Multimed Tools Appl
DOI 10.1007/s11042-017-5042-x

FPGA-based architecture of a real-time SIFT matcher and RANSAC algorithm for robotic vision applications

John Vourvoulakis[1] · **John Kalomiros**[2] · **John Lygouras**[1]

Received: 29 December 2016 / Revised: 5 July 2017 / Accepted: 18 July 2017

Abstract A fundamental problem in computer vision is finding correspondences between features in pairs of similar images. By comparing feature descriptors instead of pixel intensities, the matching capability is significantly increased. Keypoints extracted by Scale-Invariant Feature Transform (SIFT) provide superior matching ability, however, a small proportion of false corresponcences is always inevitable. The exemption of false matches is achieved using robust fitting algorithms, with RANSAC (random sample consensus) being a popular one. SIFT and RANSAC are computationally demanding and time consuming algorithms. When the target application operates in real-time, conventional approaches based on personal computers usually fail to meet the requirements. In this paper, an FPGA-based architecture for real-time SIFT matching and RANSAC algorithm is presented. The proposed scheme is applied to identify the correspondences between point features across consecutive video frames and reject the false matches. The architecture is verified using the DE2i-150 development board. Using Cyclone IV technology, the system supports a processing rate of 40fps for VGA resolution and therefore meets real-time requirements.

Keywords Scale-Invariant Feature Transform (SIFT) · Random sample consencus (RANSAC) · Field Programmable Gate Array (FPGA) · Robotic vision · Real-time · Robust fitting

1 Introduction

A common task in many computer vision problems is to establish image correspondences between similar images. Applications such as image stitching [21], object recognition [22],

✉ John Vourvoulakis
jvourv@ee.duth.gr

[1] Section of Electronics & Information Systems Technology, Department of Electrical & Computer Engineering, Polytechnic school of Xanthi, Democritus University of Thrace, 67100 Xanthi, Greece

[2] Department of Informatics Engineering, Technological and Educational Institute of Central Macedonia, Terma Magnisias, 62124 Serres, Greece

Published online: 31 July 2017

visual odometry [4], stereo vision [26] require to deal with this issue. During the last two decades, many feature extraction algorithms have been proposed, such as Scale Invariant Feature Transform (SIFT) [22, 23], Speeded Up Robust Features (SURF) [2, 3], BRIEF [5], BRISK [20]. The matching procedure was significantly improved by using descriptor-based metrics rather than comparing pixel intensities. A well-studied review of the matching performance of local descriptors was presented in [24]. Even with advanced descriptors, the matching process can result in a considerable number of false matches which need to be identified and removed. Random Sample Consensus (RANSAC) [11] is one very popular robust fitting algorithm which can be used to eliminate false matches.

In order to apply SIFT descriptor matching, the calculation of the distance between vectors of 128 dimensions is required. The feature descriptors of the first image are compared with the feature descriptors of the second image and those with the smallest distance are detected. If the smallest distance is less than a threshold value, then the corresponding pair of features is classified as a match. The aforementioned procedure constitutes a heavy processing load which can become even heavier when the number of features is several hundreds or thousands. In addition, RANSAC algorithm receives the set of correspondences, which is output from the matching step, in order to remove the false matches. In general, RANSAC is an iterative process which selects a number of random samples from a set, in order to fit a model for this set. The samples which satisfy the model are called inliers and the corresponding set constitutes the consensus set. The samples which do not satisfy the model are called outliers. The model which produces the larger consensus set is considered as valid and the corresponding inliers are kept for further processing.

Due to their huge computational complexity, SIFT matching and RANSAC processes are rarely used when real-time operation is required. Implementations based on personal computers fail to meet real-time requirements and therefore special designed hardware accelerators are used to resolve this problem. In the literature, proposed architectures make use of GPUs, FPGAs and ASICs in order to accelerate the algorithms execution, exploiting characteristics of paralellism. Unfortunately, the use of hardware in algorithms implementation has its own limitations, since there is a finite amount of resources on the chips. Some parts of the algorithms can be parallelized accelerating the execution, while others are implemented in an iterative manner using state machines.

