FPGA BASED SPEEDED UP ROBUST FEATURES

Jan Šváb, Tomáš Krajník, Jan Faigl, Libor Přeučil

The Gerstner Laboratory for Intelligent Decision Making and Control
Department of Cybernetics, Faculty of Electrical Engineering
Czech Technical University in Prague
Karlovo namesti 13,
121 35 Prague 2,
Czech republic
Email: {svab,tkrajnik,xfaigl,preucil}@labe.felk.cvut.cz
Phone: +420 22435 5754
Fax: +420 22492 3677

ABSTRACT

We present an implementation of the Speeded Up Robust Features (SURF) on a Field Programmable Gate Array (FPGA). The SURF algorithm extracts salient points from image and computes descriptors of their surroundings that are invariant to scale, rotation and illumination changes. The interest point detection and feature descriptor extraction algorithm is often used as the first stage in autonomous robot navigation, object recognition and tracking etc. However, detection and extraction are computationally demanding and therefore can't be used in systems with limited computational power. We took advantage of algorithm's natural parallelism and implemented it's most demanding parts in FPGA logic. Several modifications of the original algorithm have been made to increase it's suitability for FPGA implementation. Experiments show, that the FPGA implementation is comparable in terms of precision, speed and repeatability, but outperforms the CPU and GPU implementation in terms of power consumption. Our implementation is intended to be used in embedded systems which are limited in computational power or as the first stage preprocessing block, which allows the computational resources to focus on higher level algorithms.

Index Terms— machine vision, visual navigation, SURF, FPGA

I. INTRODUCTION

Most of today's mobile robots utilize computer vision methods to gain information about their surrounding environment. Significant advantages of vision-based sensing are affordable price of cameras, amount of provided data and easy interpretation of these data by humans. The quantity of provided data can be also regarded as a drawback, because real-time processing of image data streams requires computationally strong hardware. However, this drawback diminishes due to Moore's law and therefore machine vision methods have found applications in mobile robotics, where real-time processing is necessary. Vision-based object recognition [1], reactive navigation [2], three–dimensional reconstruction [3], and efficient mapping and localization methods [4] [5] are used in mobile robotics [6] today.

Many of these methods rely on feature extraction algorithms [7] like Scale Invariant Feature Transformation [8], Gradient Location and Orientation Histogram [9] or Local Energy based Shape Histogram [10]. Although these algorithms are based on different principles and therefore perform differently [9], their purpose is to provide descriptors of significant image areas, which are immune to illumination and camera position changes. Since detected feature descriptors contain information independent of viewpoint and illumination, they are especially suitable for image-matching tasks. However, these algorithms are computationally demanding and therefore represent a significant bottleneck in many computer vision systems, forcing researchers to focus on improvement of their speed.

One of possible ways to increase speed of feature extraction is to exploit the fact, that image processing can easily be parallelized. Implementations of feature extraction algorithms utilizing GPUs [11] [12] have achieved performance over 30 FPS, which is considered as suitable for real-time applications. Although these implementations are based on an affordable computational hardware, small robotic platforms usually can not afford to carry entire PC system. As a consequence small robots have to use small intelligent cameras, e.g. CMUCam[1], with on-board image processing. These intelligent cameras can rely on standard microcontrollers, digital signal processors or specialized integrated circuits [13]. A popular choice for embedded machine vision

[1] http://www.cmucam.org

978-1-4244-4992-7/09/$25.00 ©2009 IEEE

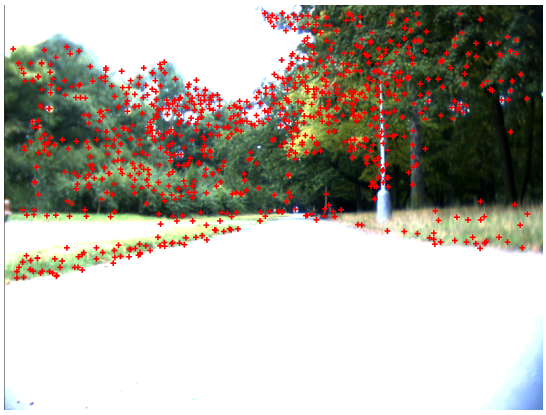

Fig. 1. Example of detected features.

