FPGA-based module for SURF extraction

Tomáš Krajník · Jan Šváb · Sol Pedre · Petr Čížek · Libor Přeučil

Received: date / Accepted: date

Abstract We present a complete hardware and software solution of an FPGA-based computer vision embedded module capable of carrying out SURF image features extraction algorithm. Aside from image analysis, the module embeds a Linux distribution that allows to run programs specifically tailored for particular applications. The module is based on a Virtex-5 FXT FPGA which features powerful configurable logic and an embedded PowerPC processor. We describe the module hardware as well as the custom FPGA image processing cores that implement the algorithm's most computationally expensive process, the interest point detection. The module's overall performance is evaluated and compared to CPU and GPU based solutions. Results show that the embedded module achieves comparable disctinctiveness to the SURF software implementation running in a standard CPU while being faster and consuming significantly less power and space. Thus, it allows to use the SURF algorithm in applications with power and spatial constraints, such as autonomous navigation of small mobile robots.

The EU supported this work by FP7 programme projects ICT-600623 'STRANDS' and ICT-216240 'REPLICATOR'. The Czech Ministry of Education funded the work by projects 7AMB12AR022 and 7E08006 and Argentinean Ministry of Science supported this work by project ARC/11/11.

Tomáš Krajník
Lincoln Centre for Autonomous Systems,
School of Computer Science, University of Lincoln
E-mail: tkrajnik@lincoln.ac.uk

Jan Šváb, Petr Čížek, Libor Přeučil, Tomáš Krajník,
Department of Cybernetics, Faculty of Electrical Engineering,
Czech Technical University in Prague

Sol Pedre
División de Robótica CAREM, Centro Atómico Bariloche,
Comisión Nacional de Energía Atómica, Argentina.

1 Introduction

Low prices of digital cameras together with the increasing computational power of nowadays computers are causing increasing popularity of computer vision methods. These methods have been successfully employed in the tasks of reactive navigation of autonomous vehicles [34], object recognition [19], three–dimensional reconstruction [24], and efficient mapping, localization and exploration [25,11]. Many of these methods are based on feature extraction algorithms [18] like Scale Invariant Feature Transformation - SIFT [31], Gradient Location and Orientation Histogram - GLOH [22], Local Energy based Shape Histogram-LESH [27] or Center Surround Extremas-CENSURE [1]. However, these algorithms are computationally demanding and represent a significant bottleneck in many computer vision systems, forcing researchers to focus on improvement of their speed.

Luckily, the local image feature extractors are easy to parallelize because they perform the same operations on different sets of data. This property makes these algorithms ideal candidates for implementation on parallel architectures, like graphics processing units (GPU) and field programmable gate arrays (FPGA). One of the most popular local feature extraction algorithms is the Speeded Up Robust Features (SURF) [5]. Although its graphics processing unit implementation, the GPU-SURF [10] achieves real-time performance, it requires to use an entire PC-based system, which is infeasible in applications which impose restrictions on hardware power consumption, dimensions or weight. Such applications include a wide variety of embedded systems ranging from robotic navigation [32,35], μUAV autopilot systems [36,26,2], micro satellite sensing [37,15], to mobile visual search in commercial cell phones [8].

1.1 Main contributions

We present a standalone FPGA-based embedded module capable of real-time extraction of the Speeded-Up Robust Features (SURF) from its camera image. The module processes roughly ten 1024×768 pixel images per second[1], consumes approximately 6 W and occupies significantly less space than a GPU-based system with a similar performance. Despite of it's small size and low power consumption, the module's performance is comparable to state-of-the-art SURF implementations. To demonstrate the module's suitability for applications with spatial and energy constraints, we show that it can be used for visual navigation of small mobile robots.

The main contributions of this work are:

- As far as we know, it is the first complete standalone embedded module for the SURF algorithm, that includes all steps from image capture to data transmission while satisfying real time constraints.
- The solution includes a customized baseboard with fast SSRAM, data storage and other interfaces specific for machine vision applications.
- The inclusion of an embedded processor running Linux OS that enables any user to program, compile and run particular applications in the embedded module. In this manner, real-time standalone embedded developments for many SURF-based applications can be achieved with little effort, making the embedded module truly reusable.

Additionally, the work presents the complete FPGA hardware/software co-design, which gives flexibility to also add, remove or modify hardware modules for further customization.

2 Related Work

Since the Moravec operator [23] was proposed, feature detection has been a growing field in computer vision, being the underlying algorithms in many computer vision systems. The first widely used feature detection algorithm was the Harris corner detection [13], which is more robust (to noise, intensity and rotation changes) than the Moravec algorithm.

Several other feature detection algorithms have been proposed. One of the most robust approaches is the Scale Invariant Feature Transform [31] that apart from detecting features proposes a descriptor that is invariant to scale, rotation and illumination. Fully implemented SIFT has a high computational cost, which has led to the proposal of its optimized variants [16,12], GPU [38], FPGA [9,39] and ASIC [14] designs.

[1] Considering around 140 descriptors per image.

From the FPGA SIFT implementations, one of the most complete and closely related to our work is described by Bonato et al [6]. They present a complete on-chip implementation of the SIFT algorithm, using a Stratix II FPGA with a NIOS II embedded processor. Using a development board with four CMOS cameras they have tested feasibility of their implementation for Simultaneous Localization and Mapping (SLAM). Although they achieved a good performance of 33 ms per 320×240 frame for feature detection, the NIOS II software implementation for descriptor creation takes 11.7 ms per detected feature. This restricts the possible embedded applications of this solution, since to keep real-time performance, only a few features per frame can be calculated.

Another approach to achieve robust features with low computational costs is to propose different detection and description methods. Bay et al [5] proposed the Speeded Up Robust Features (SURF) algorithm that has since become widely used. The SURF detector is based on a basic approximation of the Hessian blob detector, relying on integral images to reduce the computation time. The descriptor uses a distribution of Haar-wavelet responses within the interest point neighborhood, exploiting the use of integral images to achieve higher speed. Moreover, the SURF descriptor dimension can be reduced to 32, lowering not only the time for descriptor computation, but also for subsequent matching. Comparisons of SIFT and SURF suggest that although SIFT features perform better than SURF features, the gain in computational cost outweighs this for many applications [4]. However, SURF is still computationally demanding and has been implemented on parallel architectures like GPUs [10] and FGPAs [33,7, 28].

The first published FPGA acceleration of SURF is [33], that is the basis for our present work. Three other FPGA implementations of SURF can be found in literature. Bouris et al [7] present a programmable logic implementation of the detection and orientation assignment in a Virtex5 FPGA. They achieve 56 fps in 640×480 pixel images, but these results do not include the descriptor calculation process and the number of processed features in the orientation assignment phase is unclear. Schaeferling et al proposed Flex-Surf [28], a co-designed implementation featuring a special tile memory access to reduce memory bandwidth, one of the bottlenecks of SURF. In their approach, the detector is implemented in programmable logic, while the descriptor is implemented in software on a PowerPC embedded processor. In [29] they present the use of Flex-Surf for object recognition tasks. Finally, Battezzati et al [3] presented a short paper on an SURF architec-

	Flex-Surf	Bour-Surf	SV-Surf
Frame size	457×630	640×480	1024×768
Octaves, intervals	3, 4	3, 4	2, 4
Detector speed [ms]	759	7.55	102
Descriptor[ms/surf]	1.4	N/A	0.7
Virtex version	5FX70T	5FX130T	5FX70T
Slice Registers	2656	11 457	16 548
Slice LUTs	5450	13 272	15 271
Block RAMs	0	271	86
DSPs	6	50	40
Clock [MHz]	100	200	100
Consumption [W]	unknown	20	6

Table 1: FPGA SURF implementation comparison

ture for industrial applications. From this paper it can be concluded that massive parallelization of the detection phase in a larger FPGA might boost the SURF algorithm speed dramatically. Table 1 shows the area consumption and performance characteristics of Flex-, Bouris- and SV- SURF implementations. All of these FPGA SURF versions are implemented in the same FPGA Family as our proposal, so speed and performance comparisons can be done. However, each of the implementations chooses a different way to deal with descriptor calculation. The Flex- and SV-SURF implement the descriptor on the Xilinx PPC in fixed and floating point respectively, while the Bouris descriptor is wired in FPGA logic. Moreover, the SV-SURF descriptor omits the orientation assignment phase and the Bouris implements only the orientation assignment. Therefore, the descriptor speed in Table 1 is informative only and should not be used as performance measure.

