Low-latency Image Processing for Vision-based Navigation Systems

Petr Čížek Jan Faigl Diar Masri

***Abstract*— This paper concerns a problem of the latency reduction in the vision-based mobile robot navigation, which is considered as the crucial system property to determine a control command based on visual data in practical deployments of mobile robots. The problem is addressed by a processor centric FPGA-based System-on-Chip design allowing power and computationally efficient on-line image processing. The proposed architecture is considered in an autonomous vision-based navigation with a teach-and-repeat algorithm based on detection and tracking of image salient points. The architecture has been evaluated and compared with a CPU-based solution on different platforms and the results indicate that the proposed FPGA-based implementation outperforms pure CPU solutions in the overall latency, speed, and power consumption.**

***Index Terms*— latency, visual navigation, FPGA, system on chip, image processing**

I. Introduction

Image processing is an important part of the computer vision that has a fundamental impact on the field of mobile robotics because vision-based techniques are key components to obtain information about the robot motion and its surroundings. In particular, extractors of salient points are popular image processing techniques that forms crucial building blocks for implementing robotic navigation systems; however, the feature extraction process is usually computationally very demanding.

A deployment of vision-based algorithms is a very daunting task on small and micro robotic platforms like micro aerial vehicles [1] (MAVs), swarm robots [2] or crawling platforms [3] because of limited dimensions, payload, and battery capacity. Therefore, the computational deficiency of these platforms can be addressed by more efficient algorithms and software implementations. Besides, it can also be addressed by a dedicated hardware solution.

Parallel capabilities of modern processors and graphics cards allow to process several images simultaneously [4], [5], which can increase the number of processed images per second significantly. However, it does not necessarily mean the processing results are provided in a shorter time, see Fig. 1. The time to process an image depends on the image processing pipeline, and therefore, a higher image processing throughput does not guarantee a low-latency of the data processing. The latency between the data acquisition and the produced control command is crucial in many practical robotic missions, especially in dynamic and confined spaces, which impose imminent threats to the mobile robot.

Authors are with Dept. of Computer Science, Faculty of Electrical Engineering, Czech Technical University in Prague, Czech Republic {cizekpe6|faiglj|masridia}@fel.cvut.cz

The presented work has been supported by the Czech Science Foundation (GAČR) under research project No. 15-09600Y.

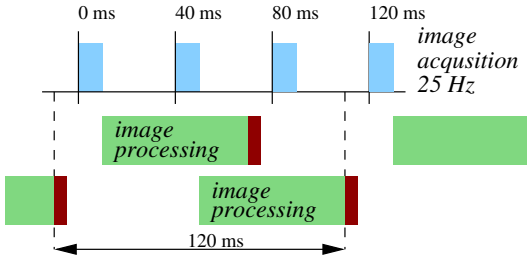

Fig. 1. An example of image processing in two parallel threads. Images are acquired every 40 ms (25 Hz) and the acquisition takes 10 ms. For 60 ms processing per image, two CPU cores can process 25 images per second. However, the processed data (denoted as the red box) are available in 70 ms from the image acquisition, which gives the effective frequency of 14 Hz.

In this paper, a processor centric Field-programmable Gate Array (FPGA) based on the System-on-Chip (SoC) design is considered for an efficient implementation of the image processing pipeline for an extraction of image salient points and their further processing in vision-based navigation systems. A real-time low-latency performance is achieved by a direct on-line processing of visual data from the sensor. The proposed approach provides benefits of both: the general purpose CPU; and a highly optimized FPGA architecture.

The main source of the achieved low-latency is in the ability to process data not as a whole image (which is typical for CPU based techniques), but in an incremental way as individual pixels are read from the image sensor. This system feature provides any-time capability of the proposed architecture to vision-based navigation techniques. In addition, the proposed architecture is of a low complexity and it has a low power consumption which makes it well suitable for computationally constrained robotic platforms.

