The 2010 IEEE/RSJ International Conference on
Intelligent Robots and Systems
October 18-22, 2010, Taipei, Taiwan

# A Cloud Computing Approach to Complex Robot Vision Tasks using Smart Camera Systems

Hannes Bistry and Jianwei Zhang

*Abstract*— **In this paper we show our work on enabling service robot systems to distribute parts of the image processing functions to different off-board computer systems in the working environment of the robot. Thus complex algorithms can be carried out on high performance systems circumventing the restrictions considering space and power consumption that a mobile platform imposes. As high resolution cameras provide a huge amount of image data and the bandwidth of a wireless network connection is strongly limited, we are using intelligent camera systems on the mobile robot platform to execute parts of the image processing functions directly on the robot. This way only preprocessed image information will be transmitted instead of raw image data. We are using a flexible modular software framework that allows us to split image processing tasks into a pipeline of modular functions that can run on different systems. We show how our approach can be used to enable a service robot system to speed up high resolution SIFT-based object detection.**

## I. INTRODUCTION

In recent years a lot of research has been done on autonomous mobile service robots. Progress has been made in all related disciplines. But there is still a big gap between the state of development of traditional robots and mobile service robots that need to cope with an unstructured and dynamically changing environment. One general challenge of mobile robot systems is the perception of this environment. Only if enough information on the position of the robot itself, on obstacles, objects and interaction partners can be acquired, it is possible to perform actions autonomously.

Vision is one of the most promising but also challenging sensor modalities that are used for robot systems. The challenge for the integration of vision systems into service robots is the high computational effort that current image processing algorithms generate. Furthermore the actual effort depends on the resolution, the format and on the number of frames processed each second. Only high-quality image data provides precise information about the environment, while a high frame rate is necessary for the system to react in real-time. On the other hand, mobile robot systems provide only a limited amount of computing capabilities, that are furthermore needed for other real-time critical sensor and actuator systems.

Therefore we are working on smart camera systems intended for the use on service robots. The general idea of smart camera systems is to generate image information instead of raw image data. This way high level results of image analysis can directly be incorporated in the task planning software of the service robot system. We are developing a software architecture to distribute parts of the image processing functions to those smart camera systems. Due to an universal implementation this architecture also allows us to integrate further dedicated systems on the robot as well as systems in the surroundings of the robot. Studies revealed that typical desktop PC are idle most of the time [1]. As the office environment of the testing platform provides many PCs, we try to take advantage of their computing power.

In this paper we will show how our distributed approach can yield advantages in the object detection process of the robot system. We chose the SIFT algorithm since it has proved to be an effective method for object detection in various research projects. The problem is that this algorithm generates high computational effort, especially if many objects need to be detected in parallel.

This research is not proposed as an advance of the SIFT-algorithm itself. Instead we take the SIFT algorithm as an example and show how a distributed implementation and a smart camera architecture can be used to integrate complex vision algorithms.

The remainder of this paper is organized as follows: In section II we present our research background and discuss the challenges that processing of image data provides. We introduce approaches of similar research projects and refer to our prior work in this research field. Section III introduces the developed software architecture for running image processing functions on intelligent cameras as well as on arbitrary computer systems. The integration of the SIFT algorithm into this framework is described in section IV. Experimental results are discussed in section V. A conclusion on the achievements and an outlook to future research is given in section VI.

## II. RELATED RESEARCH

The TAMS group is doing research on mobile robot systems with different sensor and actuator systems. The main research platform is the service robot TASER. It is mainly used for delivery tasks in an office environment with its subtasks like human interaction, localization, object detection and grasping. Descriptions of the whole architecture of TASER and of the occurring problems can be found in [2]. In the initial setup all actuator and sensor systems have been connected to one control PC. Several real-time critical components need to be operated in parallel, but especially if

This work was supported by the German BMBF (Bundesministerium für Bildung und Forschung)

Hannes Bistry and Jianwei Zhang are with the Institute of Technical Aspects of Multimodal Systems, Department of Computer Science, University of Hamburg, Germany. {bistry, zhang}@informatik.uni-hamburg.de