The rest of the Table 1 contains detector implementation details, i.e. only the IP cores relevant to detector calculation are taken into account when estimating FPGA area consumption. Our implementation is 16 times faster than the Flex-SURF, but occupies more of the FPGA circuitry. When comparing to Bouris-SURF, we can see that our detector is slower. However, our module processes larger images, implements descriptor calculation and consumes less power. This follows the known thumbnail rule that better throughput can be achieved through further parallelization at the cost of area and power consumption.

Note that both Flex- and Bouris-SURF methods run on multi purpose development boards. The Bouris-SURF implementation is tested by loading a reduced set of images in the Flash memory and therefore its deployment in real world scenarios is rather limited. The FPGA-based object recognition system [30] includes a complete processing from image grabbing to information output. However, the presented solution is targeted for a particular application, which does not impose a strict real-time restictions. Due to this, it processes lower (320×240) resolution images with slightly lower framerates compared to our implementation. This would prevent the usage of the module in scenarios like visual based mobile robot navigation.

Contrary to the aforementioned solutions, our work presents a complete standalone low power module that can be easily adapted for applications that use Speeded Up Robust Features as a core algorithm. We have not only accelerated the SURF algorithm by implementing several IP cores in FPGA logic, but also built a hardware baseboard especially customized for machine vision applications. This baseboard includes the required power supplies, a camera interface, extra SSRAM memory, a SD card slot and SATA connectors for chaining several modules together for multicamera applications. Thus, the presented low power module can be used in real life applications. Moreover, we have adapted a Linux distribution so the applications using the extracted features can be deployed and tested comfortably.

3 SURF algorithm

The SURF algorithm [5] takes a grayscale image as an input and returns a set of interest point locations along with a set of their descriptors, which are partially invariant to viewpoint and illumination changes. The algorithm relies on estimation of Gaussian and Haar wavelet filter responses by box filters. It processes the image in four consecutive stages. First, the method calculates a so-called integral image $\mathbf{I_\Sigma}$, which is used to speed up the following stages of the algorithm. Then, points of interest are identified by means of "Fast Hessian" blob detector, which pinpoints local brightness extrema. In the third stage, each interest point is assigned an orientation based on the direction of a highest brightness gradient of its neighbourhood. Finally, the method calculates a multidimensional descriptor from luminance gradients around the interest point.

3.1 Integral image calculation

The integral image ($\mathbf{I_\Sigma}$) is calculated from the original image $\mathbf{I}$ by means of the following equation :

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

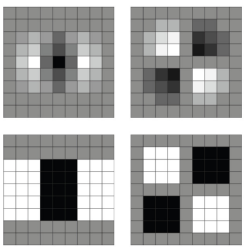
The integral image allows to calculate the response of a box filter of any size by means of three additions. Exploiting this fact in the following stages of the SURF algorithm leads to a significant speedup of the entire method.

3.2 Interest point detector

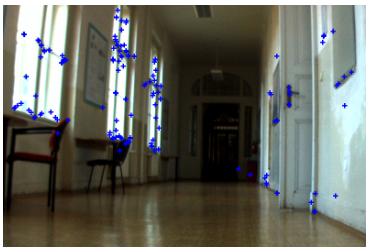
The purpose of the second stage is to identify interest points, which would retain their positions in the perceived scene despite changes in viewpoint and illumination. These points are located by finding local maxima of image Hessian determinants approximated by

$$\mathcal{H}(x,y,\sigma) = \begin{vmatrix} D_{xx}(x,y,\sigma) & D_{xy}(x,y,\sigma) \\ D_{xy}(x,y,\sigma) & D_{yy}(x,y,\sigma) \end{vmatrix}, \quad (1)$$

where $D_{xx}(x,y,\sigma)$ represents a convolution with an approximated second order derivative of a two dimensional Gaussian of variance σ. To achieve scale invariance, the algorithm uses filters with multiple sizes, creating a three dimensional space of determinant results, called a "scale space". The scale is quantized to "octaves", where an octave refers to a series of "intervals" covering a doubling of scale. The SURF detector approximates the Gaussian kernels with box filters (see Figure 1a), which are calculated much faster from the integral image. Once the scale space is calculated, its



(a) Gaussian and box filter kernels.



(b) Indoor scene with identified SURF points

Fig. 1: SURF detector principle and results

local maxima are found and those which pass a preselected threshold indicate position of the interest points. A typical result of the SURF detector is shown on Figure 1b.

3.3 Orientation assignment

The interest point is then assigned a "dominant direction", which is calculated from the responses of Haar wavelet filters centered around the interest point. Since the presented module is primarily aimed for applications in mobile robotics domain, we assume that its camera will be oriented horizontally. In this case, the camera rotation along its optical axis can be obtained by accelerometric measurements and rotation invariance is not needed. Therefore, our module does not implement the orientation assignment step.

3.4 Descriptor calculation

The final stage establishes a SURF descriptor from a square shaped interest point neighborhood. This neighborhood is divided in 16 equal sub-squares, which are regularly sampled by Haar wavelet filters. Horizontal d_x and vertical d_y Haar wavelet responses within each sub-square are summed to form a vector consisting of $\sum d_x, \sum d_y, \sum |d_x|, \sum |d_y|$, which describes the particular sub-square. The vectors of all subsquares are chained to form a 64 dimensional SURF descriptor, which is finally normalized.

A sign of the Hessian matrix trace calculated during the detection step is included in the descriptor. Since interest points are usually found on blob-like structures, this sign distinguishes light blobs on dark background and vice versa.

4 Hardware overview

The designed module is based on Avnet AES-MMP-V5FXT70-G MiniModule Plus. This COTS module offers Xilinx Virtex5 FXT FPGA, 64MB DDR2 SDRAM, 32MB flash, USB 2.0, 1G Ethernet and two 120-pin expansion connectors for baseboard attachment. The baseboard has to provide a module with all required power supplies. Thus the core of our hardware solution lies in a custom-designed baseboard for this module, tailored specifically for computer vision applications. Figure 2 shows main hardware features of our module.

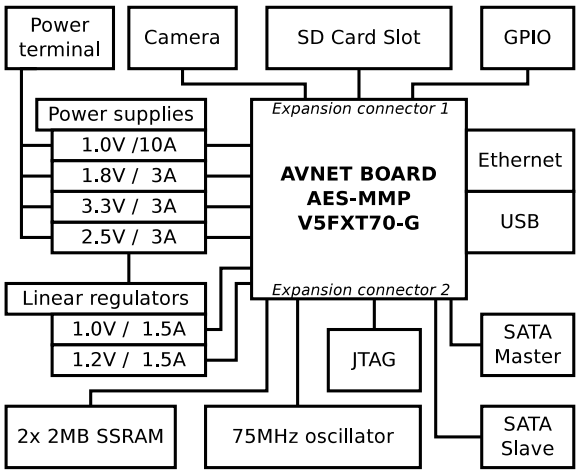


Fig. 2: The module hardware block diagram.