The paper is organized as follows. Related work on FPGA-based image processing in mobile robotics is presented in Section II. A brief overview of the image processing pipeline for vision-based navigation method and the considered teach-and-repeat navigation algorithm [6] utilized for a practical deployment of the proposed architecture are provided in Section III. Latencies in the processing pipeline of the vision-based navigation are discussed in Section IV. The proposed architecture of the processor-centric FPGA SoC image processing pipeline is described in Section V. Evaluation results of the proposed architecture and its comparison with standard CPU implementations are reported in Section VI. Concluding remarks are in Section VII.

II. Related Work

One way how to evaluate a performance of vision-based algorithms is an evaluation of the processing speed in the

number of processed frames per second (fps). However, the fps can be easily confused with the latency of the method, which is a much harder to evaluate, but a more important in real robot navigation tasks.

The most related work has been presented by the Computer Vision and Geometry group with the ETH Zurich, because the authors concern and evaluate the latencies in their approaches. Latencies of data transmissions are thoroughly discussed and evaluated in [7] and [8], where a synchronization of the data acquisition process from the camera and the inertial unit (IMU) is crucial to achieve a more precise motion control of MAVs using on-board computer vision techniques.

In [9], the authors present a stereo image processing module based on the FPGA and ARM Cortex A9 quad-core CPU. The system is capable of the on-line calculation of the dense disparity images with the resolution of 752×480 pixels at 60 fps while the latency of the disparity calculation is reported to be only 2 ms. The module is extended in [10] by a reactive based collision avoidance method for MAVs and the authors report on an experimental evaluation in outdoor environments. Four stereo pairs have been utilized in [11] to build an omnidirectional collision avoidance system. The reported overall latency of the system including the image acquisition is 26.9 ms.

Notice, the further references report only on the processing speed, albeit they discuss FPGA-based computational and power efficiency improvements in a vision-based navigation.

A miniature module with a low-cost FPGA and MCU for a visual navigation is presented in [12]. A reduced PTAM algorithm [13] (mapping top of 200 features due to memory limitations) is utilized for an estimation of the visual odometry based on the on-line FPGA implementation of the FAST [14] features detection and BRIEF [15] feature description. The MCU performs the visual odometry calculation and the whole system operates on 160×120 images at 30 fps. Compared to our approach, the method [12] uses a complex memory access pattern for the feature response function calculation.

In [16], the FPGA is utilized to synchronize data from the IMU and FAST features detection from a camera stereo pair for a robust inertial assisted real-time visual SLAM. The reported processing throughput is about 20 fps for the 752×480 image. Similar results are reported in [17]. Both approaches use the SoC design with FPGA-based feature detection. The later approach presents the most similar feature detection chain to our proposed approach; however, without sufficient implementation details.

III. Overview of the Vision-based Navigation

Vision-based navigation systems basically consist of four main parts: image acquisition, feature detection, feature description, and feature processing. Although a feature processing depends on a particular localization method, almost all vision-based localization techniques perform some variant of the feature matching. Therefore, the image processing pipeline can be considered as in Fig. 2.

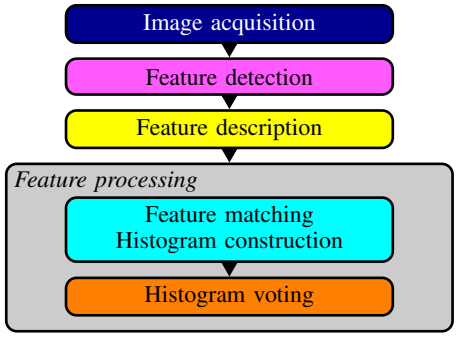

Fig. 2. Image processing pipeline for the vision-based navigation

The image acquisition is usually made by camera drivers that provide the image as a whole into a part of the system memory, denoted as the frame buffer. Besides, standard image processing libraries, e.g., OpenCV [18], are image-based oriented and take the whole image to produce a set of detected features or their descriptors. This is in a direct contrast with the proposed on-line processing which processes the image incrementally that requires a specific implementation of the processing pipeline.