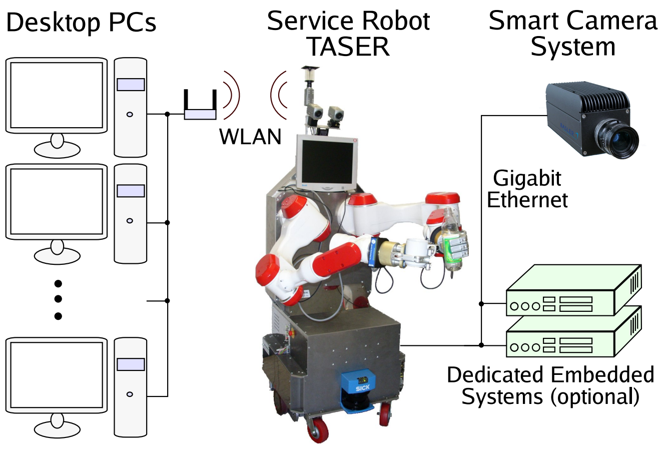

Fig. 1. This figure shows the systems that will take part in the distributed processing of visual information. The Smart camera will be connected to the control PC via Gigabit Ethernet. Additional dedicated systems may be added to increase the overall performance. The robot system can connect to external systems via a wireless network connection.

the system is loaded with computationally intensive image processing tasks, these response times cannot be guaranteed. A similar problem occurred with the connection of the laser range finders. It was solved by the development of an embedded system that preprocesses the data and transmits them via Ethernet [3]. Regarding the camera systems a similar solution is under development. As mentioned above image processing tasks can be sourced out to an intelligent camera system, to dedicated systems or to arbitrary systems in the surroundings. The setup of the connections is shown in Fig. 1. In [4] we have shown how our approach can yield advantages in face detection applications. The smart camera can be configured to transmit only image regions where faces are found. Therefore the amount of data that needs to be processed by further systems is reduced.

There are different approaches of adding smart capabilities to camera systems. One possibility is to directly integrate processing elements into so-called intelligent image sensors. An application of these kinds of sensors in high-speed object recognition and tracking is shown in [5]. The resolution of this chip is significantly better than that of prior types but still lags behind classical digital camera systems. Due to the fixed processing strategy the flexibility is strongly limited.

Smart cameras can also base on embedded processors. A comparison of two different smart camera systems using low-power embedded boards with ARM9 CPUs is shown in [6]. These systems are intended to be used for traffic surveillance. Benchmark results show that performance lags behind a desktop system. An application of face recognition running on a smart camera system with a multi-processor architecture is shown in [7]. Due to a DSP-based hardware and an optimized implementation, the process of face recognition runs in real-time.

Dedicated hardware is another way of realizing computing capabilities of smart camera systems. In [8] the authors propose an active vision system that can apply various preprocessing functions implemented as dedicated hardware. They introduce an algorithm to track an object based on a template and color segmentation based region-of-interest selection. In [9] a smart camera system intended for tracking applications is shown that can be configured to read out only a partial area of the image sensor and reaches up to 1000 frames per second.

Compared to these projects, the innovative feature of our system is that the image processing algorithms are not fixed and can be exchanged at run-time. This way we want to fulfill the needs of the heterogeneous tasks that may be assigned to a service robot system. As this paper describes our way of enhancing the use of SIFT features with intelligent camera systems and dedicated hardware, we will give some references about fundamental and recent research. The general idea of the SIFT algorithm is to find significant points in an image and describe them in a way that the description is invariant to rotation, translation and scaling. Extracted features of images can be compared to find corresponding points. If multiple corresponding points can be found, a transformation matrix can be calculated. A detailed description of SIFT-features can be found in [10].

In [11] a service robot system is described that uses SIFT-features to detect objects and build a complete model of the scene. It is stated that the described system needs about 8 $s$ for matching the feature vector to a database. Actually all known objects are stored in this database and are matched at a stroke.

A different object detection algorithm is described in [12]. It uses Gabor wavelet transformation and chooses points with the Shi-Tomasi algorithm. The matching procedure with a database of 50 objects lasts 27 $s$.

We see a big potential in our approach of distributed detection and matching. It should be possible to speed up the object detection process significantly.

## III. DISTRIBUTED SOFTWARE ARCHITECTURE

Using several systems invokes the problem that image processing sub-tasks need to be distributed to these systems properly. The overall image processing strategy needs to be divided into several parts. This possibility is necessary in order to integrate smart camera systems, dedicated systems as well as external networks systems into the visual information processing robot system.