In this paper, a complete FPGA-based architecture able to identify image correspondences for robotic vision applications is proposed. Real-time SIFT matching is applied between features in consecutive video frames. This work assumes a preceding step of SIFT feature extraction. The present architecture has been developed to supplement previous work published by the current authors [28]. However, the proposed system can be also used in conjunction with other SIFT extraction architectures presented in the literature, such as in [17]. A first implementation by the present authors of a pipelined matcher with basic RANSAC support appeared in [30]. In the present paper, both matcher circuitry and RANSAC implementation are revisited. Considering the SIFT matcher, an improved design is proposed, which is more resource efficient. As a result, the architecture can fit in a medium scale FPGA device. Furthermore, the new RANSAC implementation is more efficient in terms of false matches' elimination, under general image transformations such as translation, scaling and rotation. When a new feature is extracted from the current frame, its descriptor is compared with stored feature descriptors from the previous frame, using combinational parallel circuitry. If the matching criterion is satisfied, then the pixel coordinates are stored in on-chip RAM. A shift register structure shifts the descriptors across a moving window of 16 feature descriptors,

which is used to facilitate the pipelining of the matching procedure and supports a standard number of parallel comparisons between features. Additional shift registers are used in order to store the current feature descriptor and use it again later for a new set of comparisons, when the moving window is filled with 16 new features. The corresponding intermediate results are stored as well. By using repeated comparisons with sets of 16 descriptors in the moving window, the active size of the moving window can be practically increased, without reserving additional resources for each new set of comparisons between feature descriptors. As it has been discussed in [30], the active size of the moving window affects significantly the matching ability. The larger the moving window, the better the matching results.

Furthermore, RANSAC algorithm is applied to reject the false matches after the completion of the matching step. In the current work, the supported image transformations are isotropic scaling, translation and rotation. Although a subset of image transformations is supported in comparison with our previous work [29], the present scheme is more resource efficient, rendering the complete architecture able to fit in a mid-range FPGA device. Moreover, the limitation of the supported transformations does not affect the quality of outlier elimination, since the input frames are read with high speed and in general, the overall affine transformation does not occur frequently between frames. In feature matching between images, a set of correspondences is used to derive the transformation matrix between the two images. The required number of random samples is selected in order to compute the transformation model. Afterwards, based on the calculated matrix, the compliance of every match with the derived model is examined. Inliers constitute the true matches, while outliers constitute the false matches and consequently, they are rejected. The model which gives the higher number of inliers is selected to produce the consensus set.

The main contributions of this paper are listed below:

- A new hardware design of a SIFT matcher is proposed. The system matches features extracted in successive video frames. The design can fit in a middle range FPGA device and is capable to support real-time operation.
- RANSAC is parallelized to a significant degree; its execution requires as many clock cycles, as the selected random samples.

The rest of the paper is organized as follows. In Section II, a quick review of the related literature is presented. The proposed matcher scheme is described in Section III. In Section IV, the RANSAC module is presented. In Section V, the proposed architecture is evaluated and Section VI concludes the paper.

2 Literature survey

The current work deals with two subjects that are usually faced separately in the literature. The feature matching process in conjunction with the robust fitting algorithms is not met in standalone papers. Regarding the matching process, to the best of our knowledge, there are not any other papers that present FPGA-based implementations using SIFT features, apart from [30]. Therefore, the literature survey is extended to include matchers in general, regardless of the type of features or the special hardware used in the implementation.

In [30], the present authors introduced a SIFT matching architecture with RANSAC support. The system was capable to process 81fps, meeting real-time requirements. This was

the first attempt in the literature to solve this problem. However, certain important issues needed improvement, like the considerable amount of chip resources required for an efficient matcher and the ability of the RANSAC implementation to manage rotated images.

Condello et al. [6] presented a software-based SIFT matcher developed in the OpenCL environment. The matching procedure was accelerated by exploiting GPU's capabilities, such as parallel processing. The authors claimed that their implementation did not degrade the matching ability in comparison with the Best-Bin-First algorithm used by Lowe [23]. Their system supported processing time 833 μs/descriptor or else 1200descriptors/s.