A. Teach-and-Repeat Navigation

The proposed low-latency architecture for image processing has been considered with a practical deployment of the teach-and-repeat autonomous navigation method [6]. The method relies on the on-line detection and tracking of image salient points that have been previously mapped. During the repeat mode, the matched features are used to correct the robot heading and steer the robot motion to autonomously traverse the pre-learned path.

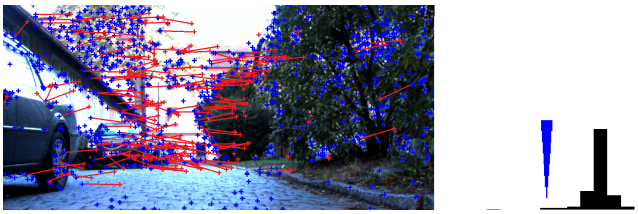

Fig. 3. Matched features from the current image with the previously learned features in the teach-and-repeat navigation [6]. A corresponding navigation histogram of the matched features is depicted on the right.

The method [6] considers a relative localization provided by the odometry only locally for traversing short straight line segments along which visual landmarks are utilized to correct the robot heading. The heading corrections are based on a modus of the horizontal displacements of the tentative correspondences between previously mapped and the currently perceived visual features, see Fig. 3. The modus is found by a histogram voting, which is quick to compute.

Although the method has been originally proposed with the SURF [19] feature detection and descriptor (also implemented on the FPGA [20]), it has been found out [21] that a better performance is achieved with the FAST [14] feature detection and BRIEF descriptor [15]. Besides, the

combination of FAST detector and BRIEF descriptor is less computationally demanding than SURF, which make them a suitable choice for the evaluation in this study targeting computationally deficient platforms.

IV. Latencies in Vision-based Navigation

The overall latency T of the vision-based navigation algorithm is the time needed to produce the control command based on the obtained visual data. The time T consists of particular latencies in each block of the processing pipeline:

$$T \simeq T_{exp}+T_{acq}+T_{det}+T_{desc}+T_{match}+T_{vote}+T_{sys}, \quad (1)$$

where we can also distinguish latency caused by the exposure time T_{exp} and latencies related to the data handling by operating system services T_{sys}. The individual latencies are discussed in the following paragraphs.

T_{exp} is the image exposure time that is determined by the camera shutter, lens aperture, and scene luminance. Although two types of shutter can be found in camera sensors (the global and rolling shutter), the type does not affect T_{exp}, because the exposure always precedes the camera readout. In the rest of this paper, T_{exp} is not considered, because it depends on the other factors than the image processing itself.

T_{acq} stands for the acquisition time, which is the time needed for the transmission of the image data from the camera sensor to the main memory. The data are transmitted pixelwise line-by-line as the camera sensor readouts the image pixels.

T_{det} is the required time for feature detection, which depends on the computational complexity of the algorithm, size of the image, and also available computational resources.

T_{desc} refers to the time needed for the computation of the feature descriptor for each detected feature. A descriptor usually describes a local image neighbourhood of the detected feature and thus, its construction requires many memory read operations, which can be computationally demanding.

T_{match} is the time needed for establishing pairwise tentative correspondences between the currently detected features and the pre-learned ones. It is based on a comparison of the feature descriptors to determine whether the features correspond to the same salient object already detected in the environment. In the utilized navigation method [6], a histogram of the horizontal displacements of the detected features is constructed during the matching phase.

T_{vote} is the time needed to find the principal bin in the histogram and determine the control command for the robot [6].

T_{sys} is a latency induced by the operating system, which is caused by, e.g., data transmissions, scheduling, etc. It is extremely volatile and thus hard to assess.