We are developing a widespread software architecture that is capable of controlling the camera hardware and carrying out software-based preprocessing. Our general idea is to provide modular based image processing functions that are connected to a pipeline. Those pipelines can also be set up across network connections.

Our distributed software architecture can control the processing function of many systems from a single instance. The software is written for Linux, but could be transferred to other operating systems, too. Transfer and processing of image data will be done within the GStreamer [13] framework. This open-source multimedia framework is used by many multimedia applications under Linux. Many functions needed for this application are already implemented in GStreamer, like format conversion, image resizing, encoding, decoding, timing issues and network data transmission. The GStreamer framework is plugin-based, so the functionality can be expanded by new elements that also can define their own data types. There is also the possibility to set up branched pipelines where data of one image is processed by many elements in parallel.

Image processing functions can also be run on dedicated systems without camera hardware, so the tasks can be run on many systems in parallel. This can be useful when the computational effort is large compared to the additional overhead of transferring image data over the network.

Timing issues can be analyzed by the so-called timestamps that every unit of data (i.e. one image of a video-stream) provides. We set this value to the current NTP [14] timestamp directly after the image was captured. In different stages of the processing pipeline, the latency can be determined by comparing the timestamp to the current system time. Therefore, we have to synchronize all systems including the smart camera by an NTP timeserver. The achievable accuracy is better than 1 ms in a local area network. Considering the usual processing times of image processing functions and jitter caused by the scheduler of the operating system, this accuracy is sufficient.

The framework for distributed image processing is integrated in the control software system of the robot. Closely in relation to the ability to assign tasks to systems in the network environment is the possibility to come to know which systems in the surroundings are available. We are using the Roblet-framework [15] to announce the availability and handle changes dynamically.

## IV. DISTRIBUTED SIFT BASED OBJECT DETECTION

In this section we will describe the implementation of the SIFT-algorithm into our software framework.

Our implementation needs to provide flexibility for many use cases. It should be possible to match camera images with features stored in a database for object detection as well as with live camera images from another camera. We take advantage of the modular idea of our architecture and split feature extraction and feature matching into two different plugins. Different timing modes will support both live streams and static images.

### A. *Feature Extraction Elements*

The feature extraction element is the element that takes an input image in various formats and chooses significant points that are described by the SIFT algorithm. The implementation uses a modified version of the OpenCV-based [16] implementation of Hess [17]. Some performance enhancements were applied and the memory consumption was lowered by reusing the allocated memory for multiple frames of a video stream. Since feature extraction and matching take place in different elements, the representation of feature data is relevant. We implemented different methods of storing and transmitting SIFT vectors:

- Internal memory representation (1140 bytes per feature)
- Lowe-representation (361 bytes per feature (string))
- Lowe(modified) (160 bytes per feature (raw))

One method is to directly copy the internal memory representation of the applied implementation. As a lot of additional information like pointers to matching features is already stored within this structure, this method consumes quite a lot of memory. If both extraction and matching take place on one machine, it is still advantageous to use this method as no additional effort needs to be spent on format conversion. The implementation of Lowe is using plain-text representation with space-separated values. Floating point values are normalized to the range of char values. The advantage of this type of storage is that it is readable by humans, independent of the endianness of the machine and it is quite compact. But storing integer information inside of plain-text wastes a lot of memory. Therefore we implemented a modified version of Lowe's approach of storing the information in a machine readable form. This way we could shrink the amount of data significantly.

As an example we analyzed several typical scenes of the workspace of the robot with the camera system. Analyzing the gray-scale image from the 1.45 MPixel camera system yielded about 300 extracted feature points. Considering the amount of data, it is possible to store the significant image information of an image with a size of 1.41 MB in about 50 kB. This data can easily be transferred over wireless and wide area network connections.

### B. *Feature Matching Element*

The matching element takes two different feature vectors and calculates matching features. It has two input pads for the two feature vectors. To make it work with vectors of a video stream as well as with those of still images, it keeps the latest incoming data and matches each data against the latest available data if triggering is enabled for this input. For performance reasons it makes sense to build a KD-tree [18] of the features of one of the images, preferably of the still image as it only needs to be built once.

