's spectra. Therefore, measures as min-value, max-value and mean-value are obtained from objectives spectra. Furthermore, the first derivative is computed, where the slope is an additional feasible feature. These features are computed in the *I-SENSE* framework and form a vector to represent the information obtained from granulated materials infrared spectra.

E. Granulated Materials Classification Results

Bulk good separation or more precise granulated materials classification is an important task in industrial applications. To keep this demonstration application as simple as possible several autonomous experiments were conducted. For simplicity reason the segmentation task is not further considered in this section. Only the quantitative results of the conducted experiment are presented in the following. In this case study texture and color features are extracted as described in Section V-C in order to distinguish between two classes of granulated material, e.g., as indicated by Fig. 15.

The results shown in the following tables (cp. Table VII) present the classification behavior for separating granulated material. Therefore, the confusion matrix, which contains information about actual and predicted classifications done by a classification system, is built. From the confusion matrix for each individual involved class, statistical measures as accuracy, true-positive rate and false-positive rate are computed. Whereby, the *accuracy* is the proportion of the total number of predictions that were correct, the *true-positive rate* (TP) is the proportion of positive cases that were correctly identified, the *false-positive rate* (FP) is defined as the proportion of negatives cases that were incorrectly classified as positive. To evaluate the feasibility of *I-SENSE* approach the following experiments were conducted: a) color based feature extraction and feature selection and PTD LS-SVM classifier, b) color and spectral imaging based feature extraction and feature selection and PTD LS-SVM classifier, c) color and spectral imaging based feature extraction and feature selection and PTD LS-SVM classifier and Fusion based on multiple decisions (weights were set to 0.5 for each individual classifier), d) color and spectral imaging based feature extraction and PTD LS-SVM classifier without feature selection, and e) color and spectral imaging based feature extraction and classification based on k-means clustering as described in [37]. The tables given below are obtained from 40 times randomly repeated selection of learning and evaluation examples out of the pool of objects in order to consider the generalization behavior.

The results presented in Table VII show the classification behavior based on single-sensor color information. The overall classification accuracy is about 87%, and therefore quite high, while both involved classes are classified with similar true-positive rates. Adding new types of sensors may have very significant impact in classification capability, because of an additional added dimensionality of sensed data, an fact which is indicated in Table VII(b). Additional features from spectral imaging sensor increase the overall accuracy by approx. 11%, due to the integration of a suitable feature selection stage. Quite similar to these results are the results obtained by multiple single-sensor classifiers and a decision based fusion (i.e., weighted DS combination), as given in Table VII(c). In this case it seems that the final classifier tends to have problems treat both involved classes similarly. Table VII(d), demonstrate the main difficulties in feature based information fusion. Without a suitable feature selection process the classifier is mislead, caused by the high mutual

TABLE VII
GRANULATED MATERIAL SEPARATION RESULTS BASED ON PTD LS-SVM CLASSIFIER AND UNSUPERVISED K-MEANS CLUSTERING

Mean classification accuracy	86,84%	
	mean TP [%]	mean FP [%]
Class 1	88,73	11,27
Class 2	84,95	15,05

(a)

Mean classification accuracy	97,99%	
	mean TP [%]	mean FP [%]
Class 1	97,55	2,45
Class 2	98,43	1,57

(b)

Mean classification accuracy	93,40%	
	mean TP [%]	mean FP [%]
Class 1	92,22	7,78
Class 2	94,58	5,42

(c)

Mean classification accuracy	61,53%	
	mean TP [%]	mean FP [%]
Class 1	68,82	31,18
Class 2	54,23	45,77

(d)

Mean classification accuracy	72,82%	
	mean TP [%]	mean FP [%]
Class 1	72,21	27,79
Class 2	73,43	26,57

(e)

correlation of features from the two involved sensors. Therefore, the overall accuracy (i.e., 61%) decreases dramatically. Using a unsupervised learning strategy, i.e., k-means clustering, is not feasible in the presented application field, a fact which is illustrated by Table VII(e).

Summarizing the results obtained in this granulated material separation case study beside the qualitative benefits of the presented multilevel fusion approach another important interpretation can be done regarding the reusability of the *I-SENSE* framework. The *I-SENSE* framework is a generic fusion model suitable for a broad range of classification applications. The model can be easily adapted for numerous applications by simply exchanging the feature extraction tasks.