Regarding latencies and parallel computation of individual tasks of the image processing pipeline, three CPU-based ways how to organize the computation can be identified, see Fig. 4. The first is a single-threaded CPU implementation where all the blocks are processed in a sequence, and therefore, the update rate equals to the overall latency T. For multi-core CPUs, individual parts of the algorithms can

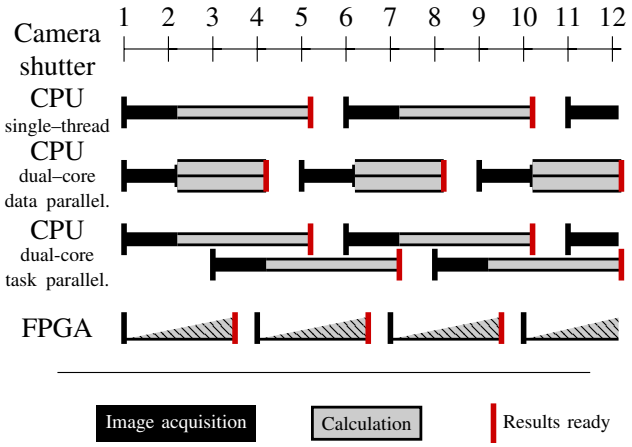

Fig. 4. Latency of image processing in CPU-based and FPGA systems for the camera with a fixed frame rate.

be accelerated by parallel data processing that results in the overall lower latency and higher update rate; or individual tasks can be parallelized to increase the update rate but it preserves the overall latency. On the other hand, the proposed FPGA architecture performs feature detection directly on the incoming data pixel stream and thus eliminates T_{acq}. Moreover, detection is performed simultaneously with the feature description and matching.

V. Proposed Low Latency Architecture

The main idea of the proposed latency reduction is based on exploiting benefits of the FPGA-based coprocessor and designing an efficient architecture for on-line processing of incoming data from the camera sensor. The architecture also exploits advances of the processor centric System-on-Chip design to combine benefits of CPU and FPGA. While a complete FPGA-based implementation of the navigation algorithm [19] would be extremely complicated, time consuming, and resource greedy; we rather focused on individual parts of the algorithm and derive which of them are suitable for the FPGA acceleration.

The feature detection and feature description are two fundamental phases of the image processing pipeline regarding the abilities of the FPGA. The detection is a more suitable for the FPGA because it can benefit from the true parallelism and it is usually calculated from a smaller area of the image than the descriptor. In addition, all points in the image need to be checked for the presence of interest point, while only few of them are recognized as features that are selected for the feature description. Therefore, it is beneficial to perform a feature description after a feature point is detected.

The developed processor-centric architecture consists of the FPGA feature detection, while at the same time, the embedded processor of the architecture is used for the feature description and further processing. The proposed architecture forms a pipeline visualized in Fig. 5 that consists of the following blocks: camera sync, preprocessor, response function calculation core, non-maxima suppression core, detector interface, memory buffering core, and processor subsystem. Particular blocks are detailed in the following paragraphs.

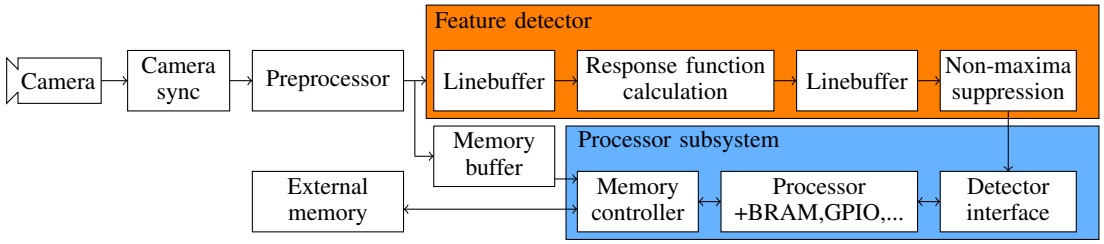

Fig. 5. Proposed FPGA-based image processing architecture for vision-based navigation

The camera sensor is directly connected to the **Camera sync** core in the FPGA fabric. The core takes the visual data on its input and provides the video stream. While the input interface of the core is a platform and implementation dependent, the output interface shall implement the following signals to achieve reusability of the other pipeline blocks for different FPGA fabric:

`pixel_clk`	Master clock, signals are valid on the clock rising edge.
`pixel_data`	Parallel pixel data output.
`pixel_blank`	Binary signal – high indicates blanking image area.
`x_cnt`, `y_cnt`	X and Y coordinates of right outputted pixel.
`h_sync`	Binary signal – horizontal synchronization pulse.