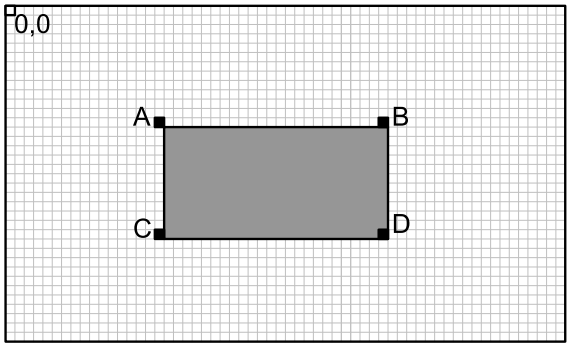

Fig. 2. Integral image principle illustration

is usage of the Field Programmable Gate Arrays (FPGA), because they effectively utilize parallel nature of the image processing. Several authors [14] [15] [16] [17] reported successful implementation of robotic vision algorithms on a FPGA-based hardware. Among these, some conclude that their SIFT algorithm implementation performs better than on conventional CPUs [17] and some demonstrate their implementation in a real robotic system [16].

One of the most efficient feature extracting algorithms is the Speeded Up Robust Features (SURF) [18], which outperforms SIFT implementations on general purpose CPUs in both speed and robustness. Most significant improvement in calculation speed is achieved by use of "Integral Image", which allows fast calculation of box filter responses used throughout the algorithm. The SURF algorithm has been reported to achieve around 2 frames per second on a general purpose CPU.

We present an implementation of this algorithm massively accelerated by the FPGA logic. Considering the FPGA architecture, we have decided that only some parts of the original SURF algorithm are suitable for implementation using FPGA logical blocks, while other parts are implemented on CPU with floating-point arithmetics. Although our implementation follows the original definition as closely as possible, we had to apply optimizations which affect precision of the detector and therefore repeatability and distinguishability of the whole algorithm. Performed experiments show that the repeatability and robustness of the FPGA-SURF have not been severely affected and are comparable to the original and GPU-SURF implementation. However, the FPGA-SURF is faster than the original - it processes approximately 10 images (1024x768 pixels) per second[1], consumes less than than 10 W and occupies less space than GPU-based system. Higher speed of feature extraction at lower size, weight and power consumption opens possibilities for applications in small embedded devices.

[1]Considering around 100 descriptors per image.

I-A. Paper structure

Next chapter will overview the SURF algorithm and outline its detection and description stages. The following section is concerned with the implementation itself. Afterwards, we present experiments comparing results of FPGA-SURF with the original SURF implementation and show, how the FPGA-SURF can be used as a part of mobile mobile robot navigation system.

II. ALGORITHM

This chapter contains brief description of SURF algorithm, according to definition in [18]. Algorithm consists of four main parts:

1) Integral image generation,
2) Fast-Hessian detector (interest point detection),
3) Descriptor orientation assignment (optional),
4) Descriptor generation.

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

Integral image is used by all subsequent parts of algorithm to significantly accelerate their speed. Integral image is defined by eq. (1). When using integral image, it is necessary to always read only four pixel values to calculate surface integral of any size from original image. Fox example integral over grayed area in image 2 is equal to $\Sigma = I_\Sigma(A) + I_\Sigma(D) - I_\Sigma(C) - I_\Sigma(B)$. This fact is widely used when calculating Gaussian and Haar wavelet filter responses.

$$\mathcal{H}(x,y) = \det \begin{pmatrix} \frac{\partial^2 f}{\partial x^2} & \frac{\partial^2 f}{\partial x \partial y} \\ \frac{\partial^2 f}{\partial x \partial y} & \frac{\partial^2 f}{\partial y^2} \end{pmatrix} \tag{2}$$

$$\begin{aligned} &\mathcal{H}(\bar{x}) = D_{xx}(\bar{x})D_{yy}(\bar{x}) - (0.9D_{xy}(\bar{x}))^2 \\ &\bar{x} = (x,y,s) \end{aligned} \tag{3}$$