4.1 Generic baseboard features

The board power supplies, expansion connector, SD slot, etc. - these elements belong to a group of features that is useful in almost every embedded system design. However, our baseboard also contains two SATA connectors. A possible application is to connect a harddrive to one of them and gain a noticeably large storage for the embedded project. But the role of the second is less obvious – it is wired as a slave port, which allows to daisy-chain two or more of our modules using ordinary SATA cable. This feature provides a fast and low-power link useful e.g. for stereovision applications. The two stand-offs are designed for a FPGA heat

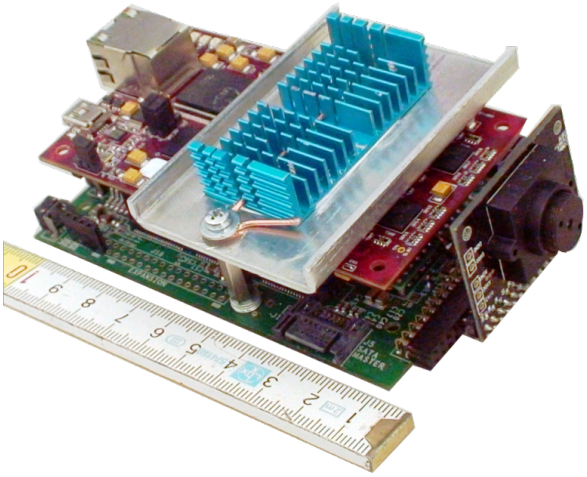


Fig. 3: The module with an attached camera.

sink, which also helps to hold the mini-module firmly attached to the baseboard. Since the module has been used for other image processing methods, which might have a higher power consumption, a large heatsink has been placed on top of the FPGA, see Picture 19. However, this heatsink is not needed when the FPGA is running the method presented in this article.

4.2 Computer-vision specific baseboard features

The baseboard provides several features that are tailored specifically for the computer vision applications. The camera connector, located in front of the module, contains 25 general purpose IOs as well as filtered 3.3V and 2.5V power supplies. Currently, it is used to communicate with the OmniVision OV9653 CMOS sensor. The module resolution is 1300×1028 and maximal frame rate is 120 fps. In current configuration, the camera is set to 30 fps, 1024×768 pixels and YUV 4:2:2 output format.

The GS816032BGT SSRAM has been included to support demanding memory requirements. It allows the user to store 4MB of data accessible through a 32bit 200MHz synchronous interface. The most significant advantage of this memory over the DDR2 SDRAM is that its contents can be accessed in an arbitrary order without any bandwidth penalty.

5 Hardware/Software co-design solution

The algorithm to generate the SURF descriptors follows the original SURF description as closely as possible. The most time-consuming part of SURF (the interest point detection) has been selected for hardware implementation using FPGA logic. The SURF descriptors are then calculated by software running on the PowerPC-440 embedded processor incorporated in the Virtex-5 FXT. The BusyBox-based Linux distribution has been created to make the module easily usable. The custom hardware-based image processing pipelines are available to the user thanks to custom-designed kernel module which controls their operation.

Since the Fast-Hessian detector is computed in hardware, the determinant calculation is done in integer arithmetic with a limited precision for a specific number of octaves and scale intervals and limited image size. Current image size limit is 1024×1024 pixels and our IP cores are designed to calculate determinants in two octaves and four intervals per octave.

5.1 HW/SW partitioning

To decide which phase of the SURF algorithm to implement in hardware, we have considered the potential speed gain, complexity of implementation and dependencies of the individual steps. Since the desired module application scenario is visual based navigation of small mobile robots, the SURF detector and descriptor performance has been tested on a nettop PC[2], which might be carried even by a small robot. The time to extract n SURF features from one 1024×768 image takes $5200 + 1.4n$ milliseconds for the PC's CPU and $105 + 0.1n$ milliseconds for the PC's GPU, see Table 2. This result suggests, that if the number of features required by the intended module's application scenarios would not be too high, the most time consuming phase of the algorithm is the interest point detection. The visual navigation algorithm [17] which is intended to be

[2] For profiling, we used a NT330-i with Intel Atom 330 1.6GHz, 1GB RAM and nVidia ION graphics card

used with the module is reported to use about 150 landmarks. In this case, more than 97% of the CPU time is spend with interest point detection. Moreover, the speed of the GPU SURF implementation indicates that parallelizing the detector achieves a higher speedup factor than parallelizing the descriptor. Considering the fact, that migrating the descriptor calculation to hardware is at least as complex as migrating the detector, we have decided to run the descriptor calculation simply on the module's PPC.

5.2 FPGA configuration

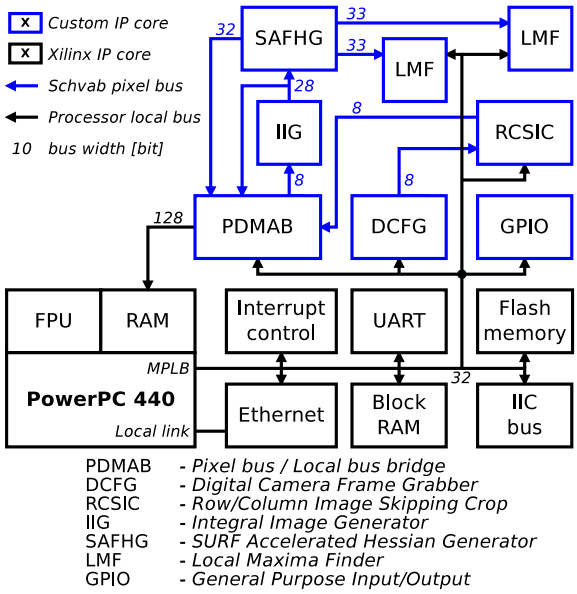


Fig. 4: Current overall PSoC architecture

The main role in the FPGA part of our design is played by several custom-designed reusable blocks described in following subsections. All of them conform to Xilinx XPS IP core specification which allows their easy integration into arbitrary PSoC design. The most commonly-used IP core is Schvab Pixel Bus (SPB), a single-master, multi-slave unidirectional bus with zero latency and a tiny logic overhead. It is basically an unified interface between all data processing blocks.

Another important block is the (`PDMAB`) bridge. It allows user to feed data to/consume data from the SPB. Current version provides three SPB slaves and one master. This block is connected through its 128-bit PLB master to Xilinx crossbar switch and allows convenient direct access to main module memory (64MB DDR2).

Our system is based on a straightforward pipeline-like structure, that can be reconfigured for different purposes. For example, using the SPB to connect the master port of the (`PDMAB`) to one of its slave ports creates a simple DMA engine. The following subsections describe image processing blocks - these can be daisy chained in the XPS using SPB to obtain an "image processing chain". The chain has to have a source of image data and a sink [3].

Our current design features two processing chains. The first one (`DCFG` → `RCSIC` → `PDMAB`) can grab, subsample and crop image frames from the attached camera and store them into the main memory. The second one (`PDMAB` → `IIG` → `SAFHG` → `LMF`) can take an image from main memory, calculate its integral image, calculate Fast-Hessian responses and search them for local maxima. As you can see from Figure 4, the second chain's two branches return to `PDMAB`. The first branch serves to store integral image data from currently processed image into main memory for descriptor calculation and the second (branch from `SAFHG`) serves for possible debugging of `SAFHG` - it can store Fast-Hessian results into main memory for later analysis.