The proper function of the **Camera sync** core is essential for the correct function of the feature detection pipeline. The `pixel_clk` needs to be continuous and each line of data defined by the `h_sync` signal needs to have precisely the same number of pixels as other lines of the current image. This greatly reduces the complexity of the following cores.

The **Preprocessor** core is responsible for performing low-level pixel-wise image processing like a color conversion, histogram equalization or rectification. It produces a stream of visual data as the **Camera sync**; thus, it can be easily replaced by another core or just omitted. Then, the visual data split among the feature detection pipeline and memory transmission for a later feature description.

The true parallelism of the FPGA is utilized for calculation of the feature detector responses. The architecture is best suitable for feature detectors relying on the convolution, like [22], [19], or pixel-wise comparisons, like [14], [23], which need an image window of the size n. The feature detector works as follows.

First of all, $n-1$ lines of the visual data are buffered into the FPGA embedded memory resources as the first-in first-out (FIFO) dual-port single-clock memories with the `write_enable` and `read_enable` signals. The FIFOs inputs and outputs are chained; so, the k-th FIFOs output is connected to the $k+1$-th FIFOs input. The same applies for the `write_enable` and `read_enable` signals which are controlled by a simple state machine triggered by the `h_sync` signal. When the `h_sync` goes high, the read from the k-th FIFO and write to the $k+1$-th FIFO is enabled. Aggregated outputs of all FIFOs and the input pixel provide a simultaneous access to n vertically aligned pixels which form the input to the **Response function calculation** core.

There are two principal ways of calculation of the feature response function. Either the data from **Linebuffer** core are buffered to shift registers to provide a simultaneous access to the $n \times n$ window, or a separable convolution-like structure can be utilized. It is up to the programmer to choose a suitable implementation for the given purpose, e.g., for smaller windows, direct buffering is easier than for larger windows, for which the FPGA fabric utilization would be extremely high. It is also beneficial to use pipelining for the feature response function calculation, which means a decomposition of complex mathematical operations into individual substeps calculated in individual ticks of the `pixel_clk`.

For non-maxima suppression of feature responses, three lines of values are buffered according to the same scheme described above. The **Non-maxima suppression** core then buffers 3×3 values in the shift registers and compares the central value with the user defined threshold and its eight-neighbourhood for the detection of the local maxima. The detected features are then handed to the processor subsystem through the **Detector interface** core.

The **Detector interface** core recalculates the feature coordinates by subtracting the induced x and y pixel latencies from the current `x_cnt` and `y_cnt` coordinates since the latency induced by the processing pipeline is deterministic, and stores the features until the CPU fetched them for the further processing. The core acts as a memory-mapped slave device and the CPU can either read the content of the core in a busy loop waiting for a new feature, or more efficiently generate an interrupt signal when a feature is detected.

The inner memory resources of the FPGA are limited to store a particular part of the image necessary for the feature description. Therefore, the **Memory buffer** is utilized to share visual data with the **Processor subsystem** that computes a feature descriptor whenever a new feature is detected. Then, the feature descriptor is immediately matched with the pre-learned map stored in the memory to efficiently use resources of the embedded CPU. After that, the feature together with its descriptor and matching results are fetched to the following part of the navigation algorithm, which directly produces the control command.

VI. Evaluation

The proposed FPGA-based architecture for the vision-based navigation algorithm [6] has been experimentally compared with its pure CPU-based counterpart. The evaluation consists of the focused examination of the individual parts of the processing pipeline, see Fig. 2. The individual measured latencies follow (1) and the image feature extraction consists of the FAST [14] feature detector (FAST12) and BRIEF [15] feature descriptor. In particular, the 256bit long version of the

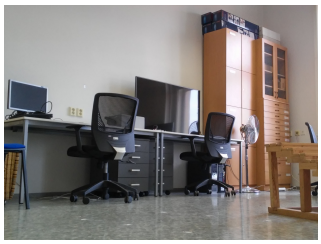

Fig. 6. A view to the environments where images have been captured during the evaluation: outdoor (left) and indoor (right).

