Hindawi Publishing Corporation
International Journal of Reconfigurable Computing
Volume 2012, Article ID 368351, 16 pages
doi:10.1155/2012/368351

Research Article

Object Recognition and Pose Estimation on Embedded Hardware: SURF-Based System Designs Accelerated by FPGA Logic

Michael Schaeferling, Ulrich Hornung, and Gundolf Kiefer

Department of Computer Science, Augsburg University of Applied Sciences, An der Hochschule 1, 86161 Augsburg, Germany

Correspondence should be addressed to Michael Schaeferling, michael.schaeferling@hs-augsburg.de
and Gundolf Kiefer, gundolf.kiefer@hs-augsburg.de

Received 4 May 2012; Revised 17 September 2012; Accepted 17 September 2012

Academic Editor: René Cumplido

State-of-the-art object recognition and pose estimation systems often utilize point feature algorithms, which in turn usually require the computing power of conventional PC hardware. In this paper, we describe two embedded systems for object detection and pose estimation using sophisticated point features. The feature detection step of the "Speeded-up Robust Features (SURF)" algorithm is accelerated by a special IP core. The first system performs object detection and is completely implemented in a single medium-size Virtex-5 FPGA. The second system is an augmented reality platform, which consists of an ARM-based microcontroller and intelligent FPGA-based cameras which support the main system.

1. Introduction

Computer vision (CV) and augmented reality (AR) are growing areas of research with many applications. For example, automotive industry makes use of passive optical sensors in the field of offboard traffic observation and management [1–3]. Onboard systems often utilize CV techniques for driver assistance and traffic sign detection [4, 5]. Turning CV to account, AR enhances real environments by virtual elements and allows manifold applications such as guided order picking or maintenance tasks [6, 7].

Optical object detection and pose estimation are very challenging tasks since they have to deal with problems such as different views of an object, various light conditions, surface reflections, and noise caused by image sensors. Presently available algorithms such as SIFT or SURF can to some extent solve these problems as they compute so-called point features, which are invariant towards scaling and rotation [8–11]. However, these algorithms are computationally complex and require powerful hardware in order to operate in real time. In automotive applications and generally in the field of mobile devices, limited processing power and the demand for low battery power consumption play an important role. Hence, adopting those sophisticated point feature algorithms to mobile hardware is an ambitious, but also necessary computer engineering task.

This paper describes two embedded systems for SURF-based object recognition and pose estimation. The first system performs feature-based object recognition and has been implemented as a SoC on a single FPGA (Virtex-5 FX70). It can process images at a frame rate of up to five frames per second and (in our experiments) recognize and distinguish 9 different objects at a sensitivity of 91% and a specificity of 100% (no false positives). The second system determines the 3D pose of one or more objects. It features an ARM microcontroller board and is equipped with two FPGA-based cameras which support the main system in the stage of feature detection. As this system is very power efficient, it may be used as a mobile AR system, for example, for educational or training applications.

Among the existing feature detection and description algorithms, SURF is considered to be both robust and efficient [8, 12]. However, it is still very demanding in terms of computational effort and memory bandwidth, especially within the detector stage. One of the core parts of the presented systems is *Flex-SURF+*, a configurable hardware module which accelerates the time-critical part of the SURF detector [13, 14]. It contains an array of *Difference Elements*

(DEs) in order to overcome the irregular memory access behavior exposed by SURF during the computation of image filter responses. A special computing pipeline processes these filter responses and determines the final detection results. Due to its customizable and flexible structure, *Flex-SURF+* allows a tradeoff between area and speed. This flexibility allows to efficiently implement this IP core within high-end FPGAs as well as low-end ones, depending on the application requirements. Besides accelerating the SURF algorithm by means of configurable hardware, lightweight object recognition and pose estimation algorithms have been designed to minimize the computational effort and memory requirements of the software part.

Section 2 gives an overview on related work concerning different implementations of SURF, object recognition, and pose estimation. Section 3 summarizes the SURF algorithm and describes the *Flex-SURF+* hardware module which is used to accelerate the feature detection step of SURF. Section 4 presents the one-chip object recognition system, followed by the pose estimation system in Section 5. The main hardware components and developed software frameworks are disclosed, providing an overview on the algorithms and some implementation details. A conclusion to this work is provided in Section 6.

2. Related Work

With increasing processing power of conventional PC hardware, object recognition nowadays is often based on natural features. These may be global or local features, where global features evaluate the whole image space (e.g., by comprising the image histogram) and local (point) features solely use significant areas of the image to gain scene information [15]. Many application fields greatly benefit from feature recognition techniques. Especially autonomous robot systems as well as automotive systems are popular fields where optical image processing is used to solve manifold problems [16]. Recent publications even deal with safety-related topics such as car recognition, collision avoidance, or traffic and danger sign recognition [1–3, 17, 18]. The impact of very difficult light conditions may be weakened, for example, by applying point feature algorithms to infrared images for pedestrian or vehicle detection [19].

Common algorithms for detecting and describing point features are the Harris Corner Detector [10], SIFT [11], and SURF [9]. Providing local features, both SIFT and SURF actually outperform global features whereby in terms of speed SURF is considered to be ahead of SIFT [8, 12]. These algorithms have been developed with high distinctiveness of features in combination with scale and rotation invariance in mind. As such, they are very demanding in terms of computational power and in their original form need to be implemented on powerful, PC-like hardware [16].

For this reason, it is desirable to speed up these algorithms. This may, for example, be achieved by the use of modern GPU hardware [20, 21]. Other proposals focus on reducing the algorithmic complexity, for example, by combining different approaches in feature recognition and description [22] or software optimizations [23].

An important technique to accelerate feature detection is the use of programmable logic (FPGAs). Application-specific logic allows to implement specific functionality at low power consumption compared to conventional PC and GPU hardware, making it an attractive candidate especially for mobile applications. An overview of several implementations of the SIFT algorithm is given in [24]. FPGA implementations of SURF have been proposed independently in [13, 25, 26]. In contrast to the other two approaches, the architecture of [13] is scalable and does not need any block RAM resources of the FPGA, thus it has been chosen to be integrated in both the object recognition system and the augmented reality framework.

3. Hardware-Accelerated Feature Extraction

The most time-critical components of the presented object recognition and pose estimation systems are the detection and the description of SURF feature points.

In the following subsections, we give a short overview on the SURF algorithm and present an outline of the *Flex-SURF+* module. This module gains speedup by improving memory access behavior and parallelizing arithmetic operations where it is adequate.

3.1. Overview on the SURF Algorithm. *Flex-SURF+* is based on the *OpenSURF* implementation of the original SURF algorithm. (Open)SURF operates in two main stages, namely the *detector* and the *descriptor* stage. The detector analyzes the image and returns a list of center points for prominent features. For each point, the descriptor stage computes a vector characterizing the appearance of the feature. Both stages require an *integral image* to be computed in advance. Figure 1 shows the overall flow of the SURF algorithm.

Integral images can be used to efficiently calculate the sum of pixel values of upright rectangular image areas. Given the original image I, each pixel (x, y) of the integral image I_Σ contains the sum of all image pixel values above and to the left of (x, y):

$$I_\Sigma(x, y) = \sum_{i=0}^{i \le x} \sum_{j=0}^{j \le y} I(i, j). \tag{1}$$

After calculating the integral image once, the summation of any rectangular area of the original image can be determined with a constantly low effort by adding/subtracting the four integral image values enclosing the desired rectangle as depicted in Figure 2. In SURF, integral images are used to minimize the incurring memory bandwidth and computational effort when calculating box filter responses in the detector and descriptor stage.

In the detector stage, SURF first filters the entire image using 3 box filter kernels, which are depicted in Figure 3. White, gray, and black pixels refer to a weighting of the corresponding pixel value by 1, 0, and -2 for D_{xx} and D_{yy} filters and 1, 0, and -1 for D_{xy}, respectively. In order to achieve scale invariance, the image is filtered multiple times for different scale levels σ, where the kernel sizes increase with σ (see [9] for details). With the concept of integral

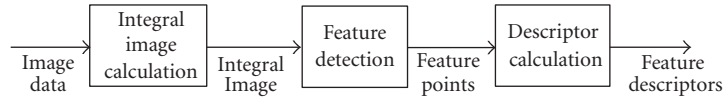


Figure 1: Stages of the SURF algorithm.