Depending on the format of input data, the first step is to convert the types of data to the internal representation. After the matching process there are several possibilities of treating the results. They can be put out as a coordinate-list of matching features. This can be useful for visualization or for external processing. It is also implemented to directly calculate a transformation between two images using the RANSAC algorithm [19]. If one of the source images contains one object, the position of this object in the second image can be calculated. For grasping tasks of the robot system this method is applied. The element can also be configured to match against many objects sequentially. This so called agglomerating mode will save all vectors received from one input (at system start) and match each incoming feature vector of the second input against each of them (at run-time). Our implementation builds a separate KD-tree for

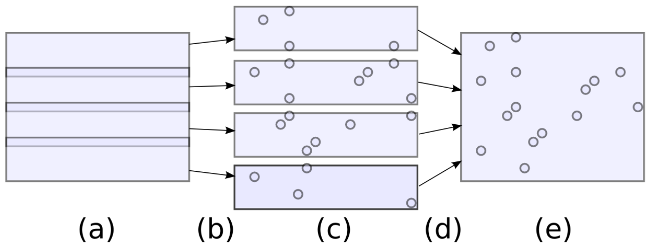

Fig. 2. This figure shows the function of distributed SIFT feature extraction. The image(a) is split into different overlapping partitions(b). Feature extraction processes(c) are run on these regions independently, the feature vectors are merged(d) into one resulting vector(e).

every object as the chance of mismatches increases with bigger KD-trees. Actually this partial aspect is a tradeoff between accuracy and speed and will not be considered any further within this work.

## C. Distributed feature detection

The elements described so far in combination with basic elements of GStreamer allow us to match the feature vector of one image against many objects. Results of this analysis will be sent to the higher-level robot control that integrates this information in its task planning process. Anticipating the results of the performance analysis, the extraction process take a lot of processing time. Therefore we analyzed ways of speeding it up. One possibility is to split the input image into several partitions and run the feature extraction process on each of them independently. The general problem here is that SIFT features are selected and described by their neighborhood. Especially the features at the border could be skipped or described incorrectly. Therefore we add an overlap to each image region. After being processed on different paths, the different feature vectors are merged in another element that checks whether features in the overlapping regions are duplicates. The principle of distributed feature detection is shown in Fig. 2.

Tests so far yielded no negative influence on the detection quality. Sometimes additional features are chosen as significant and get included in the resulting vector. This is because finding features is a process of building a Gaussian pyramid and calculating the difference of Gaussians. This implies that a large neighborhood has influence on the choice of key-points. In general, SIFT-based object detection is a probabilistic process with a lot of parameters. From tests it can be assumed that it works reliably, but it cannot be guaranteed. As the SIFT algorithm is developed to allow matching under difficult circumstances like partial occlusion of the object or different backgrounds, it can be assumed that the detection quality will not suffer from this measure.

Splitting and processing in different paths can also be applied to different image processing strategies like convolution. Here the possibility of adding overlaps is also necessary. In the described examples splitting is done along a horizontal axis. It could also be configured to be done along the vertical axis, but this would be less efficient. Due to the fact that

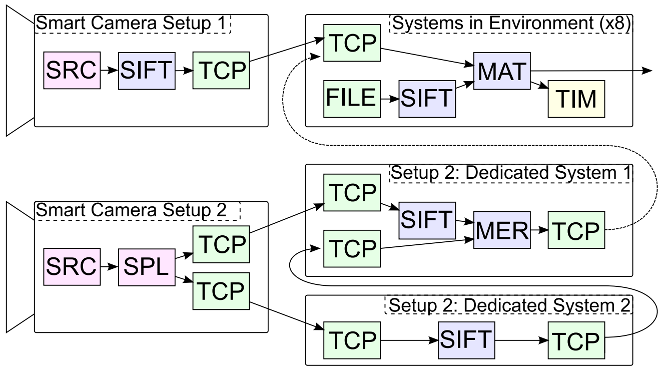

Fig. 3. This figure shows the first two setups of the processing pipelines. The actual pipelines need some more elements like format converters, queues and additional timestamp comparators. Also, the redistribution of the feature vector is not shown. The smart camera system (left side) calculates feature vectors that are sent to the systems in the surroundings (Setup 1, top). At the end of the pipeline the timestamp of the source data is compared to the current time.
Within the second setup (bottom) the smart camera splits the images of the video stream and transmits these streams independently to different dedicated embedded systems on the robot. After the resulting feature vector is calculated, it is treated like in the first setup.
Elements that are used in this setup:
**SRC**: access to the camera driver libraries, generating raw image data
**SIFT**: extracts feature vectors of images
**MAT**: matches sets of feature vectors, calculates transformation
**MER**: merges two feature vectors, eliminates duplicates
**FILE**: reads file(s)
**TCP**: transmits data through a TCP connection
**TIM**: compares the timestamp of data to current NTP time

images are stored in the memory line by line, the single slices can still use the same memory region. Thus no additional copy operations need to be done on splitting.