Some algorithm parameters – such as number of octaves – are given by the system and IP-core architecture. If the user would wish to change these, he would need to modify the FPGA design only on the system level withouth altering the IP cores.

For example the addition of two more octaves would require branching the second processing chain, subsampling the data stream (`RCSIC`) and addition of instances of `SAFHG` and `LMF`. This addition wouldn't slow down the detector (since the original chain would still be operating in parallel on 4× larger image) but it would cost a significant amount of FPGA resources.

The run-time parameters of the image processing (such as sizes, thresholds, subsampling, memory addresses) are controllable using ordinary PLB-attached register banks on appropriate processing blocks. One of important characteristics of our solution is that thanks to the DMA capabilities the CPU doesn't have to touch the image data until it wants to calculate the descriptor or send the image through the Ethernet.

5.3 Digital Camera Frame Grabber

The Digital Camera Frame Grabber (`DCFG`) core is designed to adapt a CMOS camera interface to the SPB. This core not only transfers data, but also generates synchronization signals for the SPB from the vertical and horizontal timing signals of the camera. So far we have tested this IP with the OmniVision OV9653 cam-

[3] Actually it may have more sinks - the SPB is multi-slave.

era chip, which provides 16 bits per image pixel in YUV 4:2:2 format over an 8-bit data bus.

5.4 Row Column Skipping Image Crop

The Row Column Skipping Image Crop (`RCSIC`) core is capable of subsampling and/or cropping images going through the SPB.

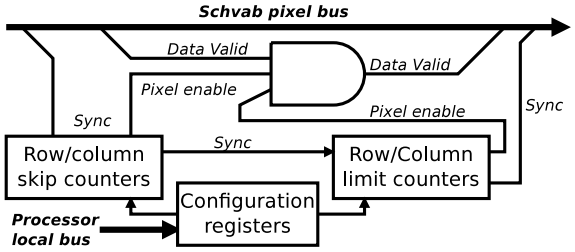


Fig. 5: The `RCSIC` core scheme.

The image subsampling and cropping is performed by suppressing a preset number of pixels. Due to simplicity of this core operation, it does not introduce any delay to the SPB. This core has been added to the design to allow greater flexibility of the module, because some applications do not require a full scale image or process only part of it. For example, the visual mobile robot navigation presented in [17] requires only the upper half of the image. However, in our experiments, we did not use this core to alter the image. The operational parameters of this core are run-time modifiable using configuration registers.

5.5 Integral Image Generator

The Integral Image Generator (`IIG`) core is responsible for the integral image generation.

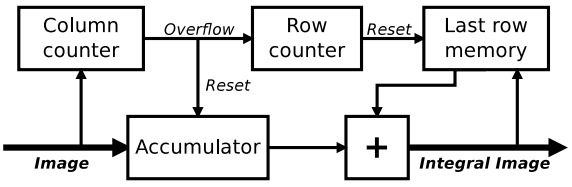


Fig. 6: The `IIG` core scheme.

The structure and principle of operation of this core can be seen in Figure 6. As aforementioned the resulting integral image is sent not only to the Fast Hessian Generator, but also to the main memory for later reuse by the descriptor calculator, see Figure. 4.

5.6 SURF Accelerator - Fast-Hessian Generator

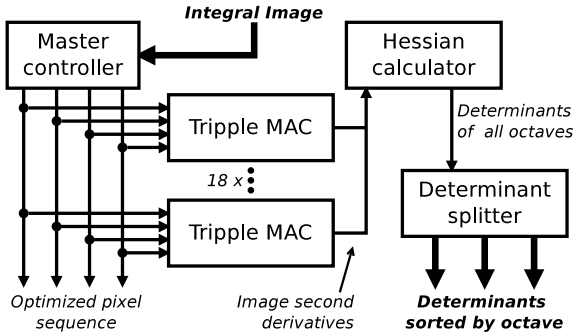


Fig. 7: The `SAFHG` core block diagram.

The SURF Accelerator - Fast-Hessian Generator IP core (`SAFHG`) (Fig. 7) is a key component of SURF detector acceleration. It calculates the Fast-Hessian responses from the integral image and forms the entire scale space used by the detector. An important factor influencing the performance of the determinant calculation is the optimization of memory access, which is performed by the `MasterController` block. This block outputs integral image data in a parallel optimized order suitable for image second order derivatives calculations in the `TripleMAC` blocks. Afterwards, the final determinant values are calculated by the `HessianCalc` block. The resulting stream of determinants, which contains mixed values from two scale space octaves, is split by the Determinant Block Triple Splitter (`DBTS`) block. The following three subsections explain the key components of the `SAFHG` core in more detail.

5.7 Master Controller

This block produces optimally ordered stream of integral image data suitable for image derivatives calculation by the `Triple MAC` blocks. `Master controller` iterates through the image data in a 2×2 pixel steps – this is step referred to as a "determinant block". One such "determinant block" requires calculation of 18 determinants (2×2 pixels×4 intervals for the first octave and additional 2 for the second octave). If we overlay all box filter masks (for integral image) for all determinants in one block and count how many times each pixel is read we can count that one determinant block

calculation requires reading of 576 integral image values on 405 unique locations inside a 52×52 pixel area. The main task of `MasterController` is to read this data as fast as possible in an optimized order.

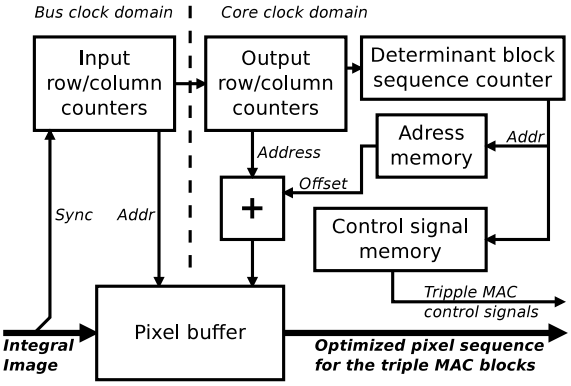


Fig. 8: The `Master controller` core block diagram.

The `PixelBuffer` inside the `MasterController` block contains 56 block RAMs each with the capacity of 36kb. Each block RAM contains exactly one image line, which simplifies the address calculation but imposes a constraint on a maximum image line length[4]. The buffer is divided into four quarters, each of which outputs its data to one output bus. Each TripleMAC can grab a value from one bus during one clock cycle. To prevent the need for re-reading any integral image pixel, all `TripleMAC`s that require the same image pixel for calculation of one of their determinants have to be able to store this value at the instant of its presence on one of the buses. Since every pixel of an integral image is required for derivative calculation at most 18×, at least 18 `TripleMAC` blocks are needed. The aforementioned architectural characteristics imposes restrictions on the integral image read order as well as on the assignment of calculated derivatives to the `TripleMAC` blocks. To construct this reading sequence we have created a set of scripts that try to assemble an optimal read sequence which respects all aforementioned constraints. During the "placement" process (the pixels are "placed" into bus cycles) the script tries to first satisfy the pixels with higher utilization, continuing to less utilized pixels, while verifying placement conformity to the aforementioned restrictions. Although the placement method is based on a relatively simple principle, its result is quite satisfying – only less than 10% of bus cycles are idle. This whole block represents, in fact, something like an algorithm-optimized cache memory, which plays a key role in the performance of our solution. Its architecture has been chosen as a tradeoff between logic utilization, performance and limitations to the image size.

[4] Currently, the image width is limited to 1024 pixels. The BRAM usage is essentially given by the maximal box filter dimension, since all the needed image lines must be in the buffer for this calculation. Hence, if the design would be ported to a bigger FPGA with say twice the BRAM resources, the image width could be also roughly doubled.