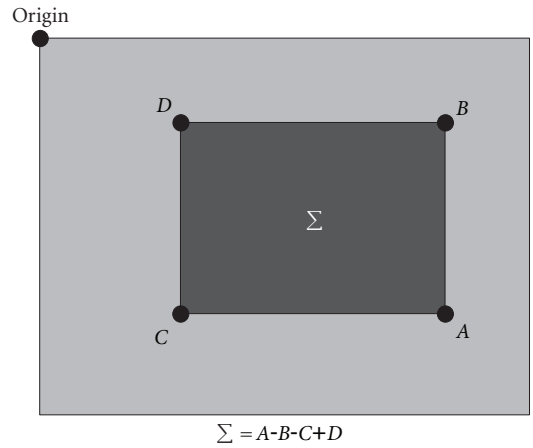


Figure 2: Summation calculation of a rectangular image area using an integral image.

images, the number of memory accesses is independent of σ, as only those few pixels marked with a dot in Figure 3 need to be accessed. Figure 4 depicts the combination of all these pixels for two different kernel sizes.

Box filter responses D_{xx}, D_{yy} and D_{xy} represent the entries of the Hessian Matrix H, which are determined for each point coordinate (x, y) and scale level σ. As the used filter kernels are actually approximations of Gaussian filters, a weight factor ω is used to compensate the occurring approximation error:

$$H(x, y, \sigma) = \begin{bmatrix} D_{xx}(x, y, \sigma) & \omega D_{xy}(x, y, \sigma) \\ \omega D_{xy}(x, y, \sigma) & D_{yy}(x, y, \sigma) \end{bmatrix}. \quad (2)$$

In *Flex-SURF+*, a value of $\omega^2 = 0.875$ is used as proposed in [18]. The determinant $\det(H)$ is an indicator for the quality of a feature point candidate:

$$\det(H) = D_{xx}D_{xy} - \omega^2 {D_{xy}}^2. \quad (3)$$

Determinants that fall below a given threshold T or which are not the local maximum within their $3 \times 3 \times 3$ neighborhood around (x, y, σ) are suppressed.

The descriptor stage of SURF calculates a descriptor vector for each feature found by the detector. To this end, Haar wavelets are applied to the pixels within a square window which is centered around the feature point and whose size depends on the features scale σ. To achieve rotation invariance, it optionally can be rotated according to a feature direction. The final result is a 64-dimensional floating point descriptor vector which is commonly used in a subsequent feature matching stage. To speed up the matching process, feature candidates are partitioned with the help of the sign of the Laplacian $\text{lap}(H)$:

$$\text{lap}(H) = \text{sgn}\left(D_{xx} + D_{yy}\right). \quad (4)$$

More details on SURF and *OpenSURF* can be found in [9, 27], respectively.

3.2. Overview of the Flex-SURF+ Module. The design of *Flex-SURF+* uses the concept of *tile-based processing*, as it is introduced in [13] and further developed in [14]. Its main idea is to minimize repeated memory accesses to the same pixels.

During image filtering in the detector stage of SURF, scattered memory accesses at multiple filter sizes according to Figure 4 are needed for the calculation of the filter responses. However, memory footprints for neighboring points overlap. Figure 5 shows the combined footprint which results from applying 9×9 filter kernels to a group of 4×4 neighboring points. The darkness of a cell indicates how often the respective memory location is accessed. A *tile* is defined by the minimum rectangle enclosing the overlapping memory footprints for a group of points. The whole image is divided into such point groups, whose filter responses are now calculated in parallel. The *Flex-SURF+* module contains an array of small worker units called *Difference Elements (DE)*, whereby the size of the array corresponds to the arrangement of the point groups. Each *DE* is responsible for the calculation of the filter responses D_{xx}, D_{yy}, and D_{xy} for one particular point of the group. A central instance performs the memory accesses and efficiently reads the required tile data in a linear way. Each worker unit autonomously picks the relevant image data from the memory data stream. After completing the filter

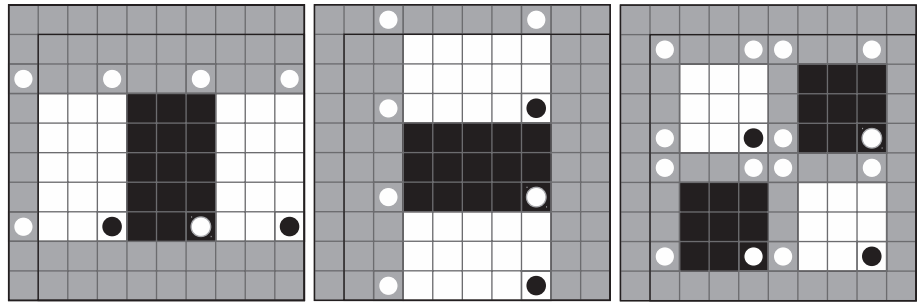


Figure 3: D_{xx}, D_{yy}, and D_{xy} filters of size 9×9.

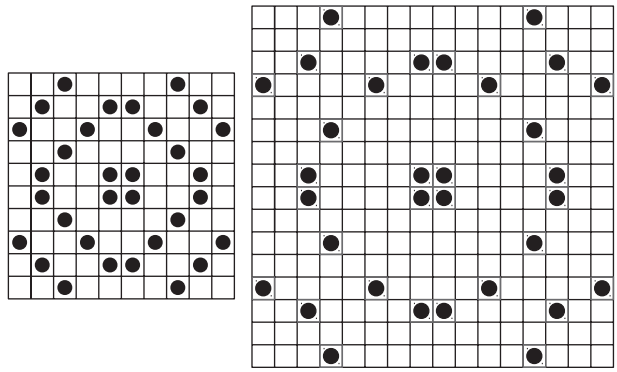


Figure 4: Joint memory footprint of D_{xx}, D_{yy}, and D_{xy} filter kernels of size 9×9 (left) and 15×15 (right).

response calculations, the worker units pass their results to a central unit which calculates determinants and Laplacians.

The main building blocks of *Flex-SURF+* are depicted in Figure 6. The *Memory Access Module (MAM)* contains the main controller logic which performs linear memory accesses over the system bus. After issuing burst read accesses to this bus, incoming pixel values are distributed to a configurable and arbitrarily large array of *Difference Elements (DEs)* over the *Image Data Bus (IDB)*. Simultaneously with the incoming *Image Data Enable* signal from the *MAM*, an *Enable Unit (EU)* generates enable signals, depending on the recent filter kernel size as illustrated in Figure 4. According to these enable signals, each *DE* picks the relevant pixel data from the *IDB* and calculates its filter responses.

In order to save area, multiple *DEs* may share one *EU*, together forming a *DE line*. Over the *Filter Response Chain (FRC)*, the filter responses are passed to the proximate *Determinant Calculation Pipeline (DCP)* which then calculates determinants and Laplacians. Finally, determinants and Laplacians are written back into system memory by the *MAM* which consecutively are used in the descriptor calculation step.

Unlike the work presented in [25] or [26], this whole system does not need any internal memory arrays and can therefore be implemented without using any valuable block RAM resources.

3.3. Array of Difference Elements. The efficiency of *Flex-SURF+* highly depends on the number of filter operations which can be performed in parallel. To allow large arrays of *DEs*, it is essential to minimize the size of an individual *DE*.

One *DE* merely consists of two 26 bit adders, three 26 bit registers to hold intermediate results, and one 21 bit register for result propagation as depicted in Figure 7.

Each *DE* receives the image data stream, broadcasted via the *IDB*. According to the *EU's* enable signal, each *DE* picks relevant image data from the IDB and immediately accumulates this value to the corresponding D_{xx}, D_{yy} or D_{xy} filter value. As soon as all of the three filter responses are fully computed by a *DE*, they are passed to the *DCP*. To avoid large multiplexer structures, we decided to daisy-chain (*FRC*) the *DEs* for the result propagation towards the *DCP*. As the *DEs* provide filter responses one after another anyway, this propagation structure does not introduce any significant delay to the overall system.

3.4. Determinant Calculation Pipeline. The *DCP* can process incoming data at every clock cycle and sequentially receives the *DE's* filter responses. Filter values are additionally marked and recognized by a "valid" flag as they may arrive irregularly, for example, due to possible delays during memory access operations over the system bus when reading tile data. Determinants and Laplacians are calculated according to (3) and (4) and finally written back to system memory.

The *DCP* contains six 18 bit multipliers (e.g., 32 bit multiplications are broken down to four 18 bit multiplications), a small set of adders and some registers to hold intermediate results. The *DCPs* structure is shown in Figure 8, where

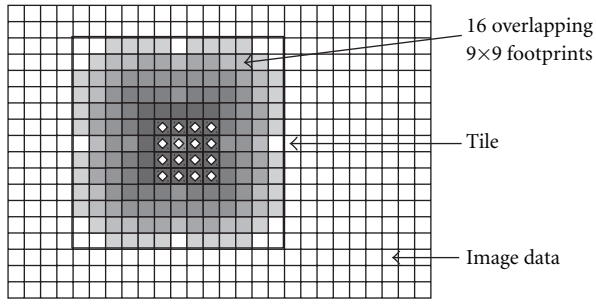


Figure 5: Overlapping footprints inside a *tile*.