SURF uses determinants of Hessian matrices to locate image's significant points. Equation (2) shows original definition of this determinant for general two dimensional function. Fast-Hessian detector modifies this equation in two significant ways: **1)** Second order partial derivatives are replaced by convolutions of image with approximated Gaussian kernels second order derivatives. Approximation is done by box filters with coefficients $1, -1, 2, -2$ [2]. Coefficient 0.9 in eq. (3) is used to compensate this approximation. **2)** Gaussian kernels are not parameterized only by position in image, but also by their size. To achieve scale invariance, algorithm searches for significant points on multiple image scale levels. Since scaling whole image is computationally expensive, SURF only scales Gaussian kernels (exactly box filters which approximate them). Scaling is represented by third parameter s in eq. 3, this parameter creates the three dimensional space of determinant results, usually refered to as "scale space". Scale differs (and is quantized) according to octaves and intervals.

Table I. Scale quantization into octaves and intervals and associated box filter sizes.

Octave	1				2		
Interval	1	2	3	...	1	2	...
Box filter edge size	9	15	21	...	15	27	...
$s = \frac{size}{9} \times 1.2$	1.2	2	2.8	...	2	3.6	...

As we can see in table I box filter size increase doubles between octaves, moreover image sampling step also doubles. When determinants in all preselected scales and intervals are calculated they are filtered by positive threshold value. Afterwards a non-local maxima suppression is performed among 26 closest neighbors in determinant scale space. Determinant local maxima marks position of interest point. However, this position has to be refined by interpolation to sub-pixel precision.

$$\mathcal{H}(\mathbf{x}) = \mathcal{H} + \frac{\partial \mathcal{H}}{\partial \mathbf{x}}^T \mathbf{x} + \frac{1}{2}\mathbf{x}^T \frac{\partial^2 \mathcal{H}}{\partial \mathbf{x}^2}\mathbf{x} \tag{4}$$

$$\hat{\mathbf{x}} = -\frac{\partial^2 \mathcal{H}^{-1}}{\partial \mathbf{x}^2}\frac{\partial \mathcal{H}}{\partial \mathbf{x}} \tag{5}$$

Equation (4) shows second order Taylor polynomial approximation of Hessian in scale space centered around examined local maxima, where $\mathbf{x} = (x, y, s)$. Position correction is obtained by seeking for zero value of this polynomial's derivative. Resulting formula for position correction is eq. (5), where $\hat{\mathbf{x}} = (\hat{x}, \hat{y}, \hat{s})$. Local maxima is accepted as interest point if none of absolute values of position correction vector $\hat{\mathbf{x}}$ elements is bigger than 0.5.

[2]Since area with coefficient -2 in D_{xx} and D_{yy} filters splits area with coefficient 1 it is possible to save some memory read cycles by not splitting underlying 1 area and by calculating with coefficient -3 instead of -2.

Rotation invariance is achieved by assigning each interest point a "dominant direction". If application doesn't require descriptors to be rotation invariant we can simply skip steps described in this paragraph and assign orientation $(0, 0)$. A circle of radius $6s$ is set around interest point. Inside a circle Haar wavelet responses in x and y direction are calculated from $4s$-sized filters with sampling step s. These responses are weighted by Gaussian function with $\sigma = 2.5s$ centered around interest point. Afterwards they are summed by a sliding sector window of $\pi/3$ with approximate step $\pi/18$. Longest vector obtained as this window sum is chosen to be descriptor dominant direction.

The SURF descriptor is calculated from square interest point's neighborhood with edge size $20s$. This neighborhood is rotated to match interest point dominant direction. This square area is further divided into 16 equal sub-squares with edge size $5s$. Inside each of these sub-square areas 25 Haar wavelet filter response pairs are calculated (filter size $2s$, sampling step s, one direction along interest point dominant direction - d_d, one perpendicular in positive sense - d_p). Responses are weighted by Gaussian function with $\sigma = 3.3s$ and summed within sub-square into four dimensional vector $\mathbf{v} = [\sum d_d, \sum d_p, \sum |d_d|, \sum |d_p|]$. These vectors are chained (one for each of 16 sub-squares) to form 64 dimensional SURF descriptor. Descriptor elements are further scaled so resulting descriptor is a unit vector.