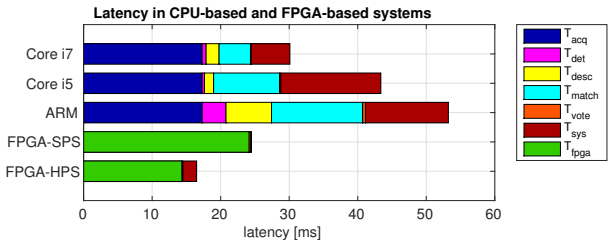

Fig. 7. Evaluation of latencies in CPU and FPGA-based systems

BRIEF feature descriptor (BRIEF-32) has been used in the evaluation. The following platforms have been considered for the comparison of the overall processing latency:

Core i7 – Lenovo Thinkpad E431 notebook with 2.2 GHz Intel Core i7 quad-core processor, 8GB RAM.

Core i5 – Samsung Q430-11 notebook with 1.7 GHz Intel Core i5 quad-core processor, 4GB RAM.

ARM – Odroid U3 embedded computer with 1.7 GHz ARM Cortex A9 quad-core microprocessor (Samsung Exynos4412) and 2GB RAM.

FPGA-SPS (soft-processor system) – A low-cost development board DE0-nano with the Altera Cyclone IV EP4CE22 FPGA utilizing the bare-metal programmed single-core NIOS IIe softcore processor clocked at 150 MHz.

FPGA-HPS (hard-processor system) – DE0-nano-SoC development board with the Altera Cyclone V SE 5CSEMA4U23 FPGA utilizing dual-core 925 MHz hard processor ARM Cortex-A9 running Yocto linux.

All the CPU-based implementations were run on Linux Ubuntu 12.04 with the Robot Operating System (ROS) Indigo and FAST and BRIEF-32 implementations from the Open Computer Vision library [18] (OpenCV 2.4.9). In the case of FPGAs, the embedded CPU has been utilized for computing the BRIEF32 descriptors, feature matching and histogram voting.

The Logitech HD Pro Webcam C920 has been used with the CPU-based implementations, whereas the Aptina MT9V034 camera sensor has been directly connected to the FPGA fabric. Both the sensors have been set to the same resolution 640×480 and 60 fps.

Two scenes considered in the evaluation are shown in Fig. 6. The number of detected features has been fixed to 200 features per image and the map used for the matching contains 200 pre-learned features.

It has to be noted that all the implementations provided similar results, since the proposed FPGA architecture only accelerates some of the most demanding computations in the FPGA fabric and it does not induce modifications to the original algorithm.

The evaluation results are summarized in Table I and visualized in Fig. 7, where the image acquisition time T_{acq} for the CPU-based platforms was measured using hardware triggered clocks as follows. One probe of the trigger has been connected directly to the camera sensor and the second probe to the io pin of the CPU-based system triggered by software. The measured T_{acq} is about 17.3 ms in average with the peak values of 16.9 ms and 22.1 ms as the lower and upper bounds, respectively. In other cases, all the reported times are average values from approx. 2000 algorithm iterations.

Notice, the results for the FPGA show aggregated latency measured for both the feature description and matching, because these two computations are made directly in a sequence before the next feature is fetched for processing on the embedded CPU of the FPGA platform.

A. Discussion

The evaluation results clearly demonstrate a computational efficiency of the FPGA. Although processing of CPU-based implementations is competitive or even faster than for the FPGA-based implementations, the FPGA outperforms the pure CPU-based solution in the latency and power consumption.

Regarding the FPGA, all the computations of the processing pipeline are distributed between the FPGA fabric, which performs the FAST feature detection, and the feature description, matching and histogram voting carried out by the embedded CPU. When using a more powerful FPGA equipped with the hard-processor system it is possible to finish all the calculation during a single image readout, which, in conclusion, implies the lowest possible latency.