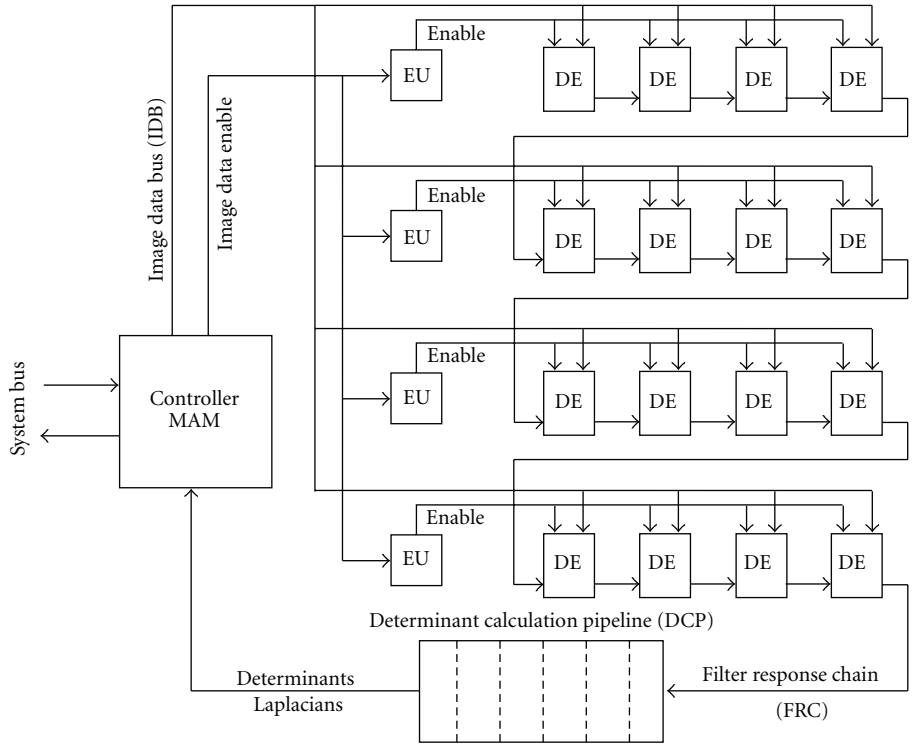


Figure 6: Overall structure of the *Flex-SURF+* module with 16 quadratically grouped *DEs* (4 *DE lines*, each containing 1 *EU* and 4 *DEs*).

inv_area is a precomputed constant required to normalize determinants according to the current filter size.

4. Object Recognition on a Single FPGA

The first system which makes use of the *Flex-SURF+* hardware module performs the task of object recognition. It searches for known objects within a live video stream, and labels them with their name in the output image. This system is completely integrated on a single FPGA chip. The following paragraphs give an overview on the overall system design, how object recognition is performed with help of the *Flex-SURF+* module, and the results which have been achieved in experiments.

4.1. System Overview. The entire hardware system for object recognition, covering all components to process the relevant image data, is depicted in Figure 9. It is integrated as a System-on-Chip (SoC) which is implemented in a single medium-size FPGA (Xilinx Virtex-5 FX70T). Xilinx Platform Studio 11.5 was used for system synthesis.

The SoC consists of several custom IP soft cores, a hard-wired PowerPC CPU core (400 MHz), and a DDR2-SDRAM controller. All these modules are connected via a CoreConnect bus system. It may be noted that this system is not constrained to the presence of a PowerPC processor, which may easily be replaced by another hard or soft core processor (as can be seen in Section 5). The custom IP cores are controlled by the main program running on the CPU and primarily handle image processing tasks.

The *VDEC* module fetches image data from a camera source and writes it into system RAM. It is parametrizable at operating time in terms of the desired resolution, the destination memory, and the operating modes "continuous stream" or "triggered single frame".

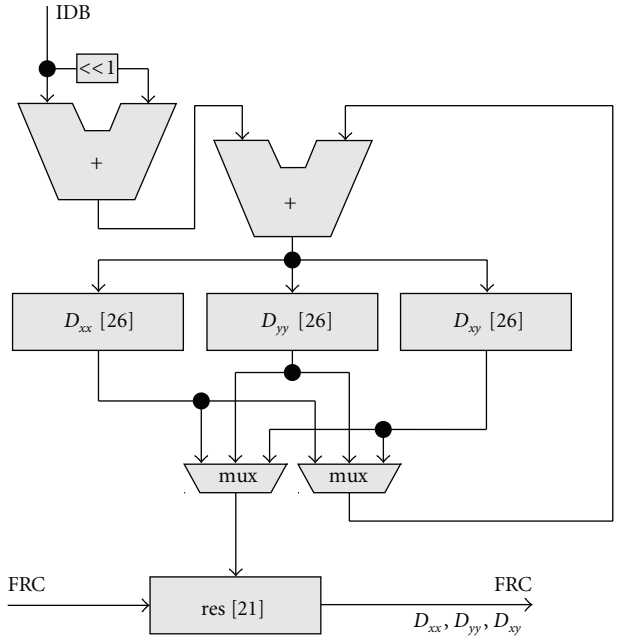


Figure 7: Structure of one *Difference Element* for calculating the filter values of D_{xx}, D_{yy}, and D_{xy}.

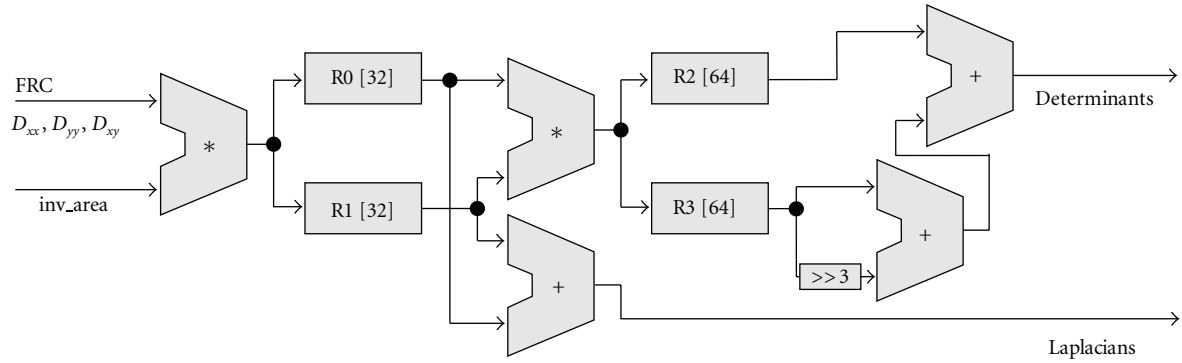


Figure 8: Structure of the *Determinant Calculation Pipeline* for calculating determinants and Laplacians according to (3) and (4).

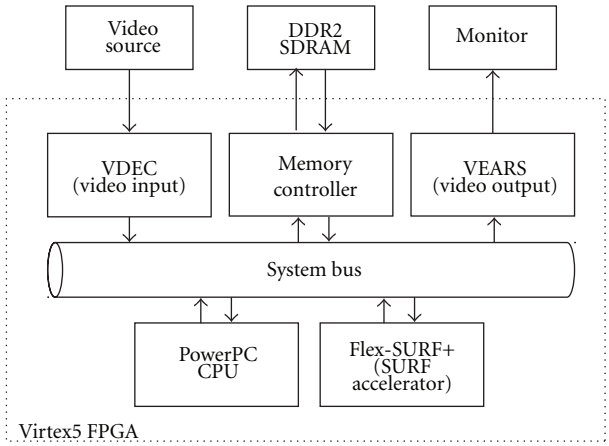


Figure 9: Structure of the Object Recognition System.

Table 1: Parameters of the first experiment.

Flex-SURF+ configuration	
DE lines	8
DEs per *DE line*	8
Database	
Image resolution	320 × 240
Number of images	16
Number of objects	9
Number of features	856
Live images	
Image resolution	320 × 240
Number of images	76
Number of object instances	73
Average live features	50

The *Flex-SURF+* module is used to accelerate the object recognition algorithm. It is clocked at the system bus clock rate of 100 MHz and was implemented at different configurations (varying number of *DEs*) in the Virtex-5 FX70T FPGA.

The *VEARS ("Visualization for Embedded Augmented Reality Systems")* module finally outputs augmented image data to a monitor. The live image data stream is read from memory and overlayed by information gained from the object recognition algorithm. Augmentation thereby is done by labeling all found objects in the video output stream.

4.2. Database Creation. The object recognition system identifies objects by comparing live image descriptors with a database, which is created in advance by a PC tool. As input it takes a few images covering views of each object to be recognized. These images do not need to be specially prepared, and typically only one or very few snapshots depicting the object are sufficient.

Database creation plays a very important role in achieving good detection rates at high performance. Due to the limited resources in the live system, the database should be as small as possible, while still allowing a high recognition rate. This has been addressed by the following optimizations.