Output of algorithm for one image gives us set of interest point locations in image's scale space along with set of descriptors describing areas around interest points. To increase matching performance the sign of Laplacian (i.e. trace of the Hessian matrix) is included into the algorithm output. Since interest points are usually found on blob-type structures this sign distinguishes light blobs on dark background from opposite.

III. IMPLEMENTATION

This chapter is dedicated to description of hardware design and software controlling functions. The algorithm to generate the SURF descriptor is written according to specification of original SURF with a few minor optimizations. As mentioned before, for the purpose of the FPGA implementation, only the Fast-Hessian detector part of SURF has been chosen for hardware-only implementation. Created hardware IP-blocks provide integral image, determinants for all inspected scale levels and also marks local maxima in calculated determinant scale-space. Generation of the SURF descriptor is handled entirely by software. Descriptor generation by itself is still quite time demanding, that is why we have chosen XC5VFX70 FPGA which incorporates PowerPC-440 CPU core with floating-point co-processor.

Hardware design of the Fast-Hessian detector introduces two major limitations in comparison with other implementations: **1)** Determinant calculation is done in an integer arithmetics with limited precision. **2)** Hardware block is de-

signed for specific number of octaves and scale intervals with limited image size. Current image size limit is 1024×1024 pixels and IP-blocks are designed to calculate determinants in 2 octaves and 4 intervals per octave.

III-A. FPGA SoC architecture

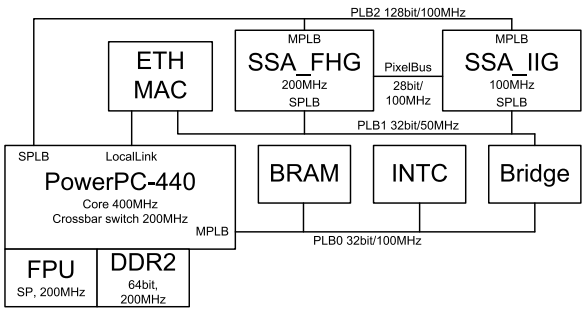

Fig. 3. FPGA SoC block schematic diagram

Fast-Hessian detector has been divided into two IP-blocks, see blocks in Figure 3: *SSA_IIG* – Integral Image Generator, *SSA_FHG* – Fast-Hessian Generator. Both IP-blocks have MPLB (Master Processor Local Bus) interfaces with simple DMA controllers, which allow them to manage data transfers from and to system memory without any CPU interaction. *PLB2* bus is designated for these transfers and has accordingly increased capacity. *PixelBus* is dedicated simple one-way bus which transfers integral image data into Fast-Hessian generator for determinant calculation. Figure 3 also includes some design details, such as bus widths and clock frequencies, which are vital in order to achieve stated performance.

III-B. Integral image generator

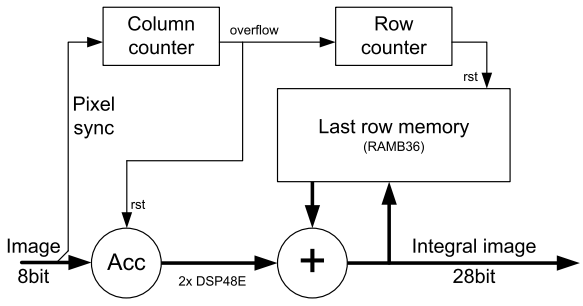

Fig. 4. Integral image generator calculation core rough block diagram

Integral image generator calculation core (fig. 4) is responsible for generation of integral image from incoming 8-bit grey scale image. It's architecture is fairly simple. IP-block *SSA_IIG* (in fig. 3) consists from this calculation core and simple DMA controller to transfer data to and from this block.