VI. CONCLUSION

In this paper, we have presented I-SENSE—our novel multi sensor fusion model focusing on multilevel data fusion on networked embedded systems. The I-SENSE model considers the data flow in the embedded sensor network and features a light-weight middleware with dynamic reconfiguration capabilities. Our framework further provides an enhanced SVM-based classifier which achieves a good compromise between computation speed, memory requirements and classification performance—all of which is important for distributed, embedded fusion applications.

We have demonstrated the I-SENSE framework in two case studies. By fusing visual and acoustic data at different levels of abstraction, we were able to increase the overall accuracy in our vehicle classification case study from 90% of vision-based classification to about 96%. This case study also showed that the discrimination among the classes for small and large trucks can be significantly improved by combining (weak) single sensor classifiers. A further case study demonstrates the reusability of the generic fusion model and confirms the tendencies obtained in the first case study.

Sensor fusion is an important technique to improve the quality and robustness of many applications. Since sensor, computing and communication devices are getting more capable, smaller and cheaper at a very fast pace, fusion will become an enabling technology for many embedded applications. By providing a middleware which considers important parameters for distributed embedded systems, our I-SENSE framework may help to develop embedded fusion applications.

However, there is still a long road ahead to support the development process to a full extend. Thus, our future work will focus on the following issues.

- **Exploit the fusion refinement**. This introduces some adaptivity such that individual units in our fusion model (Fig. 2) can be adapted for example due to changed environmental conditions.
- **Perform distributed sensor fusion**. In the current case studies we do not exploit the spatial and temporal relationship among multiple sensors. However, it is natural to do so to further improve the quality.
- **Integrate different sensors**. We plan to integrate additional sensors for our case study such as lasers for capturing the height profiles or inductive loops. An important question is to determine the tradeoff between increased hardware costs and increased performance.
- **Implement sophisticated error handling**. The entire concept permits error handling on various levels. But up to now there is no intelligent error handling implemented. We are planning to use a policy based approach, similar to [38].

REFERENCES

[1] M. Bramberger, A. Doblander, A. Maier, B. Rinner, and H. Schwabach, "Distributed embedded smart cameras for surveillance applications," *Computer*, vol. 39, no. 2, pp. 68–75, Feb. 2006.

[2] B. Rinner, M. Jovanovic, and M. Quaritsch, "Embedded middleware on distributed smart cameras," in *Proc. IEEE Int. Conf. Acoustics, Speech, and Signal Processing (ICASSP 2007)*, 2007, pp. 1381–1384.

[3] A. Klausner, A. Tengg, C. Leistner, S. Erb, and B. Rinner, "An audio-visual sensor fusion approach for feature based vehicle identification," in *Proc. Int. Conference on Advanced Video and Signal Based Surveillance (AVSS-07)*, London, U.K., Sep. 2007.

[4] A. Klausner, A. Tengg, and B. Rinner, "Vehicle classification on multi-sensor smart cameras using feature- and decision-fusion," in *Proc. First ACM/IEEE Int. Conf. Distributed Smart Cameras (ICDSC-07)*, Vienna, Austria, Sep. 2007, pp. 67–74.

[5] R. Chellappa, G. Qian, and Q. Zheng, "Vehicle detection and tracking using acoustic and video sensors," in *IEEE Int. Conf. Acoustics, Speech, and Signal Processing, 2004 (ICASSP '04)*, 2004, vol. 3, pp. 793–796.

[6] H. Durrant-Whyte and M. Stevens, "Data fusion in decentralised sensing networks," in *Proc. Int. Conf. Information Fusion*, 2001, vol. 3, pp. 19–24.

[7] A. Gad and M. Farooq, "Data fusion architecture for Maritime Surveillance," in *Proc. Fifth Int. Conf. Information Fusion*, Washington, DC, Jul. 2002.