First, the database tool does not store all, but only a small number of "strong" descriptor vectors for each object. The "strength" of a feature is determined by the determinant $\det(H)$.

Second, the tool avoids to keep descriptors for features which may similarly be found on another object already stored in the database. Such descriptors may be liable for mismatches and detecting objects incorrectly. To achieve this goal, the tool continuously matches new descriptors against existing ones and, in case of similarity (low Euclidian distance), marks them and discards all of the corresponding descriptors at the end of database creation.

Keeping only few and strong descriptor vectors helps to reduce the computational complexity of the live system in two ways. First, fewer comparisons of live image features with database features have to be performed in the matching stage. Second, since the determinant value $\det(H)$ is already known after the detector stage, time-consuming calculations of descriptors for "weak" features can be omitted.

The database itself is stored in a binary format, favoring short loading times and reducing data overhead. An object entry consists of the assigned object name and a small set of unique descriptors.

4.3. Matching and Recognition in the Live System. In the object recognition system, live images are continuously acquired from the video source. For each live image, the detector and descriptor stages are applied successively. Both stages may be applied either to the whole image or to one or more regions of interest (ROI). When ROIs are used in the object recognition system, they are set to the detected objects center positions of the preceding frame and cover a square area with a fixed edge length of 100 pixels. The advantage of using ROIs is the expected large speedup compared to full image space search, but it also entails that new objects, which may appear outside the ROI, would not be found until a full image space search is applied.

After the execution of the SURF algorithm, the resulting descriptors are matched against the database. For each database descriptor, the two live image descriptors with the lowest Euclidian distance are determined. The ratio of the first and the second best match is an indicator for the uniqueness of the best match. Thus, ambiguous matches are discarded to avoid false matches [11]. If the algorithm has attained a minimum number of unique matches for one specific database object (three in our implementation), the object is considered to be recognized.

4.4. Experiments and Results. For the experiments, we chose 9 different objects to be recognized by our system. In a first step, we took between 1 and 3 grayscale pictures of each object at a resolution of 320 × 240 pixels. The final database, which was used in all of our experiments, incorporated 16 images of 9 objects, which are all depicted in Figure 10, and contains a total of 856 SURF feature descriptors.

To evaluate the live system, a total of 76 test images (grayscale, 320 × 240 pixels), each showing 0, 1, or 2 known objects, were taken at various distances and angles. We used diffuse daylight as we decided not to lighten the scene specifically. Among the 76 test images, 12 contain extremely distracting background, but none of the known objects (see Figure 11(c) for an example), and one image contains a known object in front of distracting background (see Figure 11(b)). The remaining images show 1 or 2 of the known objects (see Figure 11(a) for an example). For ROI evaluation, the test image set contains a series of 23 images with the cup horizontally sliding through these images. If visible, the test objects covered between 8% and 98% of the image area. In three images, test objects were concealed by approximately 50% while in the other images they were fully visible.

Experiment 1 uses parameters as summarized in Table 1. Experiments 2 through 5 were performed to evaluate the impact of various parameters on the detection rate and performance of the system. Unless stated otherwise, the experimental setup was the same as in the first experiment.

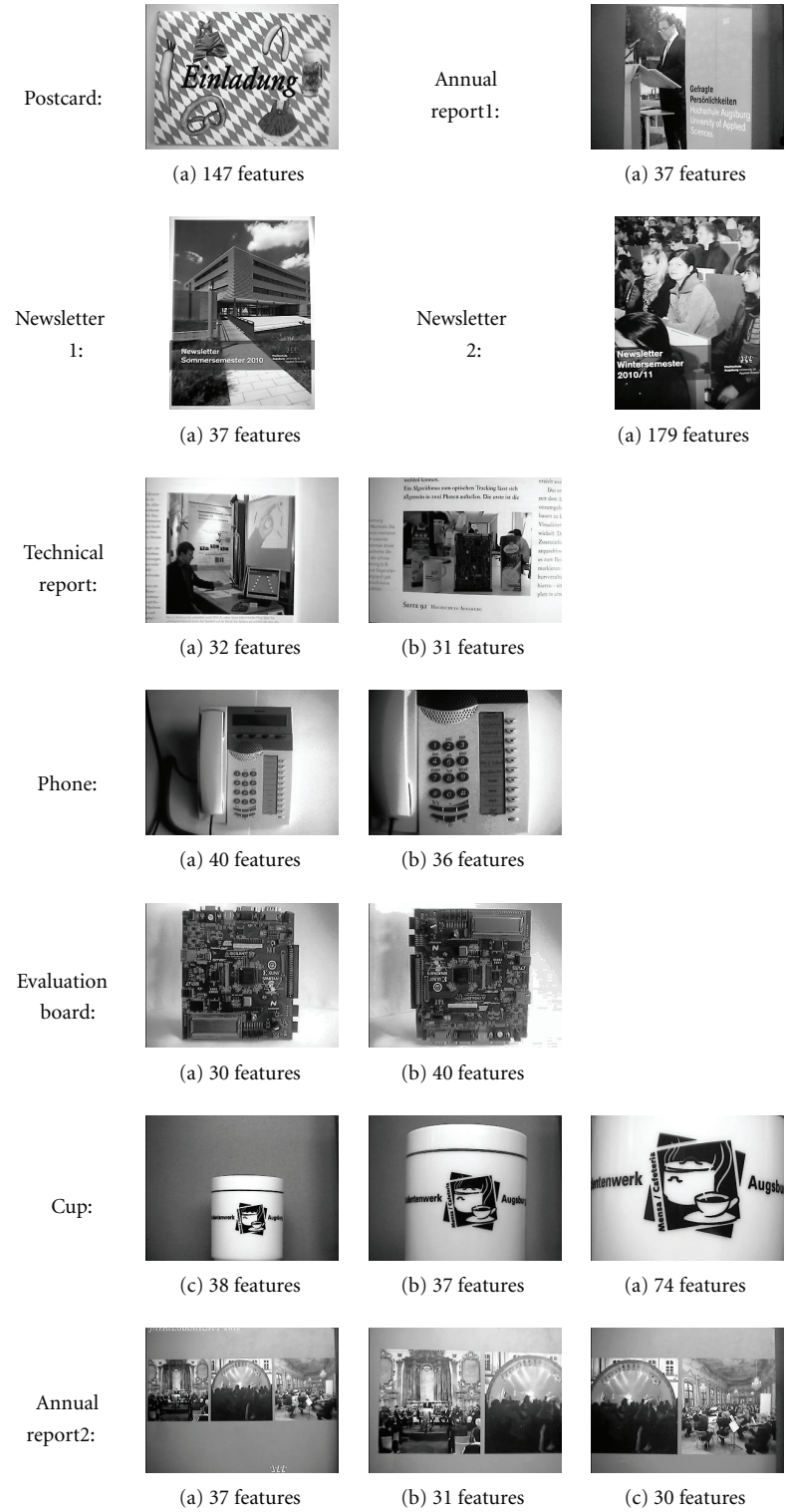
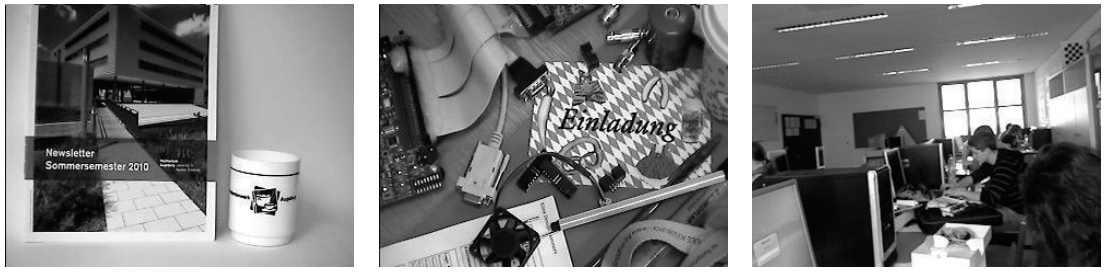


Figure 10: Reference images for creating the database.

In the first test run, the full image space was inspected by the detector and the descriptor stage (no ROI optimization). In all test images, 68 instances of known objects are found. Thus, 7 out of 73 object instances were not recognized, resulting in a sensitivity rate of 91%. The specificity was 100%, there were no false-positives in any of the 76 test images. Figure 12 shows the minimum, average and maximum execution times of the detector, descriptor, and matching stage. By the use of *Flex-SURF+*, the SURF detectors determinant calculation step requires just 70 ms per frame.



(a) (b) (c)

Figure 11: Test image examples.