III-C. Fast-Hessian generator

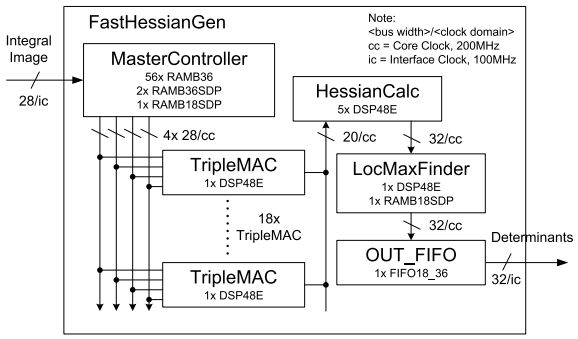

Fig. 5. Fast-Hessian calculation core block schematic diagram

Fast-Hessian calculation core carries out main functionality of *SSA_FHG* (in fig. 3) IP-block. This block generates approximated determinants of Hessian matrices and searches for local maxima among them. An overview of FPGA's dedicated blocks, bus widths and frequencies usage is summarized in Figure 5. Incoming integral image data are stored in large FIFO inside *MasterController* block. An access to integral image area of 52×52 pixels [3]. is need to calculate response of the largest Gaussian filter. Data in this area are accessed many times and in very unsequential manner, that's why *MasterController* FIFO stores 56 integral image lines and allows access to them through 4 independent busses. This configuration delivers enough bandwidth for integral image data used during determinant calculation. 18 *TripleMAC* (Multiply-Accumulate) units are connected to these busses to capture integral image data and calculate Gaussian filter responses from them. A block *HessianCalc* calculates actual determinants from Gaussians. Determinants are written into system memory using simple DMA controller in *SSA_FHG* (in fig. 3) IP-block.

To achieve higher performance optimization determinants are calculated not one by one but in "determinant blocks". Determinant block consists of all determinants calculated for 2×2 pixels image area. Since octave 2 has doubled sampling step and first 2 intervals in octave 2 have same filter sizes as intervals 2 and 4 from octave 1 we have to calculate only 18 determinants in each block (instead of 20). Integral image data are sent out through four output busses in pre-optimized order. This order is stored in "Control and addressing sequence memory". This memory also holds

[3]According to original SURF specification octave 2, interval 4 uses Gaussian filters with mask size 51×51 pixels. To calculate response of this filter using integral image we need to access one more row and column.

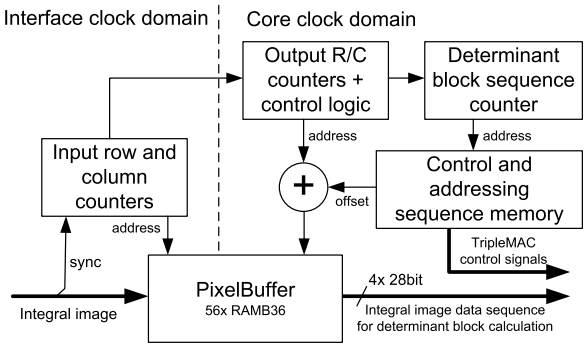

Fig. 6. *MasterController* rough block diagram

control signals for *TripleMAC* units which are sent out along with integral image data. Control and addressing sequence must be optimized in order to achieve stated performance. To carry out this optimization a specialized script, which generates this sequence, has been written. This sequence runs for every 2×2 pixels image block and delivers all integral image data needed for calculation of one determinant block.

One *TripleMAC* block handles calculation of three Gaussian filter responses. Incoming data are scaled by one of $1, -1, 3, -3$ coefficients based on input control signals and added to one of three filter responses. After each run of *MasterController*'s sequence a signal indicating end of this sequence is being generated. This signal causes *TripleMAC* blocks to load results into output buffers and to reset accumulators. On the output side all blocks are connected to one three-state bus which is used to read results from them. To save FPGA's resources *TripleMAC* performs three cycles of addition/subtraction instead of multiplication by coefficients $3, -3$. This fact has to be reflected in control and addressing sequence since *TripleMAC*s don't report *MasterController* their busy state.