[8] R. Kumar, M. Wolenetz, B. Agarwalla, J. Shin, P. Hutto, A. Paul, and U. Ramachandran, "DFuse: A framework for distributed data fusion," in *Proc. 2003 ACM SensSys*, Los Angeles, CA, Nov. 2003.
[9] J. Llinas and D. L. Hall, "An introduction to multi-sensor data fusion," in *Proc. 1998 IEEE Int. Symp. Circuits and Systems*, May–Jun. 1998, vol. 6, pp. 537–540.
[10] S. C. Thomopoulos, "Sensor integration and data fusion," in *Proc. SPIE, Sensor Fusion II: Human and Machine Strategies,* , P. S. Schenker, Ed., Mar. 1990, vol. 1198, pp. 178–191.
[11] R. C. Luo and M. G. Kay, "Multisensor integration and fusion: Issues and approaches," in *Proc. SPIE, Sensor Fusion*, Mar. 1988, vol. 931, pp. 42–49.
[12] C. J. Harris, A. Bailey, and T. J. Dodd, "Multi-sensor data fusion in defence and aerospace," *Aeronautical J.*, vol. 102, pp. 229–244, 1998.
[13] J. Schoess and G. Castore, "A distributed sensor architecture for advanced aerospace systems," in *Proc. SPIE, Sensor Fusion*, Apr. 1988, vol. 932, pp. 74–86.
[14] B. Dasarathy, "Sensor fusion potential exploitation-innovative architectures and illustrative applications," *Proc. IEEE*, vol. 85, no. 1, 1997.
[15] M. Bedworth and J. O'Brien, "The Omnibus Model: A New Model of Data Fusion?," *IEEE AES Mag.*, pp. 30–36, Apr. 2000.
[16] A. Klausner, A. Tengg, and B. Rinner, "Enhanced least squares support vector machines for decision modeling in a multi-sensor fusion framework," in *Proc. Int. Conf. Artificial Intelligence and Pattern Recognition (AIPR-07)*, Orlando, FL, Jul. 2007, pp. 327–333.
[17] A. Steinberg, C. Bowman, and F. White, "Revisions to the JDL data fusion model," in *Proc, SPIE, Sensor Fusion*, 1999, vol. 3719.
[18] A. P. Dempster, "A generalization of Bayesian inference," *J. Roy. Statist. Soc.*, vol. 30, pp. 205–247, 1968.
[19] M. Mitchell, *An Introduction to Genetic Algorithms*. Cambridge, MA: MIT Press, 1996.
[20] V. Vapnik, *Statistical Learning Theory*. New York, US: Wiley, 1998.
[21] J. Suykens and J. Vandewalle, "Least squares support vector machine classifier," *Neural Process. Lett.*, vol. 9, no. 3, pp. 293–300, June 1999.
[22] J. Suykens, P. V. Dooren, B. D. Moor, and J. Vandewalle, "Least squares support vector machine classifiers: A large scale algorithm," in *Eur. Conf. Circuit Theory and Design (ECCTD'99)*, 1999, pp. 839–842.
[23] V. Vapnik, *The Nature of Statistical Learning Theory*. New York: Springer-Verlag, 1995.
[24] G. Gates, "The reduced nearest neighbour rule," *IEEE Trans. Inform. Theory*, vol. 18, no. 3, pp. 431–433, May 1972.
[25] J. Valyon and G. Horvath, "A sparse least squares support vector machine classifier," in *Proc. Int. Joint Conf. Neural Networks (IJCNN'04)*, Budapest, Hungary, Jul. 2004.
[26] A. Tengg, A. Klausner, and B. Rinner, "I-SENSE: A light-weight middleware for embedded multi-sensor data-fusion," in *Proc. 5th IEEE Int. Workshop on Intelligent Solutions in Embedded Systems (WISES)*, Madrid, Spain, Jun. 2007, pp. 165–177.
[27] A. Tengg, A. Klausner, and B. Rinner, "An improved genetic algorithm for task allocation in distributed embedded systems," in *Proc. Genetic and Evolutionary Computation Conf. (GECCO-2007)*, London, U.K., July 2007, p. 1534.
[28] J. Gottlieb, "On the Feasibility Problem of Penalty-Based Evolutionary Algorithms for Knapsack Problems," in *Applications of Evolutionary Computing: Proceedings of the EvoWorkshops 2001*, E. J. W. Boers, S. Cagnoni, J. Gottlieb, E. Hart, P. L. Lanzi, G. R. Raidl, R. E. Smith, and H. Tijink, Eds. Berlin, Germany: Springer, 2001, pp. 50–59.
[29] M. Bramberger, B. Rinner, and H. Schwabach, "A method for dynamic allocation of tasks in clusters of embedded smart cameras," in *Proc. IEEE Int. Conf. Systems Man and Cybernetics*, Honolulu, HI, Oct. 2005, pp. 2595–2600.
[30] S. Russell and P. Norvig, *Artificial Intelligence–A Modern Approach*, ser. Prentice-Hall International Series in Artificial Intelligence. Upper Saddle River, NJ: Prentice-Hall, 2003, rUS st 03:1 1.Ex.
[31] P. Viola and M. Jones, "Rapid object detection using a boosted cascade of simple features," in *Proc. 2001 IEEE Computer Society Conf. Computer Vision and Pattern Recognition (CVPR 2001)*, 2001, vol. 1, no. 3, pp. 511–518.
[32] R. E. Shapire and Y. Singer, "Improved boosting algorithms using confidence-rated predictions," *Mach. Learn.*, vol. 37, no. 3, pp. 297–336, 1999.
[33] A. Klausner, A. Tengg, S. Erb, and B. Rinner, "DSP based acoustic vehicle classification for multi-sensor real-time traffic surveillance," in *Proc. EUSIPCO 07*, Poznań, Poland, Sep. 2007, pp. 1916–1920.
[34] R. Gonzalez and R. E. Woods, *Digital Image Processing*. Upper Saddle River, NJ: Prentice-Hall, 2002.
[35] M. Varma and A. Zisserman, "A statistical approach to texture classification from single images," *Int. J. Comput. Vis.: Special Issue on Texture Analysis and Synthesis*, vol. 62, no. 1, pp. 61–81, 2005.
[36] R. Duda, P. Hart, and D. Stork, *Pattern Classification*. New York: Wiley, 2001.
[37] S. Theodoridis and K. Koutroumbas, *Pattern Recognition*. San Diego, CA: Elsevier/Academic, 2006.
[38] M. Jovanovic and B. Rinner, "Middleware for dynamic reconfiguration in distributed camera systems," in *Proc. 5th IEEE Int. Workshop on Intelligent Solutions in Embedded Systems (WISES)*, Madrid, Spain, Jun. 2007, pp. 139–150.