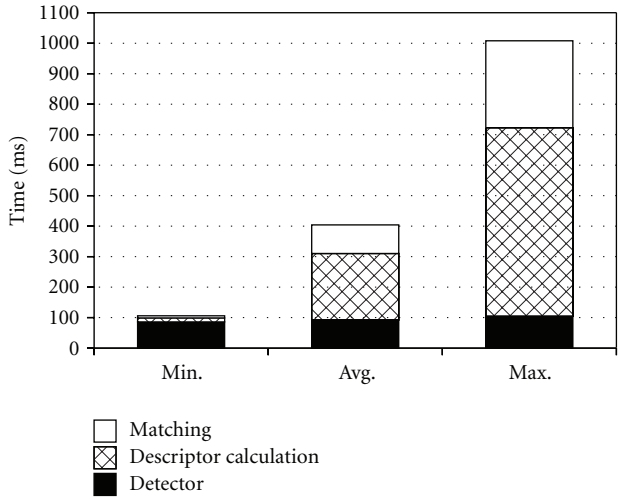


Figure 12: Time consumption of the object recognitions main tasks for 320 × 240 pixel images.

The minimum total execution time per frame was 120 ms. The maximum execution time was raised by an image comprising much distracting background, which demanded 1019 ms with 141 features found. On average, processing time took 418 ms with an average of 50 features found. Total execution time varies, as descriptor calculation and matching times are proportional to the number of features found in an image by the detector.

In the second test run we measured execution time for images with a resolution of 640 × 480 pixels in order to evaluate the impact of the image resolution on the execution time. Therefore we scaled the 76 test images up in order to maintain comparability to the previous results. Figure 13 shows the average resulting execution times. As expected, the feature detection task lasts four times longer as the image area also quadrupled. However, the durations of the descriptor and the matching stages varied to a smaller extent, and an amount of 138 features found in average contributes to the increasing general execution time.

In the third experiment, the use of ROIs was activated for the detector and descriptor stage in order to speed up the recognition system. This test covered the sliding cup image set (320 × 240 pixel images), and like in the test run on full image search, the cup was detected in 22 of 23 test images. This resulted in 2 full image searches as the ROI has not been set in the starting image and in the image succeeding the one where the cup was not recognized. Within the remaining 21 images, a ROI was used and average and minimum execution times decreased significantly. Notably, the SURF detectors determinant calculation step was 5 times faster when using the ROI: instead of 70 ms it required only 13 ms per frame. The average execution times for all stages of the SURF algorithm are depicted in Figure 14.

In order to evaluate the performance of the matching stage, we varied the number of objects within the database in a fourth experiment. We performed the test run on databases which contained 1 object up to 9 objects. This variation had no observable influence on the detection rate. Figure 15 shows the resulting average execution times.

To explore the tradeoff between slice logic usage and speed of the *Flex-SURF+* module, we also varied the amount of implemented *DEs*. Experimental results such as detection rate remained all the same (see first test run), except calculation time for determinants. Table 2 gives an overview on the implemented configurations and time consumption.

5. Pose Estimation Using Two Intelligent FPGA Cameras

Our second system which utilizes the *FlexSURF+* module is an embedded platform for augmented reality systems. The

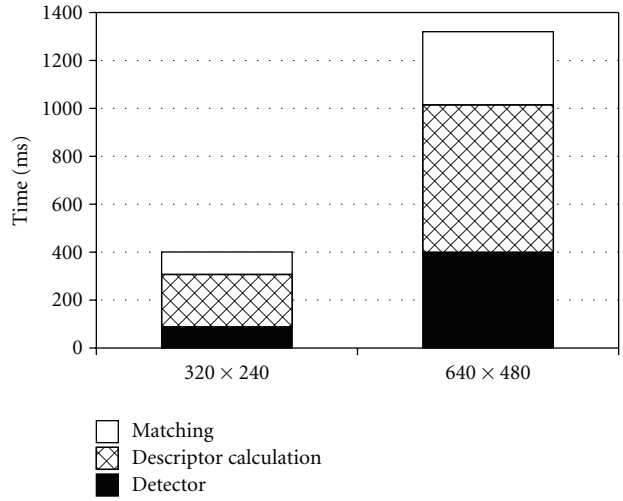


Figure 13: Average time consumption of the object recognitions main tasks for 640 × 480 pixel images.

Table 2: Different *Flex-SURF+* configurations.

Number of *DEs*	*DE lines*	*DEs* Per *DEline*	Virtex-5 Slices	Determinant calc. time (ms)
16	4	4	2243	134
32	4	8	4269	91
64	8	8	7577	70

system can recognize objects by natural features and determine their 3D pose, that is, their position and orientation. The platform contains a dual-core ARM microcontroller and is equipped with two FPGA-based cameras. The FPGAs in these cameras are smaller than in the system of the previous section and so the *Flex-SURF+* module's configuration is just half as big (4 × 8 *DEs*). The two cameras are connected to the dual-core ARM evaluation board via USB. Each FPGA of the cameras contains a system-on-chip with the *Flex-SURF+* module as a part of it. In this way the FPGAs are used to relieve the ARM platform from the time-consuming task of feature point detection.

An essential part of an AR system is an optical tracking component. Optical tracking is usually performed in two phases [28]. In the *initialization phase*, objects are detected and their 3D pose (6 degrees of freedom (DOF)) is calculated. The subsequent *tracking phase* keeps track of the objects from frame to frame. Each iteration of the tracking phase relies on the information from the initialization and from previous frames and can therefore be implemented efficiently (i.e., with interactive frame rates). However, the initialization phase cannot use any knowledge from previous frames and is considered to be very time-consuming in general (i.e., no interactive frame rates) [28]. The purpose of this work is to particularly accelerate the critical initialization phase as a basis for an efficient embedded real-time tracking system.

5.1. System Overview. The system consists of few hardware components which in combination are suited to be used in mobile devices, especially due to their low power consumption. An overview of the system's components is given in Figure 16.

First, the system captures image data with two cameras. They are each equipped with a grayscale image sensor (640×480 pixels), a Xilinx Spartan-3E-1600 FPGA, 64 MB of SDRAM, and a USB interface. Both camera FPGAs contain a *Flex-SURF+* module and perform the detector stage of the SURF algorithm for the respective image of the captured stereo pair. Hence, along with the captured image data, each FPGA camera module delivers a list of detected feature points, relieving the ARM CPUs from this task. The *Flex-SURF+* module is configured to contain 32 *DEs* (4 *DE lines* with 8 *DEs* each) and is clocked at the system bus clock rate of 50 MHz. An overview of the FPGA cameras SoC is depicted in Figure 17.

The SURF descriptor calculation stage, the subsequent feature matching and finally the 3D pose calculation stage are implemented in software, running on a *PandaBoard* evaluation platform. Its central component is a Texas Instruments OMAP4430 SoC, featuring two ARM Cortex-A9 CPU cores (1 GHz each). Furthermore, we use the available DDR2-SDRAM (1 GB), the SD card interface, and two USB interfaces, connecting the FPGA cameras to the board. For visualization we use a touch screen display with a resolution of 800 × 480 pixels which is directly connected to the board. As operating system we use Linux, which supports all involved peripherals.

5.2. Pose Calculation Algorithm. The algorithm starts with acquiring the image data from the cameras and then calculates the SURF algorithm. The most time-consuming part of

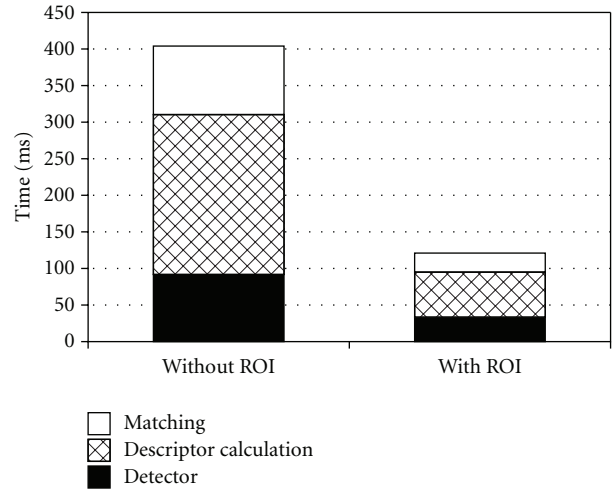


Figure 14: ROI-based average time consumption of the object recognition main tasks.