Haiyang et al. [14] used the CUDA platform in order to host their implementation. Detection and description of SIFT features and the creation of a KD-tree space was conducted by the CPU, while the search for the nearest neighbor was conducted by the GPU using multithreading processing. The system supported up to 25fps, when VGA image resolution was used.

Fassold and Rosner [10] also used CUDA environment to apply SIFT features extraction for large scale video analysis tasks. In order to accelerate the execution, each block was optimized accordingly so that all data required for the calculations was loaded to the GPU's shared memory once and was accessed as many times as needed by various threads. As a result, global memory accesses were minimized. Their work was compared with a CPU implementation based on HessSIFT library and it was found that it accelerates the execution by a factor from 4.3 to 6, depending on the resolution and the number of SIFT features.

Fürntratt et al. [12] used a SIFT matching scheme in order to develop a brand visibility application in broadcast content. GPU took over the SIFT matching procedure between N template descriptors extracted from the logo and M descriptors extracted from the current frame. The speedup factor of the GPU implementation ranged from 6 to 10 when it was compared with a multi-threaded CPU implementation using FLANN (Fast Library for Approximate Nearest Neighbor) algorithm from OpenCV.

Wang et al. [31] proposed an embedded System-on-a-Chip for feature detection, description and matching. Feature detection used SIFT algorithm, while description and matching used BRIEF algorithm [5]. The architecture included a fully pipelined detector. Description and matching were implemented using state machines. High speed clocks were applied in the description and matching schemes, 200 MHz and 150 MHz respectively, in order to support high frame rate. The system supported processing rate 60 fps for images with resolution 1280 × 720 pixels, when the total number of features did not exceed 2000.

Kapela et al. [18] presented a hardware-software co-design, in which a FAST detector and a FREAK [1] descriptor were implemented in software. The matcher was implemented in an FPGA device. The matching scheme included parallel circuits in order to calculate the Hamming distance between feature descriptors. The higher the number of Hamming calculators, the better performance the system presented. When 64 Hamming calculators were fit in the design, the system was capable to process about 30 fps, assuming a fixed number of 128 features per frame.

Di Carlo et al. [7] introduced an FPGA-based feature matcher for space applications. The Harris corner detector was used to detect features. The matching metric involved the calculation of the un-normalized cross correlation between 11 × 11 image patches around the candidate keypoints. The search for matches was limited to keypoints with coordinates differing by no more than 17 pixels among two successive frames. The architecture supported processing speed 33 fps for images with resolution 1024 × 1024 pixels.

Regarding RANSAC algorithm, one main approach found in the literature in order to accelerate its execution exploits the capabilities of GPUs for parallel programming. Although execution speedup is achieved [16], this approach is not suitable for robotic vision applications since it requires high power. Other approaches include FPGA devices, which fit better in embedded low power applications. Since the proposed accelerator constitutes an FPGA-based implementation, we focus our survey on those papers which use FPGAs for RANSAC acceleration.

Dung et al. [9] presented an FPGA implementation of RANSAC algorithm for feature-based image registration. The paper considers affine transformation between images. The transformation matrix was calculated using a systolic array structure which consisted of twelve processing elements. Divisions were performed in one clock cycle by multiplying the dividend with the divisor's reciprocal, which was preloaded to a LUT scaled up by 5 bits. The system was able to process 30 fps in images with resolution 1024 × 1024 pixels.

Tang et al. [27] proposed a hardware/software co-design of RANSAC algorithm for real-time affine geometry estimation. The authors implemented in hardware only the most intensive task of the algorithm, i.e. the fitness of the hypothesis model to all samples. The hardware module used a pipelined scheme of 4 cycles to increase maximum clock frequency. The overall number of cycles, required to process all samples for one hypothesis model, is equal to the number of samples plus 4 cycles. The architecture was able to process a video stream of 30 fps.

Dohi et al. [8] described an FPGA implementation of ellipse estimation for eye tracking. This work included pre-processing steps, the Starburst algorithm to extract feature points of a pupil contour and finally the RANSAC algorithm. RANSAC used the extracted features to fit the best ellipse. The authors used three different methods to solve the system of the five required equations to fit the ellipse, Cramer's rule, Gauss-Jordan elimination and LU decomposition. Cramer's rule was found to be the most compact method. The system achieved a throughput of 62.5 fps.