5.8 Triple MAC

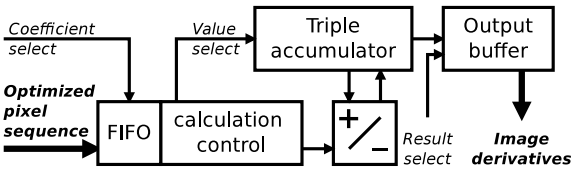


Fig. 9: The `Triple MAC` core block diagram.

Calculation of an image derivative D_{xx}, D_{yy} or D_{xy} (see section 3.2) means addition of 8 or 16 integral image values multiplied by a coefficient. Each `TripleMAC` core handles calculation of three image derivatives. This kind of multiplexing in combination with the nature of a multiply-accumulate task results in a delay in processing. The block needs two clock cycles to process a value with coefficient ±1 and four clock cycles to process a value with coefficient ±3. This is because coefficient ±3 is not implemented as a multiplication but as 3 addition/substractions. The TripleMAC block has a two position input FIFO to capture the incoming integral image data when required (this FIFO makes the read sequence assembly much simpler).

5.9 Hessian Calculator

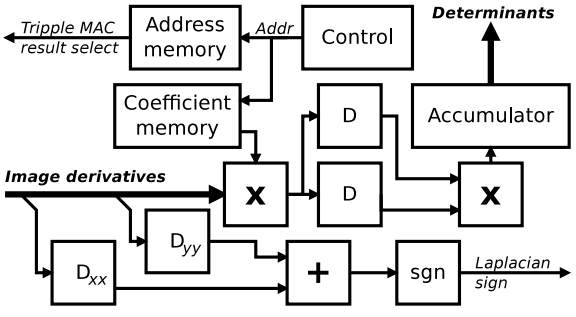


Fig. 10: The `Hessian calculator` core block diagram.

This block is responsible for the final calculation of the Fast-Hessian responses. It is activated as soon as

the Master Controller finishes the determinant block data read cycle and the image derivatives are stored in TripleMAC output registers. It uses a simple state machine to run through a predetermined sequence in which it reads the image derivatives from TripleMAC blocks and calculates the determinants. The calculation is done using fixed point arithmetic and a certain degree of precision might be lost due to the necessary rounding. This is the last important block in the SAFHG IP core, following is only a simple splitter (DBTS) that outputs Fast-Hessians to a SPB port appropriately to their octave.

5.10 Local Maxima Finder

The Local Maxima Finder (LMF) is capable of performing the thresholding and non-maxima suppression on an arbitrarily organized multi–dimensional data incoming through SPB. Its purpose is to search for local maxima of the determinants calculated by the SAFHG IP core. There are two of LMF blocks in the processing chain, one for each scale space octave. Its structure is depicted in the Figure 11.

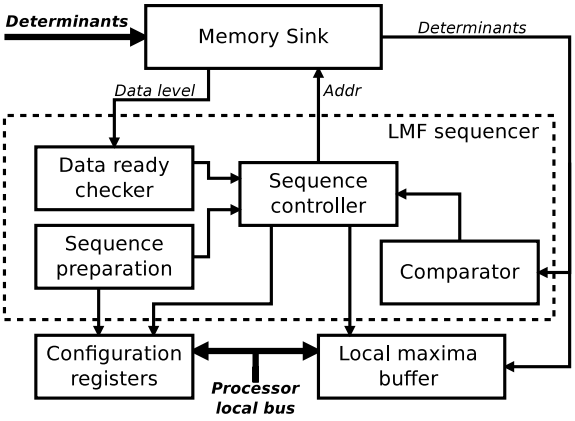


Fig. 11: The LMF core block diagram.

The Memory Sink is essentially a FIFO specifically designed to buffer two dimensional data that are coming through SPB. Since at this point we are working with 3D scale space data and SPB contains sync signals only for 2D, we have to know the data order in advance. The order of scale space data for one pixel is constant due to a constant calculation order of the SAFHG IP core and thus we are able to always identify which value belongs to which scale.

The organization of Fast Hessian data stored in the Memory Sink depends on the given octave. While the first octave has data pixels organized in 2× blocks, the second octave is organized in a pixel by pixel manner. Thus, the first octave needs 4 different offset sequences to go through each pixel scale-space neighborhood, while the second octave offset sequence is the same for every pixel. We have generated these 5 sequences using an automation script.

These sequences are passed to LMF sequencer using VHDL generic map – that means that user is eventually able to configure which LMF corresponds to which data layout (octave) using XPS IP core properties. The user of this core can also write a new offset sequence optimized for his particular data layout and re-use this core for different project. Hence, the role of LMF sequencer is to take these pre–configured sequences and apply them to data in the Memory Sink. When the running sequence discovers a neighboring data element bigger than the current comparison pilot element, it is canceled and sequencer moves to the new pilot pixel. The same applies for the comparison with threshold – it is of course done before the sequence is started. The threshold is run-time adjustable using configuration registers. During the run of the comparison sequence the neighboring scale–space values are stored to an output buffer, which is read by the software in case the local maxima is discovered and used for position interpolation. For convenience, the LMF IP Core generates interrupts when the output buffer contains a configurable amount of local maxima records

Aside from a quite complex offset calculation the sequencer has to watch for the data level in the buffer. Depending on the data layout we either have to store one or two complete lines of data in the Memory Sink. One line hast to be stored in the case when the data are transferred in serialized blocks corresponding to 2×2 pixel areas (1^{st} octave) and two lines are for a case when the data arrive only in quadruplets corresponding to each pixel (2^{nd}, subsampled octave). So, during the comparison sequence run a substantial portion of the Memory Sink has to be protected to prevent data corruption. The logic has to watch out also for image boundaries, so that data from two subsequent images are not mixed together.

5.11 HW/SW co-design summary

In this subsection we summarize the overall operation of our accelerator, focusing only on the processing chain of the SURF accelerator and including hardware-software interactions. Let's assume we have several images stored in the main memory at arbitrary locations. First, two commands per image are written into the command queue of the PDMAB – one to set the destination of integral image data and second to set the source image data

address, size and line length. Since the command queue has several positions[5] we can enqueue several images for continuous processing. Once the second command is received, the PDMAB starts feeding image data into its SPB master port. These data go through IIG, where integral image is calculated synchronously. Afterward, the data continues to the SAFHG for Fast-Hessian calculation – at this point the data flow is limited using SPB handshake signals. Integral image data stream is also simultaneously taken to the PDMAB slave port, which stores them at a requested location (first PDMAB command). Fast-Hessian results travel into the LMF blocks where, if local maxima is found, a new record (maxima and its neighborhood) is stored in the output FIFO. When the FIFO fullness reaches a certain level, an interrupt is generated and the device driver reads the local maxima records. If the FIFO is not emptied quickly enough and becomes full, the operation of the processing chain is stalled – again using the SPB handshake signals. The software then performs interpolation of the local maxima positions and calculates the SURF descriptor using the pre-stored integral image data in the main memory.

When the PDMAB finishes reading one image, it automatically moves to another (if the commands have been entered), while the software processing of the first image is still running, i.e. the PPC is calculating descriptors of a previous image while a new image is being processed by the detector chain. The user can control this process using Linux device nodes.

6 Experiments

The purpose of the performed experiments is to evaluate the overall module performance and test its correctness and applicability in a mobile robot navigation scenario. The module performance is evaluated not only in terms of the features' repeatability and distinctiveness, but also processing speed, power consumption, spatial demands and the usability in a real world scenario.