It is also worth mentioning that the proposed feature detection pipeline can process up to 200 Megapixels of image data per second, which roughly corresponds to the 1920×1080 image at 60 fps, because the processing speed depends only on the rate of the `pixel_clk` with which the image data are fetched from the camera sensor. The whole navigation pipeline is; however, in such a case limited by the computational power of the embedded CPU.

The FAST feature detection and BRIEF description are one of the least computationally demanding image feature extraction techniques, the benefits of the usage of the FPGA architecture are therefore not clearly visible. However, from the experimental verification of the feature detection pipeline, it is clear that the proposed architecture has a great potential in accelerating computer vision algorithms for a real-time performance.

In order to support this claim, we evaluated the same design with the SURF feature detector [19] for which the developed FPGA on-line architecture have been verified in simulations but not yet deployed on a real hardware.

TABLE I

LATENCY EVALUATION RESULTS OF CPU-BASED AND FPGA IMPLEMENTATIONS.

Platform		**Core i7**	**Core i5**	**ARM**	**FPGA-SPS**	**FPGA-HPS**
CPU Clock		2.2 GHz	1.7 GHz	1.7 GHz	150 MHz	925 MHz
FPGA usage		-	-	-	26% / 22320‡	22% / 40000‡
Latencies						
Image acquisition	T_{acq}	17.30 ms	17.30 ms	17.30 ms	-	-
Feature detection	T_{det}	0.58 ms	0.34 ms	3.48 ms	-	-
Feature description	T_{desc}	1.89 ms	1.33 ms	6.64 ms	24.13 ms*	14.33 ms*
Feature matching	T_{match}	4.58 ms	9.62 ms	13.30 ms		
Histogram voting	T_{vote}	0.12 ms	0.18 ms	0.41 ms	0.31 ms	0.16 ms
System latencies	T_{sys}	5.63 ms	14.60 ms	12.10 ms	-	2.00 ms
Overall latency	T	**30.10 ms**	**43.40 ms**	**53.20 ms**	**24.44 ms**	**16.49 ms**
Processing speed	$T - T_{acq}$	12.80 ms	26.10 ms	35.90 ms	-	-
Power consumption		13.16 W	19.66 W	8.13 W	1.82 W	5.60 W
Estimated cost		∼1600 $	∼1000 $	79 $	79 $	99 $

* Total time for $T_{desc} + T_{match}$.

‡ Number of available logic elements.

For SURF as the detector, the feature detection on the Core i7 takes 130.96 ms and on the ARM platform it takes 280.92 ms, while the same algorithm should exhibit an on-line performance on the FPGA with the same maximum throughput of 200 Megapixels per second.

VII. CONCLUSION

In this paper, we propose a processor-centric FPGA-based architecture for a latency reduction in the vision-based robotic navigation. The architecture has been experimentally verified with a particular navigation algorithm; however, its versatility and simplicity makes it well suitable for a deployment in other visual navigation tasks relying on detection and tracking of image salient points. The presented experimental results demonstrate advantages of the proposed FPGA-based solution over pure CPU-based implementations in both considered performance indicators: 1) the overall latency; and 2) the overall power consumption, which makes proposed solution especially beneficial for computationally constrained robotic platforms.

REFERENCES

[1] N. Michael, D. Mellinger, Q. Lindsey, and V. Kumar, "The GRASP Multiple Micro-UAV Testbed," *Robotics Automation Magazine, IEEE*, vol. 17, no. 3, pp. 56–65, 2010.

[2] F. Arvin, J. Murray, C. Zhang, S. Yue *et al.*, "Colias: an autonomous micro robot for swarm robotic applications," *International Journal of Advanced Robotic Systems*, vol. 11, no. 113, pp. 1–10, 2014.

[3] D. Belter, P. Skrzypczyński, K. Walas, and D. Wlodkowic, "Affordable Multi-legged Robots for Research and STEM Education: A Case Study of Design and Technological Aspects," in *Progress in Automation, Robotics and Measuring Techniques*, ser. Advances in Intelligent Systems and Computing, 2015, vol. 351, pp. 23–34.