Compared to the above implementations, the system proposed in the present paper excels in performance, since it can track a large number of true matches between video frames, at a frame rate of 40 fps. Moreover, the complete architecture of SIFT matching and RANSAC can fit into middle range FPGA devices, such as Cyclone IV.

3 SIFT matcher module

3.1 The moving window concept

A quite common method to accelerate processing is to apply operations in parallel. The parallelism of the SIFT matching process would require many concurrent comparisons between feature descriptors in the previous and in the current frame. Ideally, as a feature is detected in the current frame, its descriptor should be compared in parallel with all descriptors of the previous frame. However, this accomplishment would require huge resources. In the case of matching features between successive video frames in real-time, the two images are slightly different. If the comparisons between features are limited to those features that are located in the same image area, then the required resources could be decreased. Nevertheless, applying this limitation is not a trivial task when the camera sensor is moving arbitrarily.

In order to limit the number of comparisons, we use a moving window of feature descriptors. Each new detected feature in the current frame is matched against all descriptors detected in the previous frame that reside in the moving window. On each new detected feature, the moving window slides by one element inside the overall descriptor window. The concept is depicted in Fig. 1. The matching ability is heavily dependent on the size of the moving window. The smaller the moving window, the higher the probability that a new detected feature of the current frame is located in an image area, different from the area that contains the features of the moving window. If this happens, then the matching process is destined to fail, since the feature of the current frame will not be matched with any feature of the previous frame. Even if a match is found, it will be a false match. Rapid movements of the vision system or far views also make the problem more difficult. By increasing the size of the moving window, the possibility to get out of the area that is enclosed by the moving window is decreased. A larger moving window produces higher accuracy in the matching procedure. On the other hand, large moving windows demand extremely high resources from the FPGA chip. In our previous work [30], we found that in order to have satisfactory matching ability, the moving window size should be about 1/4th of the number of the detected features. Considering that the detected features will be some hundreds, a moving window size of 128 features could be sufficient for a wide range of applications. However, a moving window size of 128 features could fit only in high-end FPGA devices. Instead of employing a large window, we propose in the following a multiplexing scheme in which the employed moving window remains small, while the active moving window size is 128 and as a result, the matching ability of the architecture is not degraded. If the real size of moving window is selected to be 64 or 32 features, a great amount of FPGA resources is still required. In the proposed architecture, there is one block which calculates the SAD (Sum of Absolute Differences) between descriptors and two blocks which compare SAD values and forward the coordinates of the lowest ones to the next comparison level. Those blocks are depended on the moving window size. The architecture with sizes 64 or 32 cannot fit to mid-range FPGA devices such as Cyclone IV. The size of the moving window has been finally selected to be 16, which is the maximum size that produces a circuit that fits to our target FPGA device. As a result, the multiplexer should be 8 to 1 for the proposed scheme. More details about the architecture are given in the following paragraph.

3.2 The SIFT matcher hardware scheme

The proposed scheme of the SIFT matcher is depicted in Fig. 2. Various blocks inside the architecture are enclosed with dotted lines. The blocks with rounded corners declare combinational logic, while the blocks with rectangular shape indicate sequential structures. It is assumed that SIFT extraction has been applied in a previous step, as presented in [28]. Each feature descriptor is extracted in one clock cycle, as image pixels are read in a streaming manner from the CMOS camera sensor. The descriptor vector consists of 128 elements of 6 bits each one. The feature coordinates are also included, containing 11 bits for every axis and allowing support for images with resolution up to 2048 × 2048 pixels. As a result, 790 data bits are produced when a SIFT feature is detected and its descriptor is extracted. 768 bits are used to store the descriptor elements and 22 bits to save the feature coordinates.