## V. EXPERIMENTAL RESULTS

Our implementations are tested with the commercially available smart camera Basler “eXcite exA1390-19c”. This camera can be regarded as a prototype for a future type of smart cameras. A plugin integrating the camera in the GStreamer framework has also been developed. A Gigabit Ethernet switch is installed on the robot to connect all the different systems. If the robot needs to transfer data to systems in the surroundings, the WLAN connection needs to be used. As a first test, the data-rate of the WLAN connection (802.11g) is measured, because it will have influence on the results of the tests. Therefore the robot is placed at different positions in its workspace. The test is done measuring the speed of a TCP data transfer. In places near the access point, the data rate nearly stays constant at about 2.2 MB/s. In borders of the workspace it drops to 1.4 MB/s. The following experiments are carried out in regions where the signal quality is good.

We compare four possible setups of the feature detection process. For these tests we are searching for 100 objects in parallel. Different systems take part in the image processing tasks. The first two setups are explained in Fig. 3, the setups of the latter ones are similar.

### A. Setup 1: Preprocessing on the Intelligent Camera System

In the first setup, SIFT vectors of the captured images are directly computed on the intelligent camera system and then analyzed in a cloud of computer systems. This setup pursues the goal of directly reducing the amount of data and taking advantage of the computing power of the smart camera system as much as possible. Instead of image data the camera will put out a vector of detected features. Thus the amount of data will be far beyond the amount of transmitting raw images. Within our framework we will set up an element that transmits the detected feature vector.

We distribute the matching process over eight systems in the network (Core 2 Duo, 2.4 GHz). Each of these systems generates sets of feature vectors of the objects to be detected and tries to find matches to the current feature vector. To keep the load on the wireless network as low as possible, one of these systems is used to redistribute the feature vector to all other systems. The feature matching task with the goal of detecting 100 objects is distributed to the eight systems the way that the processing times on the systems are similar. Depending of the type of the object the complexity of the matching process varies.

### B. Setup 2: Extended Preprocessing on Dedicated Systems

The second setup uses additional hardware on the service robot TASER. Two dedicated laptop computer with dual-core processors (Core 2 Duo, 2.2 GHz) are installed on the robot and can be used to compute the SIFT-vectors of the images. Here the smart camera is used to split the image in two regions that are analyzed on the two laptops independently. On each laptop they are split again into two parts to take advantage of the dual-core architecture. The SIFT-vectors are collected on one of the systems, analyzed for duplicates in the overlapping region and merged into one vector that is treated the same way as in the first example. This setup can also be seen as a simulation of future types of smart camera systems that feature more powerful processors.

### C. Setup 3: Transmission of Raw images

In the third setup the complete set of processing functions will be run on networked systems, so that the full images need to be transmitted over the wireless network connection. Within this setup the smart camera acts like a conventional Ethernet camera. The dedicated laptop systems as they are described in setup 2 are connected to the stationary network in the surroundings of the robot. From the point of view of system configuration this test is similar to setup 2 with the difference that no preprocessing will be done on the robot platform. By comparing these setups we want to show the advantages of preprocessing directly on the robot system.

### D. Setup 4: Processing on the Control PC of TASER

Within the fourth setup, all functions are run on the control PC of the service robot (Intel Pentium 4, 2.4 GHz). In this setup, the camera is also configured to apply no preprocessing to image data. This setup is used to compare our approach to the classical approach of computing all image processing functions on the control PC. This PC is running several real-time critical tasks that load the system to about 60 % so the available computing capabilities are strongly limited.