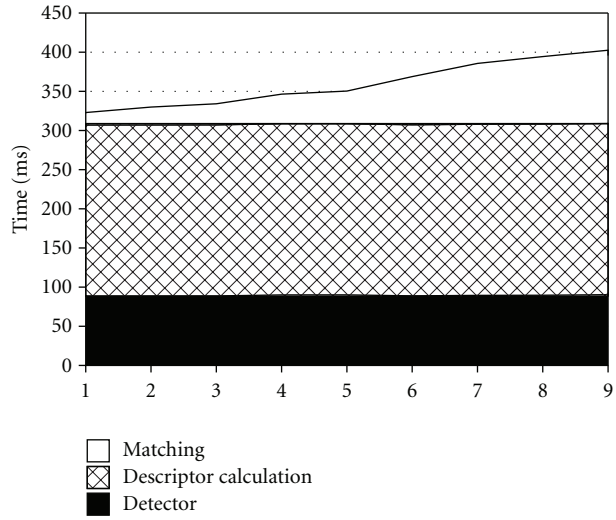


Figure 15: Average time consumption of the object recognitions main tasks for 320 × 240 pixel images at varying database sizes.

the SURF algorithm's feature detector is done on the FPGA of the camera, which is determinant calculation for the four filter stages at the highest image resolution. Subsequently the main system only needs to do the remaining part of SURF, which is basically the descriptor step. This is done in parallel on both ARM CPUs, each for one camera image.

After this, the recognition and pose calculation is done. The system uses a database of known objects assigned with a name, some meta data and a list of SURF descriptors, known to be on the surface of the object. For each descriptor, the 3D position in relation to the object-specific origin is also stored in this database. This 3D position is required for pose calculation after the matching step.

The software performs the following steps for each object stored in the database. In order to utilize the two ARM cores, the code is parallelized with OpenMP such that each CPU processes a part of the database.

(a) Matching of the Database Object against the Live Images. For each descriptor of the database object, the two descriptors of camera frame 1 with the lowest Euclidian distance are determined. As done in the object recognition system in Section 4.3, the ratio of the first and second best distance is used as an indicator for the uniqueness of the best match. For each unique matching pair, the index of the feature point in the database and a reference to the feature point in the live image are stored in a correspondence list, called the *db-cam1 list*. For camera 2, the same matching procedure is applied, but instead of comparing to all database features, only the ones from the results of camera 1 are used. The matching results of camera 2 are stored in a separate list, the *db-cam2 list*. The matching in this step is accelerated using an average nearest-neighbor algorithm based on randomized k-d trees as described in [29]. This algorithm allows the system to skip most of the feature comparisons. However, a tree structure

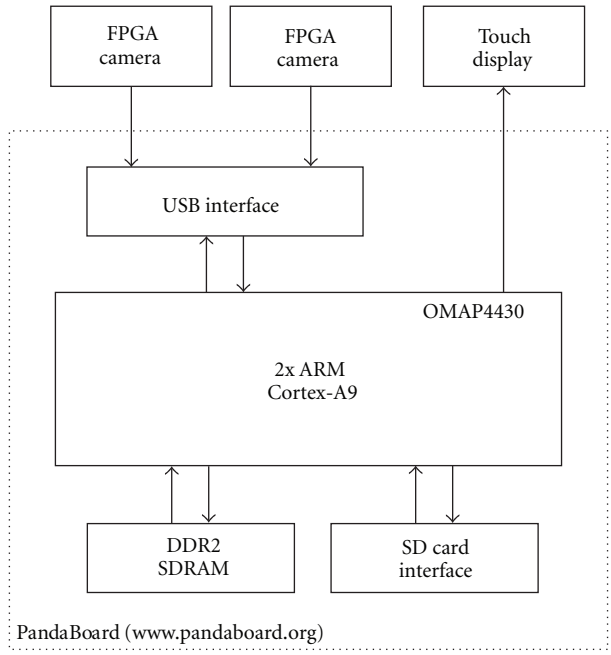


Figure 16: Structure of the pose estimation system.

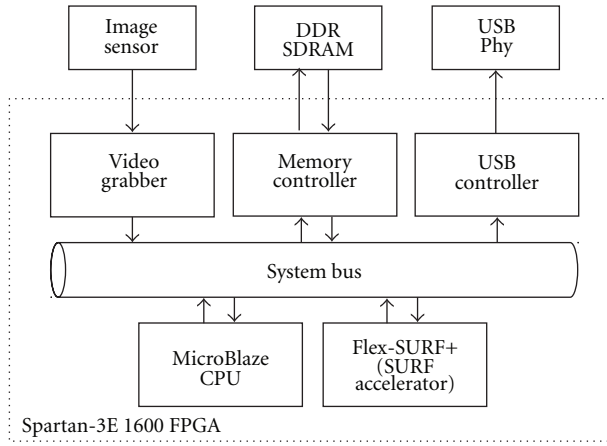


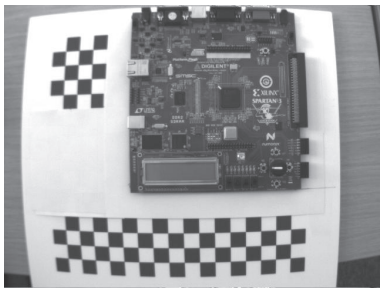
Figure 17: Structure of the SoC inside the FPGA cameras.

has to be built before matching can be done, but this is a very small overhead compared to the time cut down achieved by faster matching.

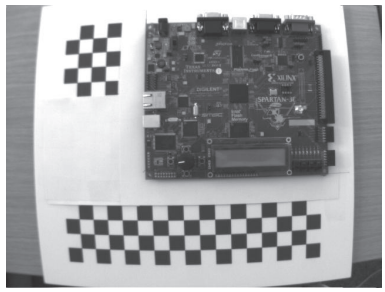
(b) Determination of Correspondences. In this step, the two lists, *db-cam1 list* and *db-cam2 list*, are merged. The algorithm searches for entries with identical database indices and stores them in a new list, the *db-cam1-cam2 list*.

(c) Reconstruction of 3D Points. For each element in the *db-cam1-cam2 list*, the two 2D elements of camera 1 and camera 2 are used to triangulate their respective 3D point in the world. Together with the 3D point from the database, a *db-world list* is created. Each of the pairs in this list is used as a support point for the final pose calculation. For the pose calculation, by theory, three such support points (which are not coliniar) are enough to calculate the pose. But, there can be outliers in this list. So, just to be sure, only objects which got four or more support points are processed in the following step, the others are considered as not present in the camera image.

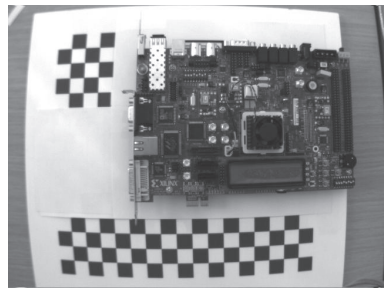
(d) Pose Calculation and Elimination of Outliers. As the set of support points, acquired in the previous step, may include outliers, the RANSAC algorithm is used as a coarse filter [30]. This algorithm basically tries different subsets (three random selected support points, in this case) and determines which of the tried subset's result would match the whole dataset best. With the cleaned set of support points the final pose is calculated with the method of least squares as described in [31]. The pose of the object in the world is returned as a 3×4 transformation matrix, which transforms the points in



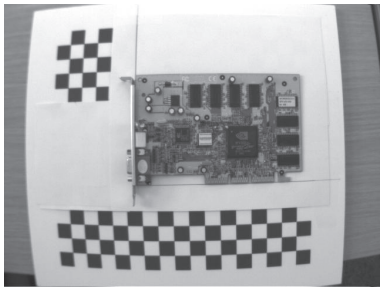
Spartan-3A evaluation board
(327 feature points)



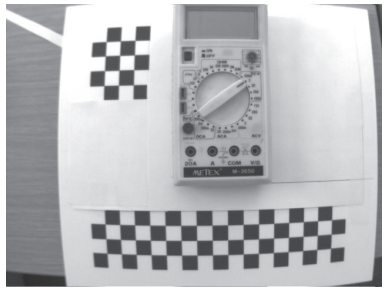
Spartan-3E evaluation board
(259 feature points)



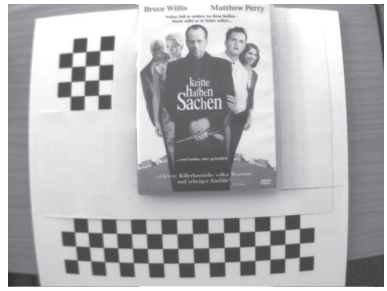
Virtex-5 evaluation board
(428 feature points)



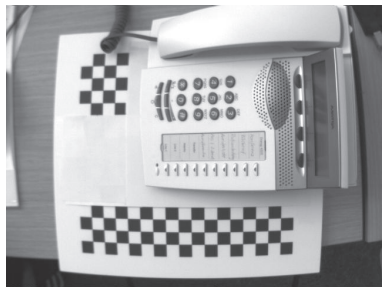
Graphics board
(426 feature points)



Multimeter
(293 feature points)



DVD
(480 feature points)



Telephone
(240 feature points)

Figure 18: Samples of images used for the database creation. The chessboard defines the 3D pose of the object and at the same time the origin of the object-related coordinate system. It is required in database creation step to filter and transform the detected 3D features.