The extracted descriptors from the previous frame are stored in the “descriptor FIFO buffer”, which is a buffer of 2048 elements. Each FIFO element has width 790 bits, so the structure can store 2048 descriptors per frame. The “SIFT module” block is described in [28]

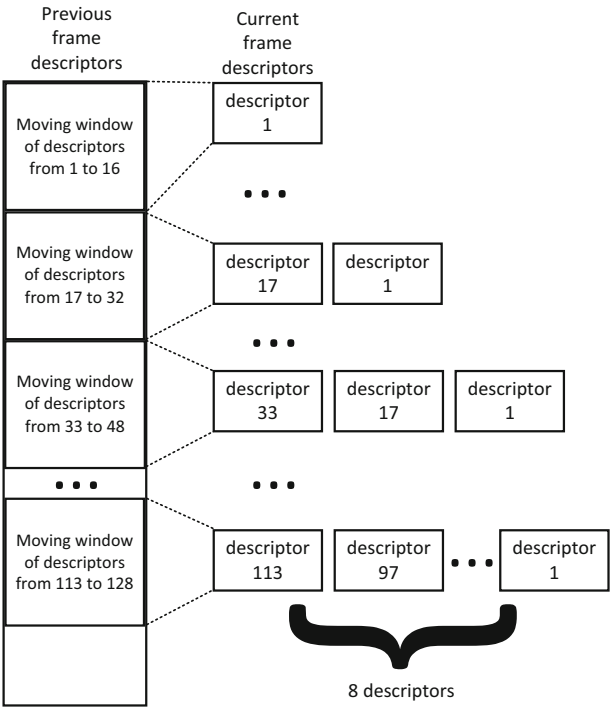


Fig. 1 The concept of moving window

and outputs streaming descriptors, as they are extracted from the current frame. At the beginning of the current frame, 16 descriptors are read from the "desctiptor FIFO buffer" and saved into the "moving window array". This task requires 16 clock cycles. The "moving window array" constitutes the descriptor moving window and in our implementation has a length of 16 descriptors.

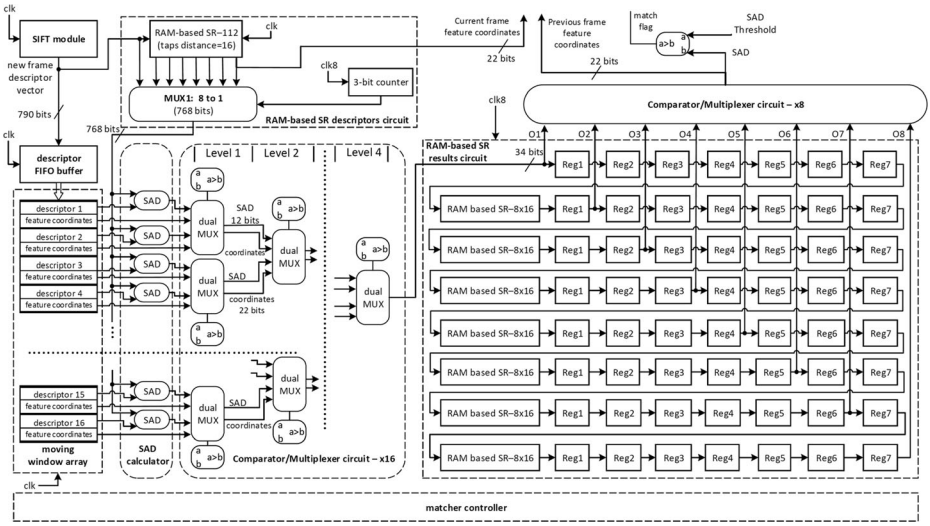


Fig. 2 The SIFT matcher architecture

The stored descriptors in the "moving window array" as well as one descriptor of the current frame are connected to the "SAD calculator" block. The "SAD calculator" is a combinational circuit, where the SAD (Sum of Absolute Differences) values between the descriptor of the current frame and the stored descriptors in the moving window of the previous frame are calculated. The output of this block is the minimum SAD value, among all possible matches, as well as the coordinates of the corresponding feature. Each SAD value is stored with 12 bits, while the coordinates are saved with 22 bits. The "SAD calculator" block and the "moving window array" are the most resource demanding circuits in the FPGA.