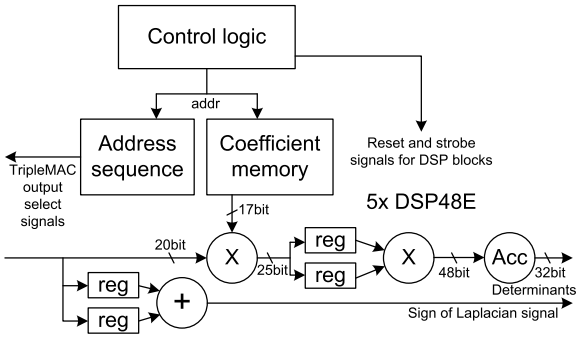

Fig. 7. *HessianCalc* rough block diagram

Block *HessianCalc* performs actual Hessian matrix determinant calculation. Gaussian filter responses, necessary for this calculation, are read from *TripleMAC* over three-state bus after *MasterController* signalizes end of determinant block sequence. Filter responses reading and determinant block calculation are controlled by control sequence stored inside *HessianCalc*.

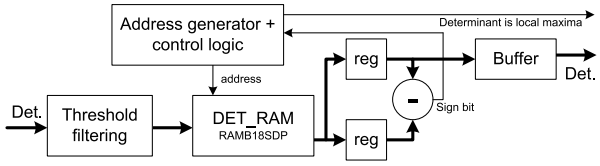

Fig. 8. *LocMaxFinder* rough block diagram

Block *LocMaxFinder* stores at least 3 determinant blocks and compares each determinant with its neighbors to search for local maxima. This block doesn't have sufficient memory resources to store all data needed to examine whole neighborhood of each determinant. Thus it performs non-maxima suppression only throughout neighbors from two determinant blocks. Each determinant marked by this block as local maxima has to be examined again by software. Determinants marked by this block are stored into FIFO in IP-block *SSA_FHG*, which is readable through SPLB interface and generates interrupts based on its occupancy.

Units *MasterController*, *HessianCalc* and *LocMaxFinder* are controlled by embedded sequences. In case of *MasterController* this sequence is quite large and needs to be stored in block RAM, other two use FPGA logic, since sequences are short. Generation of control sequences for *HessianCalc* and *LocMaxFinder* is fairly simple – these only need to process incoming determinants in preset order. However, *MasterController*'s sequence has severe impact on system's performance and is not trivial to generate. To generate this sequence only very basic algorithm has been written and thus introduces approximately 11% increase in determinant block processing time compared to optimal state[4].

III-D. Control software

This part of software manages mainly data transfers to and from IP-blocks and is represented by "libSSA2" library. Processing is divided into tasks. Library provides functions to create and process these tasks. Each task represents one image and has designated memory space to store image data, integral image data and determinants. SURF descriptors are being passed to superordinate application through callback. Task processing has four phases:

1) integral image generation,
2) determinant generation,
3) local maxima localization,

[4]Optimal state is regarded as state when *MasterController* integral image reading busses do not have to carry idle cycles to wait for busy *TripleMAC* units

4) SURF descriptor generation.

To optimize processing speed, all these phases may overlap as well as processed task may overlap. First two are carried out by hardware, library only sets data source and destination addresses in IP-block DMA controllers and waits for interrupts. Third phase is only intermediate state when image is done processing in hardware but there are still some local maxima determinants unread from *SSA_FHG* FIFO. Fourth phase is actually performing all that is left for "software" part of our SURF implementation, i.e. interest point position interpolation and SURF descriptor generation. Since all hardware is wrapped up into two Xilinx Platform Studio IP-blocks, its very simple to put more calculation accelerators on one chip and library supports this option.

IV. RESULTS

Although we have tried to follow the original description as closely as possible, some changes had to be made to optimize algorithm for implementation in the FPGA logic. In current state of development this of course applies only to detection stage. There has been a great concern for the possible degradation of the algorithm due to loss of precision in the detection phase. To evaluate impact on algorithm efficiency, we have performed tests comparing the repeatability of SURF, GPU-SURF and FPGA-SURF. These tests used a dataset of 1500 images captured by a mobile robot moving in a park environment.

IV-A. Measuring distinguishability

By distinguishability, we understand a chance of algorithm to establish a valid correspondence. This is done by establishing correspondencies solely by their descriptor Euclidean distance and checking these against geometric constraints of multiple view geometry [19].