Our main concern was the impact of the changes introduced to the original SURF implementation on the algorithm ability to establish correct feature correspondences. This could have been severely affected due to decreased precision of the detector caused by using fixed-point arithmetics during the Fast-Hessian calculation. To evaluate the impact on lowering calculation precision on algorithm efficiency, we have compared the repeatability and distinctiveness of the CPU-, GPU- and FPGA-SURF with the Mikolajczyk dataset [20].

Since one of the intended module applications is ground robot visual localization, mapping and navigation, the distinctiveness of the FPGA-SURF has been measured with another, larger dataset as well. The latter dataset is more typical for a mobile robot visual navigation scenario. In a final test, a small mobile robot has autonomously traversed 1 km long path using the module as a core component of its navigation system.

[5] their number is set by an PDMAB IP core parameter

6.1 Performance with a classical dataset

The goal of this test is to compare the FPGA-SURF to the CPU- and GPU- implementations using a well-known methodic proposed by Mikolajczyk [21]. The repeatability calculation has been performed for the original CPU version of the detector, which uses double precision numbers, GPU implementation, which uses single precision numbers and the FPGA implementation, which uses fixed point arithmetics.

Since imprecision in detector calculation might affect the following stages, the distinctiveness of the algorithms was tested as well. Both repeatability and distinctiveness tests were performed on the Mikolajczyk dataset, which is available online [20] along with a set of testing scripts.

To achieve a fair comparison, we have removed the orientation assignment phase from the original implementations of CPU- and GPU- implementation and set them to use the same number of octaves as the FPGA-SURF. Since the orientation assignment has been removed and the extractor is not rotation invariant, we have tested the repeatability and distinctiveness with relevant sequences only, i.e. sequences with varying blur, viewpoint, compression and contrast.

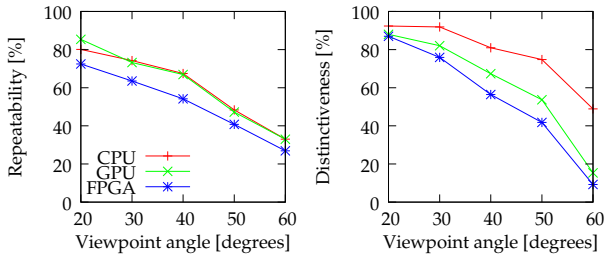


Fig. 12: Algorithm repeatability and distinctiveness with the 'wall' sequence - view point change

One can see that the repeatability of the FPGA implementation is lower compared to the SURF versions, which use floating-point arithmetics. However, distinctiveness of the detector is affected only slightly compared to the original implementations. Note the poor distinctiveness on the 'graffiti' dataset, which is caused by strong rotation of the images.

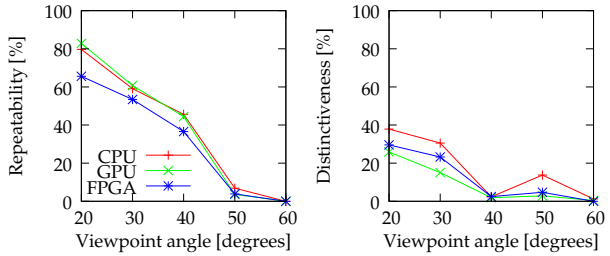


Fig. 13: Algorithm repeatability and distinctiveness with the 'graffiti' sequence - view point change

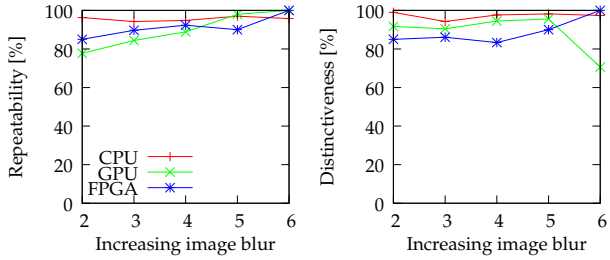


Fig. 14: Algorithm repeatability and distinctiveness with the 'bikes' sequence - variable image blur

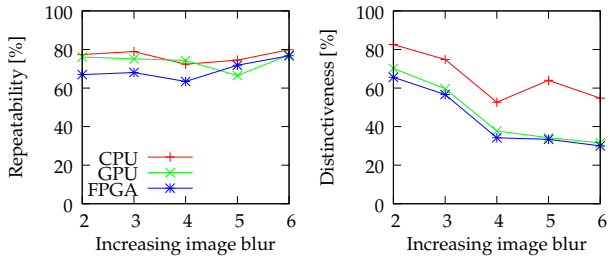


Fig. 15: Algorithm repeatability and distinctiveness with the 'trees' sequence - variable image blur

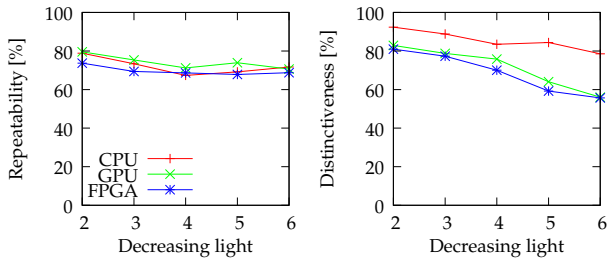


Fig. 16: Algorithm repeatability and distinctiveness with the 'Leuven' sequence - variable image contrast

6.2 Performance in a mobile robot navigation scenario

Since the module is intended primarily for visual based localization and navigation of small mobile robots, another test has been performed using a dataset typical for such a task. In case of visual based mapping, the primary measure of module's performance is its ability to track features in a sequence of images captured by its camera. This ability is closely related with the distinctiveness of the feature extractor. To establish the method distinctiveness in real world conditions, we have decided to use a dataset created by a mobile robot moving in a park-like environment. The dataset consists of 5000 (1024×768 pixel) images taken by the robot onboard camera during five 1 km long teleoperated runs in diverse conditions, see Figure 18. The robot onboard camera was aimed in the direction of the robot movement, and a picture was taken every time the robot traveled one meter or performed a sharp turn.

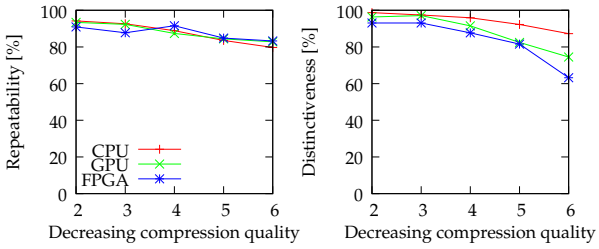


Fig. 17: Algorithm repeatability and distinctiveness with the 'UBC' sequence - variable jpeg compression

Fig. 18: Typical dataset images

6.2.1 Measuring distinctiveness

To establish the method's distinctiveness, the images were streamed through the module which was set to extract 500 SURF features per image on average. For each two consecutive images, the tentative correspondences were established based on the Euclidean distances of the SURF descriptors. Using the eight-point algorithm and RANSAC, the viewpoint change (represented by the fundamental matrix) between the images was calculated. With the calculated viewpoint change, the validity of tentative correspondences can be established

by means of epipolar geometry. To check the validity of each pair, the epipolar lines of the corresponding points were calculated using the fundamental matrix and the distance of each point from its epipolar line is calculated. If the corresponding points lie closer than 2 pixels to the epipolar lines, the correspondence is marked as valid. The ratio of valid to tentative correspondences of the entire dataset was then considered as a distinctiveness measure. During this test, the module has extracted over 2.5 million features (5000 images $\times$ 500 features/image) and has established over 500 000 tentative correspondences.