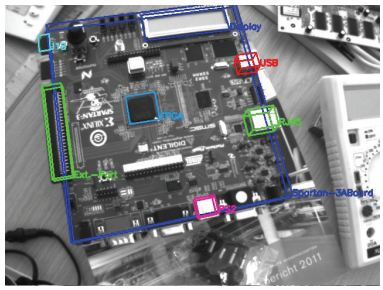
the database onto the points in the world as good as possible. The position of the object and its three rotation parameters can be extracted from the matrix easily.

5.3. Database Creation. The creation of the database is done with a tool which is able to record the SURF feature points of an object together with its 3D coordinates. Just like the live system, it uses two cameras watching the object. By matching the SURF feature points between the two camera pictures, the tool generates a list of corresponding 2D points and by triangulation calculates the related 3D points in world coordinates. Since not all of these calculated points belong to the new database object, they have to be filtered before they can be saved in the database.

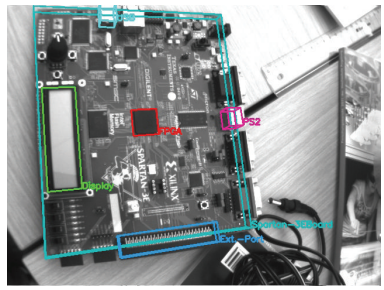
The filtration is done automatically based on the 3D coordinate of each point. As shown in the example screenshots in Figure 18, a chessboard pattern is used for (and only for) database creation. The enclosing chessboard pattern defines the exact position of the new object in the world. Since we know which size the object has, it is possible to filter the matched feature points automatically by means of their 3D position in the world. All feature points with 3D positions outside the bounding cuboid (also called *bounding box*) are discarded, and the remaining features are assigned to the object and saved in the database file. The 3D points have to be translated into the object-related coordinate system before storing to the database. Here, the chessboard pattern is needed again since it defines the origin of the object's coordinate system.

Also, it is possible to make an arbitrary number of stereo picture pairs per object in order to train different views of the object.

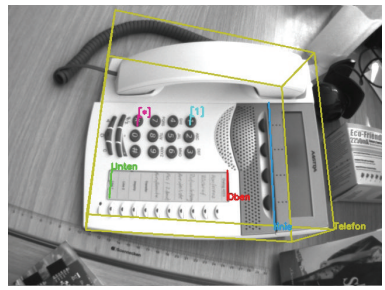
5.4. Experiments and Results. The database for the experiments was created with four image pairs per object, each with another light exposure direction. This is important, since



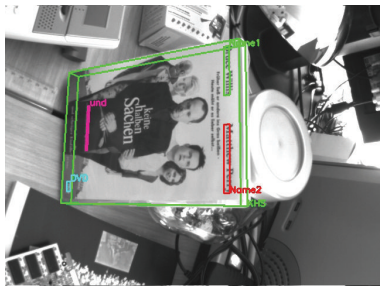
Spartan-3A evaluation board
category big



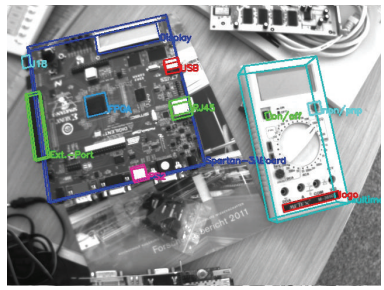
Spartan-3E evaluation board
category big



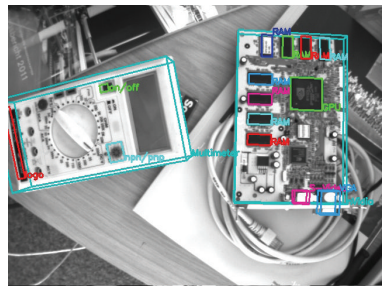
Telephone
category big



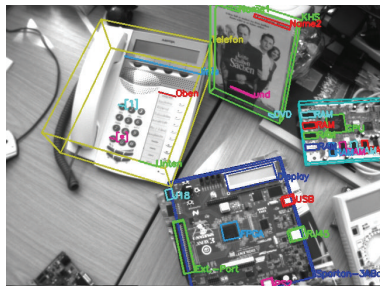
DVD
category big



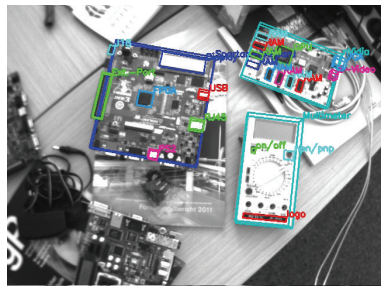
Spartan 3E, multimeter
category middle



Multimeter, graphics card
category middle



Telephone, Spartan 3A, DVD,
graphics card
category small



Spartan 3A, graphics card,
multimeter
category small

Figure 19: Screenshots of the test parcours: the pose of the estimated objects is drawn as bounding box into the images of the cameras. Additionally some smaller parts of the objects are marked as well.

the surfaces of our objects are not plain, so they look slightly different with other light exposure directions. The database consists of seven objects which are all shown on the example screenshots from the database creation in Figure 18.

To test the stability of the system, a parcours was created: a desk was filled with objects (trained and unknown) in random pose. Then the system was started and the cameras were moved slowly in random route over the desk. The route also covered different distances to the objects, so the objects appear in different sizes on the camera images. While the cameras were watching the objects, the system had to recognize the trained objects and calculate their pose autonomously. The results were recorded and evaluated afterwards. Example screenshots of the parcours are shown in Figure 19.

In the evaluation of the results, only objects which are fully visible were considered. An object is only considered successfully detected if its pose, position, and rotation in 3D space were determined correctly. The results are checked visually through the augmented bounding box and the other displayed elements of the object in the camera image (for examples see Figure 19). The criterion for a correct pose was a maximum deviation of 10 pixels for all observable reference points. So if just one of the reference points deviated more than 10 pixels, the pose was considered as uncorrect. This criterion is considered to be sufficient for our envisioned AR applications.

The results of the evaluation are shown in Figure 20. The objects were assigned into different categories (*big*, *middle* and *small size*) according to their current visual appearance on the camera picture (measured by their covered relative area, wich is an indicator for the distance to the camera). In the category *big size*, the objects covered between 26% and 50% of the image. In this group, 97% of object poses

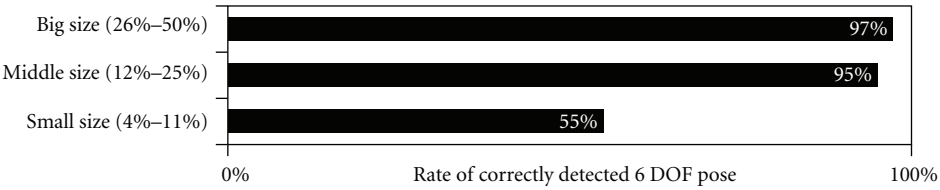


Figure 20: Detection rate of the pose of objects according to their distance (by covered area in image).

were determined correctly. This value means that the pose for most objects in this experiment is calculated correctly when the object distance to the camera is small enough. In category *middle size*, where objects covered 12% to 25% of the image, 95% were calculated correctly. This category contains objects which aren't directly in front of the camera, there is always space for a second object of same size left in the pictures. Also here, most of the objects were detected. In category *small size*, the correct pose calculation rate drops to 55%, but this is kind of expected behaviour here because each object covered only 4% to 11% of the image area. To reach a better result with the current system in this category, one could use cameras with higher image resolution, at the expense of higher hardware costs and longer computation times.

Altogether, there were 378 completely visible instances of known objects at 216 stereo image pairs. There were only 12 outliers at the pose estimation (correct object, but wrong pose) on fully visible objects. Most of the errors occurred at objects positioned close to the image border. And in all discovered cases, the outliers had a wrong rotation, but the position of the object center was near the expected one. There was only one case of false recognition, where a Spartan-3E board (see Figure 18) was detected at the position of the very similar Spartan-3A board (in addition to the correct one).

On average, 628 SURF feature points were calculated for each camera image. The utilization of the camera systems and the ARM CPU cores is quite the same. Using a pipeline concept enables the cameras to work in parallel to the main system on the next frame, so nearly all of the available processing power is used. The average computing time for each stereo image pair on the main system with the ARM CPUs was 1069 ms. Most of this time, around 70%, is spent to calculate the remaining part of the SURF algorithm. Around 20% is used to transfer data from the cameras to the main system and only 10% of the time is needed for matching and pose calculation with all database objects. The SoCs in the FPGAs of the cameras work in parallel with the ARM CPUs and need a constant time of 1014 ms per frame, including the time spent for the data transfer to the main system.

In further work, we plan to implement an additional tracking phase in our system, for example, by utilizing sparse optical flow algorithms as suggested in [28]. This way we expect to reach a performance that allows interactive frame rates.