For every feature f_a in an image A, we compute descriptor's Euclidean distance to each of feature descriptors in an image B. Two features from image B, which are most similar to f_a (i.e. they have minimal descriptor distance) are selected and therefore form two pairs with f_a. If the most similar feature pair has its descriptor distance lower than 75% of the second most matching pair descriptor distance, we proclaim it a correspondence candidate. After we find all correspondence candidates for every feature in the image A, we perform a check based on the epipolar geometry [19].

The fundamental matrix is established by the eight-point algorithm [19] and RANSAC [20]. For every feature in image A, we find an epipolar line in image B and compute the distance of the corresponding feature from that epipole. If this distance is greater than 3 pixels, the correspondence candidate is discarded, otherwise it is considered valid. The ratio of valid correspondencies to the number of correspondence candidates is a measure of the SURF algorithm distinguishability. The higher ratio means better algorithm performance.

When evaluating algorithm performance, we compute this ratio for every two consecutive images of a particular dataset. The average ratio is then taken as algorithm's distinguishability measure.

Every algorithm has been tested with four versions of the method establishing the correspondence candidates. Because the camera viewpoint doesn't change a lot between two consecutive images in the datasets, we can limit the search for a correspondence to a certain distance (200 pixels). Moreover, we might or might not match the features with same or different Laplacian sign when establishing the correspondences.

Table II. Comparison of SURF, GPU-SURF and FPGA-SURF distinguishability.

Limited search	Laplac. used	Platform		
		CPU	GPU	FPGA
Y	Y	0.62	—	0.53
Y	N	0.61	0.54	0.53
N	Y	0.43	—	0.27
N	N	0.43	0.27	0.26

The table II compares the distinguishability for our, 1500 image dataset. Similar results were obtained on a part of the dataset[2] used by the authors of the original SURF algorithm.

IV-B. Simple FPGA-SURF based navigation system

Finally, to demonstrate the usability of the proposed implementation, we deployed a navigation system with an image preprocessing module based on FPGA-SURF. The system composes of a P3AT robot with an Unibrain Fire-i601c camera with 1024x768 pixel resolution, TCM2 compass and HP 8710p laptop. This navigation system is first guided through the environment by means of teleoperation. During this drive, the robot searches for SURF features in pictures from the onboard camera. As the robot moves, it tracks the features, estimates their distance and creates a three dimensional map of them. Using this map, the system can estimate its position using currently sensed and mapped features. Therefore, the robot is able to operate in the environment. After the teleoperated drive is finished, the system can localize and navigate the robot through the mapped environment simply by matching currently captured and previously mapped. We have mapped part of the Czech technical university campus and let the robot traverse the learned path. The system has proven to be able to faultlessly navigate the learned path, which was approximately 500 meters long.

V. CONCLUSION

We have presented an implementation of the SURF algorithm massively accelerated by the FPGA logic. Despite

[2]http://www.robots.ox.ac.uk/˜vgg/research/affine/

several simplifications imposed by the limitations of FPGA hardware, our implementation offers similar distinguishability and robustness as the GPU-SURF implementation. The FPGA-SURF implementation achieves about 10 FPS at HD (1024x768 pixels) resolution, which is a necessity for real-time operation. The total power consumption of this device is less than 10 W, making it suitable for smaller robotic platforms. In the future, we will extend the algorithm by feature matching as a first step towards an embedded device realising visual localization and mapping. Moreover, we would like to port the algorithm to a Xilinx Spartan-6 family of FPGA chips, reducing the costs of the final system.

ACKNOWLEDGMENTS

This work was supported by a research grant CTU0904313 of Czech Technical University in Prague, the Research programs funded by the Ministry of Education of the Czech Republic No. MSM 6840770038, 2C06005 and by the FP7-ICT project "REPLICATOR" No. 216240.

VI. REFERENCES

[1] Patricio Loncomilla and Javier Ruiz del Solar, "Improving sift-based object recognition for robot applications," in *Image Analysis and Processing - ICIAP 2005, Italy, September 6-8, 2005, Proceedings*. 2005, vol. 3617, pp. 1084–1092, Springer.