6.2.2 Measuring speed

To determine the computational performance of the module, its speed has been measured by the time needed to process the individual pictures of the aforementioned dataset. Such a test reflects the real performance of the module in a better way that measuring the speed of the individual parts of the algorithm in separate because it includes all the overheads, communication delays etc. For the sake of simplicity, we assume that the detector speed is not significantly affected by the number of features while the time to calculate one descriptor is constant, i.e. the time t to process one image can be approximated by a linear function

$$t = t_{det} \ + \ nt_{des}, \tag{2}$$

where t_{des} corresponds to the time needed to generate one descriptor, t_{det} corresponds to the detector speed and n is the number of the extracted features. To establish both the t_{des} and t_{det} constants empirically, we have measured the time needed to process the entire dataset with two different threshold values t_{100} and t_{500} that were set to obtain 100 and 500 features per image on average. Processing the dataset with the t_{100} or t_{500} thresholds took 14 and 39 minutes respectively, which corresponds to $t_{det} = 100$ ms and $t_{des} = 0.7$ ms/feature. During this experiment, the module extracted over 3 million features (5000 images $\times$ 500 features/image + 5000 images $\times$ 100 features/image).

6.2.3 Detector and descriptor in parallel

Note that the equation (2) allows to estimate the time it takes to process one image, i.e. the time needed from image reception to transmission of its SURF features. However, when images are streamed to the module continuously, the detector processes the incoming image while the descriptor is processing the last detector output, i.e. the detector and descriptor are working in parallel most of the time. In this case, the average number of processed images per second p can be approximated by

$$p = \frac{1}{\max(t_{det}, n_{avg}t_{des})}[FPS]. \tag{3}$$

where n_{avg} is the average number of features per image. Equation (3) indicates that the maximal speed of the module is given by the detector performance and if number of the detected features n_{avg} is lower than t_{det}/t_{des}, the module's performance is kept at its maximum. In Section 6.2.2, we have established the t_{det} and t_{des} to 10 ms and 0.7 ms/feature respectively, which indicates that the module can extract ~ 140 features per frame while maintaining its maximal processing speed of 10 FPS. However, if the number of extracted features n_{avg} exceeds 140, the module's performance will start to drop as n_{avg} increases (see Equation 3). E.g. while extracting 500 features per image is possible at approximately 3 FPS, extracting 1500 features takes more than one second per image. While this might seem impractical, the module's intended use, visual-based mobile robot navigation, does not require a large amount of image features [17].

6.2.4 Module performance summary

For comparison, we have measured speed and distinctiveness of the original CPU [5] and GPU [10] SURF implementation in the same way as described in Sections 6.2. The tests have been performed on a NT330-i board with Intel Atom 1.6GHz, 1GB RAM and nVidia ION2 graphics card because of this board's small size and low weight. Although the speed of FPGA and GPU implementations is higher than the CPU version, their distinctiveness is slightly lower. Compared to both GPU and CPU implementations, the FPGA-based solution is smaller, lighter and less power demanding.

		Platform		
		CPU	GPU	FPGA
Distinguishability	[%]	98	95	95
Detector speed	[ms]	5200	105	100
Descriptor speed	[ms]	1.4	0.1	0.7
Consumption	[W]	24	24	6
Mass	[g]	850	850	210
Volume	[cm^3]	600	600	180

Table 2: Comparison of CPU-, GPU- and FPGA-SURF

6.3 FPGA-SURF based navigation system

The experiments described in the previous sections have been performed offline, i.e. the module has processed a

pre-collected dataset of images. Since the results in Sections 6.1 and 6.2.4 indicate that the distinguishability of the FPGA- and GPU-SURF are similar, one would expect that they would perform similarly in real-world scenarios. However, the FPGA-SURF might suffer from some performance penalty that may not be captured by the tests described in Sections 6.1 and 6.2. Therefore, we have performed an additional test that verifies if the FPGA-SURF implementation can be used in visual-based mobile robot navigation as well as its GPU-SURF counterpart.

To test the module in a real world scenario, we have integrated it in a monocular-based mobile robot navigation system. We have chosen to use the SURFNav [17] navigation method because of its ability to cope with diverse terrain, dynamic objects, obstacles, systematic measurement errors, low visibility, variable lighting conditions and seasonal environment changes. The original navigation method described in [17] has used the GPU-SURF algorithm that has the same distinctiveness as our FPGA-SURF implementation but needs a complete PC that makes its deployment on smaller mobile robots difficult.

To show that our solution can overcome the aforementioned constraint, the experimental verification was aimed at the FPGA-SURF module ability to guide a small mobile robot. The mobile robot used was based on an MMP-5 platform from The Machine Lab. Inc., equipped with a Gumstix Overo computer. The Gumstix computer itself was running the SURFNav navigation algorithm that allows the robot to autonomously navigate routes previously taught by a human operator. Since the Gumstix Overo is too slow to achieve real-time performance when extracting the SURF and the robot cannot carry a heavy PC with a sufficient computational power, we have equipped it with the FPGA-SURF module. The module was connected to the on-board computer via an Ethernet interface and provided it with the SURF features.

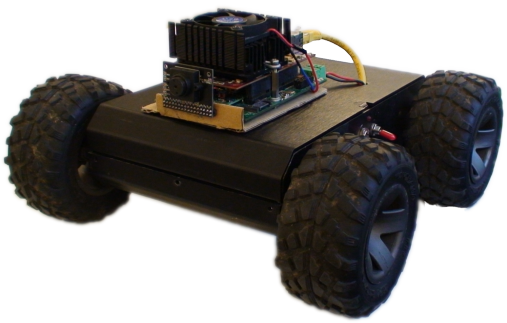


Fig. 19: MMP-5 robot with the FPGA-SURF module

To perform experimental verification, the MMP-5 robot, which was equipped with the FPGA module, has been first guided through a required path using a remote control, while storing the detected SURF features in its memory. The features recognized during this teleoperated run were then used to create a landmark map of the environment. Using this map the robot was able to navigate itself through the environment by matching the currently seen features to those previously mapped. A map of an approximately 100 m long indoor trail has been created and the robot has autonomously navigated this trail ten times.

Since the test has proven that the module deals with real-world conditions and FPGA-SURF distinguishability is similar to the GPU-SURF method, we can assume that a robot guided by this module would perform as well as described in [17]. However, performing experiments as extensive as described in [17] is beyond this paper scope.

7 Conclusion

We have presented an FPGA-based embedded module which implements the SURF image feature extraction algorithm. The presented solution, intended mainly for mobile robot navigation and mapping, offers similar distinctiveness and speed as the GPU version of the original SURF while having lower spatial and power requirements. The achieved frame rate for 1024×768 pixel images is about 10 FPS, the module dimensions are 12×8×2 cm, its mass (including camera module and heat sinks) is 210 g and it consumes less than 6 Watts. Apart from implementing the SURF algorithm, the module embeds a fully functional Linux distribution, which allows to run applications for further processing of the acquired features. Thus, the module is comfortable to use and allows its easy customization for variety of applications, which impose spatial and computational constraints, but need robust invariant image feature extraction. To demonstrate the module capabilities, the module was used to create a navigation system for a small mobile robot and guided the robot over a 1 km long path.

The IP cores occupy about 60% of the FPGA, which leaves enough space for further module improvement and optimization. In the future, we would like to increase the FPGA SURF speed by implementing the descriptor phase of the algorithm in the FPGA logic. To achieve further speedup for most computer vision applications, we also consider to add IP-cores for feature matching.