6. Conclusion

We presented the *Flex-SURF+* hardware module which supports the feature detection stage of the SURF algorithm. It is highly customizable and allows a tradeoff between area and speed, depending on the application requirements. As application scenarios, we implemented two embedded systems for object recognition and pose estimation. The first system performs the task of object recognition in a single FPGA, with the *Flex-SURF+* module directly implemented in the system-on-chip within the FPGA. The second system is an embedded augmented reality platform which is equipped with two FPGA-based cameras. In this system, the *Flex-SURF+* modules relieve the ARM hardware from the time-consuming task of feature detection. Experimental results show that robust object recognition and pose estimation can be efficiently performed on mobile embedded hardware.

Acknowledgments

The authors would like to thank the following students, who were involved in the implementation of the systems: Andreas Becher, Markus Bihler, Albert Böswald, Thomas Britzelmeier, Christine Demharter, Stefan Durner, Andrei Ehrlich, Tobias Engelhard, Andreas Füger, Christian Hilgers, Frederik Hinze, Tobias Köglsberger, Werner Landsperger, Moritz Lessmann, Markus Litzel, Matthias Maiershofer, Markus von Mengden, Michael Morscher, Christoph Pöll, Matthias Pohl, Christopher Proske, Markus Rissmann, Matthias Ruhland, Elmar Scheier, Christoph Schwarz, and Christopher Simon. This work has been supported by the German Federal Ministry of Education and Research (BMBF), Grant no. 17N3709.

References

[1] H. M. Atiq, U. Farooq, R. Ibrahim, O. Khalid, and M. Amar, "Vehicle detection and shape recognition using optical sensors: a review," in *Proceedings of the 2nd International Conference on Machine Learning and Computing (ICMLC '10)*, pp. 223–227, February 2010.

[2] C. Hermes, J. Einhaus, M. Hahn, C. Wöhler, and F. Kummert, "Vehicle tracking and motion prediction in complex urban scenarios," in *Proceedings of the IEEE Intelligent Vehicles Symposium (IV '10)*, pp. 26–33, June 2010.

[3] D. M. Jang and M. Turk, "Car-Rec: a real time car recognition system," in *Proceedings of the IEEE Workshop on Applications of Computer Vision (WACV '11)*, pp. 599–605, January 2011.

[4] B. Höferlin and K. Zimmermann, "Towards reliable traffic sign recognition," in *Proceedings of the IEEE Intelligent Vehicles Symposium*, pp. 324–329, June 2009.

[5] J. F. Liu, Y. F. Su, M. K. Ko, and P. N. Yu, "Development of a vision-based driver assistance system with lane departure warning and forward collision warning functions," in *Proceedings of the Digital Image Computing: Techniques and Applications (DICTA '08)*, pp. 480–485, December 2008.

[6] S. J. Henderson and S. Feiner, "Evaluating the benefits of augmented reality for task localization in maintenance of an armored personnel carrier turret," in *Proceedings of the 8th IEEE International Symposium on Mixed and Augmented Reality (ISMAR '09)*, pp. 135–144, Orlando, Fla, USA, October 2009.

[7] B. Schwerdtfeger, R. Reif, W. A. Günthner et al., "Pick-by-vision: a first stress test," in *Proceedings of the 10th IEEE and ACM International Symposium on Mixed and Augmented Reality*, pp. 115–124, October 2009.

[8] G. Amato, F. Falchi, and P. Bolettieri, "Recognizing landmarks using automated classification techniques: an evaluation of various visual features," in *Proceedings of the 2nd International Conferences on Advances in Multimedia (MMEDIA '10)*, pp. 78–83, June 2010.

[9] H. Bay, A. Ess, T. Tuytelaars, and L. Van Gool, "Speeded-up robust features (SURF)," *Computer Vision and Image Understanding*, vol. 110, no. 3, pp. 346–359, 2008.

[10] C. Harris and M. Stevens, "A Combined corner and edge detector," in *Proceedings of the 4th Alvey Vision Conference*, pp. 147–151, Manchester, UK, 1988.

[11] D. G. Lowe, "Distinctive image features from scale-invariant keypoints," *International Journal of Computer Vision*, vol. 60, no. 2, pp. 91–110, 2004.

[12] X. Wu, Z. Shi, and Y. Zhong, "Detailed analysis and evaluation of keypoint extraction methods," in *Proceedings of the International Conference on Computer Application and System Modeling (ICCASM '10)*, vol. 2, pp. 562–566, October 2010.

[13] M. Schaeferling and G. Kiefer, "Flex-SURF: a flexible architecture for FPGA-based robust feature extraction for optical tracking systems," in *Proceedings of the International Conference on Reconfigurable Computing and FPGAs (ReConFig '10)*, pp. 458–463, December 2010.

[14] M. Schaeferling and G. Kiefer, "Object recognition on a chip: a complete SURF-based system on a single FPGA," in *Proceedings of the International Conference on Reconfigurable Computing and FPGAs (ReConFig '11)*, pp. 49–54, 2011.

[15] K. Murphy, A. Torralba, D. Eaton, and W. Freeman, "Object detection and localization using local and global features," in *Toward Category-Level Object Recognition*, vol. 4170, pp. 382–400, 2006.

[16] A. Collet, M. Martinez, and S. S. Srinivasa, "The MOPED framework: object recognition and pose estimation for manipulation," *The International Journal of Robotics Research*, vol. 30, no. 10, pp. 1284–1306, 2011.

[17] F. Ren, J. Huang, R. Jiang, and R. Klette, "General traffic sign recognition by feature matching," in *Proceedings of the 24th International Conference Image and Vision Computing New Zealand (IVCNZ '09)*, pp. 409–414, November 2009.

[18] D. Gossow, J. Pellenz, and D. Paulus, "Danger sign detection using color histograms and SURF matching," in *Proceedings of the IEEE International Workshop on Safety, Security and Rescue Robotics (SSRR '08)*, pp. 13–18, November 2008.

[19] B. Besbes, A. Apatean, A. Rogozan, and A. Bensrhair, "Combining SURF-based local and global features for road obstacle recognition in far infrared images," in *Proceedings of the 13th International IEEE Conference on Intelligent Transportation Systems (ITSC '10)*, pp. 1869–1874, September 2010.

[20] N. Cornells and L. V. Gool, "Fast scale invariant feature detection and matching on programmable graphics hardware," in *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops (CVPR '08)*, pp. 1–8, Anchorage, Alaska, USA, June 2008.

[21] A. Schulz, F. Jung, S. Hartte et al., "CUDA SURF—a real-time implementation for SURF," http://www.d2.mpi-inf.mpg.de/surf.

[22] P. Azad, T. Asfour, and R. Dillmann, "Combining Harris interest points and the SIFT descriptor for fast scale-invariant object recognition," in *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS '09)*, pp. 4275–4280, October 2009.

[23] W. C. Chen, Y. Xiong, J. Gao, N. Gelfand, and R. Grzeszczuk, "Efficient extraction of robust image features on mobile devices," in *Proceedings of the 6th IEEE and ACM International Symposium on Mixed and Augmented Reality (ISMAR '07)*, pp. 287–288, November 2007.

[24] J. Zhang, H. Sang, and X. Shen, "Overview of approaches for accelerating scale invariant feature detection algorithm," in *Proceedings of the International Conference on Electric Information and Control Engineering (ICEICE '11)*, pp. 585–589, April 2011.

[25] D. Bouris, A. Nikitakis, and I. Papaefstathiou, "Fast and efficient FPGA-based feature detection employing the SURF algorithm," in *Proceedings of the 18th IEEE International Symposium on Field-Programmable Custom Computing Machines (FCCM '10)*, pp. 3–10, May 2010.

[26] J. Šváb, T. Krajník, J. Faigl, and L. Přeučil, "FPGA based speeded up robust features," in *Proceedings of the IEEE International Conference on Technologies for Practical Robot Applications (TePRA '09)*, pp. 35–41, Woburn, Mass, USA, November 2009.

[27] C. Evans, "Notes on the OpenSURF Library," Tech. Rep. CSTR-09-001, University of Bristol, 2009.

[28] V. Lepetit and P. Fua, "Monocular model-based 3D tracking of rigid objects: a survey," *Foundations and Trends in Computer Graphics and Vision*, vol. 1, no. 1, 2005.

[29] M. Muja and D. G. Lowe, "Fast approximate nearest neighbors with automatic algorithm configuration," in *Proceedings of the 4th International Conference on Computer Vision Theory and Applications (VISAPP '09)*, pp. 331–340, February 2009.

[30] M. A. Fischler and R. C. Bolles, "Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography," *Communications of the ACM*, vol. 24, no. 6, pp. 381–395, 1981.

[31] K. S. Arun, T. S. Huang, and S. D. Blostein, "Least-squares fitting of two 3-D point sets," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 9, no. 5, pp. 698–700, 1987.