### E. Results

Within testing procedures it emerged that image content has a significant influence on the feature extraction time. The more features are found, the higher is the effort to calculate them. Therefore for all benchmarks the robot will analyze the same table scene. The processing steps are analyzed individually for their duration. The extraction step is declared to be finished when all parts of the feature vector have been merged together. Matching is finished, when the last system has finished its part of the matching procedure. The data transfer concludes all steps like transferring data from the camera to systems on the robot, wireless data transfer from the robot to stationary systems as well as transfer of the results to the control PC of the robot.

| Setup | Extraction | Matching | Data Transfer |
|---|---|---|---|
| 1 | 6781 $ms$ | 381 $ms$ | 16 $ms$ |
| 2 | 269 $ms$ | 376 $ms$ | 62 $ms$ |
| 3 | 274 $ms$ | 383 $ms$ | 695 $ms$ |
| 4 | 4110 $ms$ | 10428 $ms$ | 50 $ms$ |

The sequence of processing steps with their durations is also shown in Fig. 4. Compared to the setup where all processing functions are run on the control PC of the service robot, the distributed processing approach yields significant advantages. The total duration until results of scene interpretation are available could be reduced from 14.6 $s$ to 0.7 $s$. Therefore it becomes possible to do scene recognition without generating noteworthy delay. The biggest advantage is achieved by distributing the task of feature matching to several systems. This approach is scalable in great measure, so even more systems could be integrated or the database of objects could be extended. Distributed feature extraction also contributes to the performance of the overall system. By way of comparison the feature extraction on the full image has been carried out on one of the same system and took about 622 $ms$. On the control PC of the service robot it took about 2 $s$ without any additional load. A further advantage of our approach is that the amount of data that is transmitted over the wireless connection is reduced by preprocessing (setup 1 & 2). This leads to a lower latency and ensures that enough bandwidth will be available for other tasks. The results also show that the processing power of the currently used smart camera system is too low in consideration of the complexity of recent image processing functions. If the camera would provide a more powerful CPU, results like those of the second setup could be achieved without the need for additional systems. The tests can also be seen as a demonstration of the flexibility of our developed architecture since all the different complex tests were set up using just few basic processing elements. A visualization, where the detected objects and the matches are overlayed above the image is shown in Fig. 5.

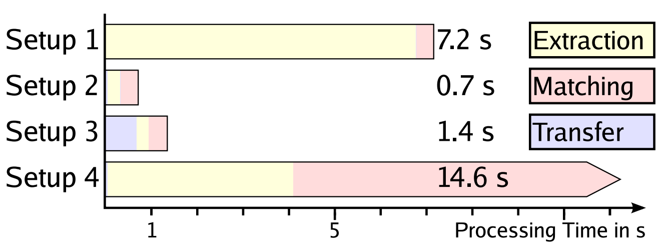

Fig. 4. This figure shows the results of timing analysis of the different setups and the overall latency. Significant data transfer time occurs only in the third setup.

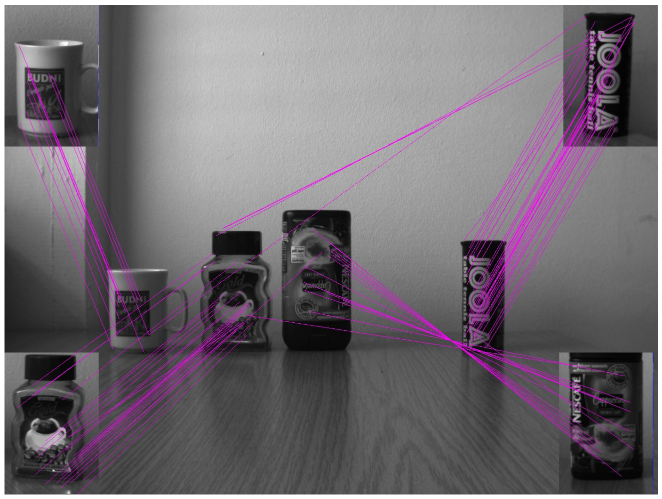

Fig. 5. This images shows a table scene with detected objects and the matches between features of the model and the corresponding ones of the image. Some mismatches occur but due to the high amount of correct matches these can be filtered out.

## VI. CONCLUSIONS AND FUTURE WORK

In this paper, we presented how our framework for intelligent camera systems can be used to distribute parts of the image processing functions to different systems via a network connection.

The image processing functions on the camera systems and further systems can be adapted according to the current task of the robot. We showed different configurations where the overall system is configured to detect objects based on SIFT-features. The latency could be reduced significantly using the computing capabilities of desktop PCs in the surroundings and additional dedicated systems on the robot. This way the robot system is now also able to build a model of its environment online without interrupting its workflow.