[4] C. G. Kim, J. G. Kim, and D. H. Lee, "Optimizing image processing on multi-core CPUs with Intel parallel programming technologies," *Multimedia Tools and Applications*, vol. 68, no. 2, pp. 237–251, 2014.

[5] A. Asaduzzaman, A. Martinez, and A. Sepehri, "A time-efficient image processing algorithm for multicore/manycore parallel computing," in *SoutheastCon*. IEEE, 2015, pp. 1–5.

[6] T. Krajník, J. Faigl, M. Vonásek, V. Kulich, K. Košnar, and L. Přeučil, "Simple yet Stable Bearing-only Navigation," *Journal of Field Robotics*, vol. 27, no. 5, pp. 511–533, 2010.

[7] L. Meier, P. Tanskanen, F. Fraundorfer, and M. Pollefeys, "PIXHAWK: A system for autonomous flight using onboard computer vision," in *ICRA*, 2011, pp. 2992–2997.

[8] L. Meier, P. Tanskanen, L. Heng, G. Lee, F. Fraundorfer, and M. Pollefeys, "PIXHAWK: A micro aerial vehicle design for autonomous flight using onboard computer vision," *Autonomous Robots*, vol. 33, no. 1-2, pp. 21–39, 2012.

[9] D. Honegger, H. Oleynikova, and M. Pollefeys, "Real-time and Low Latency Embedded Computer Vision Hardware Based on a Combination of FPGA and Mobile CPU," in *IROS*, 2014.

[10] H. Oleynikova, D. Honegger, and M. Pollefeys, "Reactive Avoidance Using Embedded Stereo Vision for MAV Flight," in *ICRA*, 2015, pp. 50–56.

[11] P. Gohl, D. Honegger, S. Omari, M. Achtelik, M. Pollefeys, and R. Siegwart, "Omnidirectional Visual Obstacle Detection using Embedded FPGA," in *IROS*, 2015.

[12] R. Konomura and K. Hori, "Visual 3D self localization with 8 gram circuit board for very compact and fully autonomous unmanned aerial vehicles," in *ICRA*, 2014, pp. 5215–5220.

[13] G. Klein and D. Murray, "Parallel Tracking and Mapping for Small AR Workspaces," in *Proceedings of 6th IEEE and ACM International Symposium on Mixed and Augmented Reality*, 2005, pp. 225–234.

[14] E. Rosten and T. Drummond, "Machine learning for high-speed corner detection," in *ECCV*, 2006, pp. 430–443.

[15] M. Calonder, V. Lepetit, C. Strecha, and P. Fua, "BRIEF: binary robust independent elementary features," in *ECCV*, 2010, pp. 778–792.

[16] J. Nikolic, J. Rehder, M. Burri, P. Gohl, S. Leutenegger, P. Furgale, and R. Siegwart, "A synchronized visual-inertial sensor system with FPGA pre-processing for accurate real-time SLAM," in *ICRA*, 2014, pp. 431–437.

[17] G. Zhou, J. Ye, W. Ren, T. Wang, and Z. Li, "On-board inertial-assisted visual odometer on an embedded system," in *ICRA*, 2014, pp. 2602–2608.

[18] Bradski, G. and Kaebler, A., *Computer vision with the OpenCV library*. O'Reilly Media, 2008.

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

[20] T. Krajník, J. Šváb, S. Pedre, P. Čížek, and L. Přeučil, "FPGA-based module for SURF extraction," *Machine vision and applications*, vol. 25, no. 3, pp. 787–800, 2014.

[21] T. Krajnik, P. Cristoforis, M. Nitsche, K. Kusumam, and T. Duckett, "Image features and seasons revisited," in *European Conference on Mobile Robots (ECMR)*, 2015, pp. 1–7.

[22] C. Harris and M. Stephens, "A Combined Corner and Edge Detector," in *Proceedings of the Alvey Vision Conference*. Alvey Vision Club, 1988, pp. 23.1–23.6.

[23] E. Rublee, V. Rabaud, K. Konolige, and G. Bradski, "ORB: An efficient alternative to SIFT or SURF," in *ICCV*, 2011, pp. 2564–2571.