[2] C. Thorpe, M. Hebert, T. Kanade, and S. Shafer, "Vision and navigation for the Carnegie-Mellon Navlab," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 10, no. 3, pp. 362 – 373, May 1988.

[3] E. Mouragnon, Maxime Lhuillier, M. Dhome, F. Dekeyser, and P. Sayd, "Real time localization and 3d reconstruction," in *CVPR '06: Proceedings of the 2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, Washington, DC, USA, 2006, pp. 363–370, IEEE Computer Society.

[4] Don Murray and James J. Little, "Using real-time stereo vision for mobile robot navigation," *Autonomous Robots*, vol. 8, no. 2, pp. 161–171, 2000.

[5] Andrew J. Davison, Ian D. Reid, Nicholas D. Molton, and Olivier Stasse, "MonoSLAM: Real-time single camera SLAM," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 29, no. 6, pp. 1052–1067, JUN 2007.

[6] G. N. DeSouza and A. C. Kak, "Vision for mobile robot navigation: A survey," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 24, no. 2, pp. 237–267, 2002.

[7] J. Li and N. Allinson, "A comprehensive review of current local features for computer vision," *Neurocomputing*, vol. 71, no. 10-12, pp. 1771–1787, June 2008.

[8] S. Se, D. Lowe, and J. Little, "Vision-based mobile robot localization and mapping using scale-invariant features," in *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, Seoul, Korea, May 2001, pp. 2051–2058.

[9] Krystian Mikolajczyk and Cordelia Schmid, "A performance evaluation of local descriptors," *IEEE Transactions on Pattern Analysis & Machine Intelligence*, vol. 27, no. 10, pp. 1615–1630, 2005.

[10] A. Saquib Sarfraz and Olaf Hellwich, "Head pose estimation in face recognition across pose scenarios," in *VisApp 2008: Proceedings of the 3rd International Conference on Computer Vision Theory and Applications*, HJ Araujo, Ed., 2008, pp. 235–242.

[11] N. Cornelis and L. Van Gool, "Fast scale invariant feature detection and matching on programmable graphics hardware," in *CVPR 2008 Workshop (June 27th)*, Anchorage Alaska, June 2008.

[12] Changchang Wu, "SiftGPU: A GPU implementation of scale invariant feature transform (SIFT)," online, 2007.

[13] Norbert O. Stöffler and Zoltan Schnepf, "An mpeg-processor-based robot vision system for real-time detection of moving objects by a moving observer," in *In Proceedings of the 14th International Conference on Pattern Recognition*, 1998, pp. 477–481.

[14] J. Diaz, E. Ros, S. Mota, and R. Rodriguez-Gomez, "FPGA-based architecture for motion sequence extraction," *International Journal of Electronics*, vol. 94, no. 5, pp. 435–450, May 2007.

[15] J Diaz, E Ros, F Pelayo, EM Ortigosa, and S Mota, "FPGA-based real-time optical-flow system," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 16, no. 2, pp. 274–279, Feb 2006.

[16] Vanderlei Bonato, Eduardo Marques, and George A. Constantinides, "A Parallel Hardware Architecture for Scale and Rotation Invariant Feature Detection," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 18, no. 12, pp. 1703–1712, Dec 2008.

[17] M. Tornow, J. Kaszubiak, R. W. Kuhn, B. Michaelis, and T. Schindler, "Hardware approach for real time machine stereo vision," in *WMSCI 2005: 9th World Multi-Conference on Systemics, Cybernetics and Informatics, Orlando, FL, Jul 10-13, 2005, Vol 5*, 2005, pp. 111–116.

[18] H. Bay, T. Tuytelaars, and L. Gool, "Surf: Speeded up robust features," in *Proceedings of the ninth European Conference on Computer Vision*, 2006.

[19] R. I. Hartley and A. Zisserman, *Multiple View Geometry in Computer Vision*, Cambridge University Press, ISBN: 0521540518, second edition, 2004.

[20] Martin A. Fischler and Robert C. Bolles, "Random sample consensus: a paradigm for model fitting," *Readings in computer vision: issues, problems, principles, and paradigms*, pp. 726–740, 1987.