As current smart camera systems provide strongly limited computing capabilities, we presented some benchmarks where dedicated mobile computer systems were also installed on the robot platform to simulate future types of camera systems. Due to the portability of our framework we could easily take advantage of additional computing power by a simple reconfiguration of processing elements.

As a future extension, it is planned to integrate GPU-based implementations into our architecture. The modular approach makes it easy to exchange the the processing elements while keeping the rest of the pipeline configuration. Therefore we could take advantage of systems in the network with high performance GPUs, but it is also thinkable that future types of cameras will provide chipsets that feature GPUs for general purpose computing tasks.

## REFERENCES

[1] J. Gray. Distributed computing economics. *Queue*, 6(3):63–68, 2008.

[2] A. Maeder, H. Bistry, and J. Zhang. Intelligent Vision Systems for Robotic Applications. *International Journal of Information Acquisition (IJIA)*, 5(3):259 – 267, September 2008.

[3] H. Bistry, D. Westhoff, and J. Zhang. A smart interface-unit for the integration of pre-processed laser range measurements into robotic systems and sensor networks. *Intelligent Robots and Systems, 2007. IROS 2007. IEEE/RSJ International Conference on*, pages 358–363, November 2007.

[4] Hannes Bistry and Jianwei Zhang. Task oriented control of smart camera systems in the context of mobile service robots. *Intelligent Robots and Systems, 2009. IROS 2009. IEEE/RSJ International Conference on*, pages 3844–3849, October 2009.

[5] T. Komuro, A. Iwashita, and M. Ishikawa. A qvga-size pixel-parallel image processor for 1,000-fps vision. *Micro, IEEE*, 29(6):58 –67, November 2009.

[6] N.F. Kahar, R.B. Ahmad, Z. Hussin, and A.N.C. Rosli. Embedded smart camera performance analysis. volume 2, pages 79 –83, January 2009.

[7] H. Fatemi, R. Kleihorst, H. Corporaal, and P. Jonker. Real time face recognition on a smart camera. *Proceedings of ACIVS 2003 (Advanced Concepts for Intelligent Vision Systems)*, 2003.

[8] P. Chalimbaud and F. Berry. Embedded active vision system based on an FPGA architecture. *EURASIP Journal on Embedded Systems*, 2007(1):26–26, 2007.

[9] I. Ishii, K. Kato, S. Kurozumi, H. Nagai, A. Numata, and K. Tajima. Development of a mega-pixel and milli-second vision system using intelligent pixel selection. In *First IEEE Technical Exhibition Based Conference on Robotics and Automation, 2004. TExCRA'04*, pages 9–10, 2004.

[10] D.G. Lowe. Object recognition from local scale-invariant features. *International Conference on Computer Vision*, 2:1150–1157, 1999.

[11] J. Kuehnle, A. Verl, Zhixing Xue, S. Ruehl, J.M. Zoellner, R. Dillmann, T. Grundmann, R. Eidenberger, and R.D. Zoellner. 6d object localization and obstacle detection for collision-free manipulation with a mobile service robot. pages 1 –6, June 2009.

[12] V. Lepetit, J. Pilet, and P. Fua. Point matching as a classification problem for fast and robust object pose estimation. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, volume 2. IEEE Computer Society, 2004.

[13] W. Taymans, S. Baker, A. Wingo, R. Bultje, and S. Kost. *GStreamer Application Development Manual (0.10.21.3)*. http://gstreamer.freedesktop.org, October 2008.

[14] David L. Mills. *Computer Network Time Synchronization: The Network Time Protocol*. CRC, 1 edition, March 2006.

[15] Daniel Westhoff, Hagen Stanek, and Jianwei Zhang. Distributed applications for robotic systems using roblet-technology. In *Proceedings of the 37th International Symposium on Robotics and Deutsche Fachtagung Robotik 2006*, Munich, Germany, May 2006. VDI/VDE-Gesellschaft Mess- und Automatisierungstechnik.

[16] G. Bradski. The openCV library. *DOCTOR DOBBS JOURNAL*, 25(11):120–126, 2000.

[17] R. Hess and A. Fern. Improved video registration using non-distinctive local image features. In *Proc. IEEE Conf. on Computer Vision and Pattern Recognition*, 2007.

[18] Jon Louis Bentley. Multidimensional binary search trees used for associative searching. *Commun. ACM*, 18(9):509–517, 1975.

[19] M.A. Fischler and R.C. Bolles. Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography. 1981.