The "Comparator/Multiplexer circuit – ×16" performs the matching process. It receives as input the output of the "SAD calculator" block. It consists of 4 levels of comparators and multiplexers. It compares in pairs the 16 SAD values and forwards the smaller SADs and the feature coordinates to the second level. The same process is applied continuously until we reach the fourth level, in which the lowest SAD is derived. It is important to maintain the aforementioned blocks as small as they can be so that the architecture can fit in the chip. For this purpose, the number of concurrent comparisons needs to be low. On the other hand, a small number of concurrent comparisons results in a reduction of the matching ability of the system. In order to achieve a satisfactory matching ability, the system should be capable to support at least 128 concurrent comparisons [30]. However, a matcher with such a lengthy comparison block cannot fit in a mid-range FPGA device.

In order to overcome this issue, the proposed architecture adopts a multiplexing scheme based on a moving window of length 16. In this scheme, each feature from the current frame is stored into a RAM-based shift register structure and is compared again with the moving window when 16 new descriptors have been loaded. This is repeated 8 times, so that the active moving window is increased to 128, without a proportional rise in the required resources.

This multiplexing scheme is implemented in the "RAM-based SR descriptors circuit". It consists mainly of the "RAM-based SR–112" structure. This structure includes 112 shift regsiters in 7 groups, with tap distance equal to 16. The width of each shift register is 790 bits, so that one full descriptor can be stored in each one element of the structure. The output taps include the descriptor vector as well as the feature coordinates. The descriptor vector at each output tap is connected to the inputs of the "MUX1" multiplexer. The control lines of the multiplexer are driven by a 3-bit counter. The 3-bit counter clock is 8 times faster than the clock used by the "RAM-based SR–112" structure. The clock of the 3-bit counter is also 8 times faster than the clock used in the "SIFT module". Let us define the clock signal of the sequential circuits "SIFT module" and "RAM-based SR–112" as *clk*, while the 8 times faster clock of the 8-bit counter as *clk8*. On each *clk8* cycle, one different input is selected from the MUX1 to be connected to the "SAD calculator" block. As a result, 8 different feature descriptors will be compared with the current moving window by the end of 8 *clk8* cycles. The current feature descriptor will be compared again with a new moving window after the detection of 16 new features, since it will have been shifted to the shift register position that is connected to the second input of MUX1. This shift proceedure will continue until the feature descriptor reaches the last place of the shift register. Thus, each feature descriptor is compared with 8 different moving windows of 16 descriptors, giving an active moving window of 128.

The adopted multiplexing scheme produces 8 matching sub-results at different times in the pipeline sequence. Those sub-results should be synchronized and compared with each other in order to find the best match. The "RAM-based SR results circuit" block is used for this purpose. It consists of 7 shift register structures, named "RAM-based SR–8 × 16", of 128 elements each one. The shift registers width is 34 bits. This sequential curcuit is clocked by the

clk8 clock. It receives the SAD value as well as the coordinates of the corresponding feature which has been detected as the best match for the current moving window. When a new feature is detected, 8 sub-results will be saved. The shift register structure provides concurrent access to all sub-results of a specified feature. Let us see the complete journey of intermediate matching results for a detected feature. When a new feature is extracted, 8 cycles of *clk8* will shift the matching results of the first moving window – named MV1 – to Reg7 of the first line in the shift registers structure. After the detection of 16 additional features, the initial feature has shifted to the second input of the MUX1. The corresponding matching result MV1 has shifted to Reg7 of the second line, while the match with the new moving window MV2 has shifted to Reg6 of the first line. After the detection of 16 more features, the match with the descriptors of the MV1 will be in Reg7 of the third line, the match with the MV2 in Reg6 of the second line and the match with the MV3 will be in Reg5 of the first line. When a total of $16 \times 7 = 112$ features has been detected, the match with the MV1 will have shifted to Reg7 of the 8th line. Similarly, the match with the MV2 will have shifted to Reg6 of the 7th line and so on, up to the final match with the moving window MV8, which will be available at the output of the "Comparator/Multiplexer circuit – ×16".

Since the "RAM-based SR results circuit" provides concurrently the matching results between a specific feature and 8 different 16-element moving windows, a circuit for additional comparisons is required in order to produce the best match. This is achieved by using the "Comparator/Multiplexer circuit – ×8". This scheme is internaly similar to the "Comparator/Multiplexer circuit – ×16" but it has 8 inputs instead of 16. The 8 inputs require 3 levels of comparators and multiplexers. The SAD output of the "Comparator/Multiplexer circuit – ×8" block is compared with a predefined SAD threshold and if it is smaller, then the "match flag" fires. Apart from the SAD value, this circuit outputs the stored coordinates of the feature that is matched. The signal "Previous frame feature coordinates" is valid when the "match flag" is asserted.