Andreas Klausner was born in 1979 in Judenburg, Austria. He studied telematics at Graz University of Technology where he received his B.S. and M.S. degrees in 2003 and 2004, respectively. His thesis was focused on modelling integrated devices at austriamicrosystems. In 2008, he received the Ph.D. degree (with highest honors) in telematics from Graz University of Technology in cooperation with EVK. His research is focused on pervasive data fusion systems covering multilevel information fusion well as embedded pattern recognition.

Allan Tengg was born in 1979 in Judenburg, Austria. He studied telematics at Graz University of Technology, Austria. In 2004, he received the M.S. degree, doing his thesis in the field of electronics and embedded devices in cooperation with EFKON AG. He is currently pursuing the Ph.D. degree at the Institute for Technical Informatics, Graz University of Technology.

In April 2004, he joined the embedded software group at ACG ID, Graz, as a Software Engineer. The focus of his work was the design and the development of RFID reader firmwares based on PIC-powered platforms. His research interests include distributed computing on embedded systems and data fusion architectures.

Bernhard Rinner (M'95–SM'04) received both the Ph.D. and M.Sc. degrees in telematics from Graz University of Technology, Austria, in 1996 and 1993, respectively.

He is Full Professor and Chair for Pervasive Computing at Klagenfurt University, Austria, where he is currently serving as vice dean of the Faculty of Technical Sciences. He held research positions at Graz University of Technology and at the Department of Computer Sciences, University of Texas at Austin. His research interests include parallel and distributed processing, embedded systems as well as mobile and pervasive computing. He has authored and co-authored about 100 papers for journals, conferences and workshops, lead several research projects and served as reviewer, program committee member, program chair and editor-in-chief.

Dr. Rinner is the co-founder and conference co-chair of the ACM/IEEE International Conference on Distributed Smart Cameras and serves as Guest Editor of a special issue on distributed smart cameras for the PROCEEDINGS OF THE IEEE.