References

1. Agrawal, M., Konolige, K., Blas, M.: CenSurE: Center surround extremas for realtime feature detection and matching. In: Eur. Conf. on Computer Vision (2008)
2. Aubepart, F., El Farji, M., Franceschini, N.: FPGA implementation of elementary motion detectors for the visual guidance of micro-air-vehicles. In: IEEE International Symposium on Industrial Electronics (2004)
3. Battezzati, N., Colazzo, S., Maffione, M., Senepa, L.: SURF Algorithm in FPGA: a novel architecture for high demanding industrial applications. In: Design, Automation & Test in Europe Conference & Exhibition (2012)
4. Bauer, J., Sünderhauf, N., Protzel, P.: Comparing several implementations of two recently published feature detectors. In: Int. Conf. on Int. and Auton. Systems (2007)
5. Bay, H., Ess, A., Tuytelaars, T., van Gool, L.: Speeded-up robust features (SURF). Computer Vision and Image Understanding **110**(3), 346–359 (2008)
6. Bonato, V., Marques, E., Constantinides, G.A.: Rotation Invariant Feature Detection. IEEE Transactions on Circuits and Systems for Video Technology (2008)
7. Bouris, D., Nikitakis, A., Papaefstathiou, I.: Fast and Efficient FPGA-Based Feature Detection Employing the SURF Algorithm. In: IEEE Int. Sym. on Field-Programmable Custom Computing Machines (2010)
8. Chandrasekhar, V., et al.: Mobile Visual Search. IEEE Signal Processing Magazine (2011). DOI 10.1109/MSP.2011.940881
9. Chang, L., Hernandez-Palancar, J., Sucar, L., Arias-Estrada, M.: FPGA-based detection of SIFT interest keypoints. Machine Vision and Applications (2013)
10. Cornelis, N., van Gool, L.: Fast scale invariant feature detection and matching on programmable graphics hardware. In: IEEE International Conference on Computer Vision and Pattern Recognition (2008)
11. Davison, A.J., Reid, I.D., Molton, N.D., Stasse, O.: MonoSLAM: Real-time single camera SLAM. IEEE Transactions on Pattern Analysis and Mach. Int. (2007)
12. Grabner, M., Grabner, H., Bischof, H.: Fast approximated SIFT. In: Proceedings of the 7th Asian conference of computer vision, pp. 918–927 (2006)
13. Harris, C., Stephens, M.: A combined corner and edge detector. In: Proceedings of the 4th Alvey Vision Conference, pp. 147–151 (1988)
14. Huang, F.C., Huang, S.Y., Ker, J.W., Chen, Y.C.: High-Performance SIFT Hardware Accelerator for Real-Time Image Feature Extraction. IEEE Transactions on Circuits and Systems for Video Technology (2012)
15. Jorg, S., Langwald, J., Nickl, M.: FPGA based real-time visual servoing. In: Proceedings of the 17th International Conference on Pattern Recognition, vol. 1, pp. 749 – 752 Vol.1 (2004). DOI 10.1109/ICPR.2004.1334300
16. Ke, Y., Sukthankar, R.: PCA-SIFT: a more distinctive representation for local image descriptors. In: IEEE Conference on Computer Vision and Pattern Recognition (2004). DOI 10.1109/CVPR.2004.1315206
17. Krajník, T., et al.: Simple, yet Stable Bearing-only Navigation. Journal of Field Robotics (2010)
18. Li, J., Allinson, N.: A comprehensive review of current local features for computer vision. Neurocomputing **71** (2008). DOI 10.1016/j.neucom.2007.11.032
19. Loncomilla, P., del Solar, J.R.: Improving SIFT-based object recognition for robot applications. In: Image Analysis and Processing (2005)
20. Mikolajczyk, K., Schmid, C.: Scale & affine invariant interest point detectors **60**(1), 63–86 (2004). DOI 10.1023/B:VISI.0000027790.02288.f2
21. Mikolajczyk, K., Schmid, C.: Scale & affine invariant interest point detectors. Int. J. Comput. Vision **60**(1), 63–86 (2004). DOI 10.1023/B:VISI.0000027790.02288.f2
22. Mikolajczyk, K., Schmid, C.: A performance evaluation of local descriptors. IEEE Transactions on Pattern Analysis & Machine Intelligence **27**(10), 1615–1630 (2005). URL `http://lear.inrialpes.fr/pubs/2005/MS05`
23. Moravec, H.: Obstacle avoidance and navigation in the real world by a seeing robot rover. Ph.D. thesis, Robotics Institute, Carnegie Mellon University, Stanford
24. Mouragnon, E., Lhuillier, M., Dhome, M., Dekeyser, F., Sayd, P.: Real time localization and 3D reconstruction. In: IEEE Conf. on Computer Vision and Pattern Recognition (2006)
25. Murray, D., Little, J.J.: Using real-time stereo vision for mobile robot navigation. Autonomous Robots (2000). DOI http://dx.doi.org/10.1023/A:1008987612352
26. Price, A., Pyke, J., Ashiri, D., Cornall, T.: Real time object detection for an unmanned aerial vehicle using an FPGA based vision system. In: IEEE Int. Conf. on Robotics and Automation (2006)
27. Sarfraz, A.S., Hellwich, O.: Head pose estimation in face recognition across pose scenarios. In: Int. Conf. on Computer Vision Theory and Applications (2008)
28. Schaeferling, M., Kiefer, G.: Flex-SURF: A flexible architecture for FPGA-based robust feature extraction for optical tracking systems. In: Int. Conf. on Reconfigurable Computing and FPGAs (2010)
29. Schaeferling, M., Kiefer, G.: Object recognition on a chip: A complete SURF-based system on a single FPGA. In: Int. Conf. on Reconf. Computing and FPGAs (2011)
30. Schaeferling, M., Kiefer, G.: Object Recognition on a Chip: A Complete SURF-Based System on a Single FPGA. In: Int. Conf. on Reconfigurable Computing and FPGAs (2011)
31. Se, S., Lowe, D., Little, J.: Vision-based mobile robot localization and mapping using scale-invariant features. In: IEEE Int. Conf. on Robotics and Automation (2001)
32. Se, S., et al.: Vision based modeling and localization for planetary exploration rovers. In: International Astronautical Congress (2004)
33. Šváb, J., Krajník, T., Faigl, J., Přeučil, L.: FPGA-based Speeded Up Robust Features. In: IEEE Int. Conf. on Technologies for Practical Robot Applications (2009)
34. Thorpe, C., Hebert, M., Kanade, T., Shafer, S.: Vision and navigation for the Carnegie-Mellon Navlab. IEEE Trans. on Pattern Analysis and Machine Int. (1988)
35. Tippetts, B., Lee, D.J., Archibald, J.: An on-board vision sensor system for small unmanned vehicle applications. Machine Vision and Applications (2012)
36. Tippetts, B., et al.: FPGA implementation of a feature detection and tracking algorithm for real-time applications. In: Int. Conf. on Adv. in Visual Computing (2007)
37. Williams, J., Dawood, A., Visser, S.: FPGA-based cloud detection for real-time onboard remote sensing. In: IEEE Int. Conf. on Field-Programmable Technology (2002). DOI 10.1109/FPT.2002.1188671
38. Wu, C.: SiftGPU: A GPU implementation of scale invariant feature transform (SIFT). URL `http://cs.unc.edu/~ccwu/siftgpu`
39. Yao, L., Feng, H., Zhu, Y., Jiang, Z., Zhao, D., Feng, W.: An architecture of optimised SIFT feature detection for an FPGA implementation of an image matcher. Int. Conf. on Field-Programmable Technology pp. 30–37 (2009). DOI 10.1109/FPT.2009.5377651