The remaining task in order to complete the matching process is to locate the feature of the current frame inside the "RAM-based SR–112" structure, which is matched with the feature that comes out from the "Comparator/Multiplexer circuit – ×8" block. Normally, when the "RAM-based SR–122" is fully filled with data, the corresponding feature is located at the last output tap of the shift registers. Its coordinates are sent to the output of the matcher circuit, named as "Current frame feature coordinates" and together with the "Previous frame feature coordinates" constitute the match. This means that when a feature is detected in the current frame, its match with the features of the previous frame will be derived after the detection of 112 more features.

The feature matching pipeline logic is depicted in Fig. 3. The columns "clk" and "clk-n", $n = 1,\ldots,8$ show data outputs at specific clock cycles. The "Mov Window" row denotes the current moving window with features of the previous frame. Rows "O1" to "O8" indicate the outputs of the "RAM-based SR results circuit". It is assumed that n-1 features have been already detected and the "RAM-based SR-112" structure has been filled completely with feature data. When the n^{th} feature of current frame is extracted, the corresponded match against the moving window that include features from the n^{th} to the $(n + 15)^{th}$ of the previous frame is identified at the first *clk8* cycle. The matching results will be appeared at the O1 output. At the second *clk8* cycle, the best match between the $(n\text{-}16)^{th}$ feature and the previously mentioned [n, n + 15] moving window, which includes features from n^{th} to $(n + 15)^{th}$, appears at the O1 output. The same reasoning is applied to the rest outputs. At the 8th *clk8* cycle the outputs "O1" to "O8" will hold the best matching results between the $(n\text{-}112)^{th}$ feature and 8 osculating moving windows forming an overall moving window from feature (n-112) to

(n + 15), which means an active moving window of 128 elements. When the $(n + 1)^{th}$ feature is detected from the current frame, the "O1" to "O8" outputs will hold the matches between the $(n\text{-}111)^{th}$ feature of the current frame and the moving window from (n-111) to (n + 16). Finally, the n^{th} feature of the current frame will have been matched when (n + 112) features have been detected after the 8th *clk8* cycle.

There is also one more block, which is named "matcher controller". The "matcher controller" is responsible for the control of sequential operations of the matcher scheme. It produces all control signals that the matcher blocks require for their operability. It also produces the required clock enable signals to create the *clk* signal from the faster *clk8* clock. It loads the initial descriptors from the "descriptor FIFO buffer" to the "moving window array" at the beginning of each frame. It stores every new extracted descriptor of the current frame in the FIFO buffer. In general, its logic is implemented using state machines.

Moreover, the "matcher controller" manages the behavior of the matcher at specific points of the process, for example at the start and at the end of a frame. Let us assume that the last feature of the current frame has been extracted while the "descriptor FIFO buffer" still contains descriptors, meaning that it is not empty. In this case, the "matcher controller" will produce the appropriate *clk* enable signals to the RAM-based shift register structures so that in each *clk* cycle, the last feature will be propagated until it arrives at the last output tap of the "RAM-based SR–112" structure. If this is accomplished while the moving window still does not have reached to the end of the "descriptor FIFO buffer", then the last descriptors in the FIFO buffer will not be matched. This case occurs when the previous frame has far more features than the current frame. If the moving window reaches the end of the "descriptor FIFO buffer" before the last feature comes out from the "RAM-based SR–112" structure, then the moving window will stall while it contains the last 16 descriptors. This means that some of the last detected features will be matched with a descriptor moving window with active size less than 128. The

	clocks	clk							
		clk8-1	clk8-2	clk8-3	clk8-4	clk8-5	clk8-6	clk8-7	clk8-8
	MUX1: 8 to 1	n	n-16	n-32	n-48	n-64	n-80	n-96	n-112
f	Mov Window	[n, n+15]	[n, n+15]	[n, n+15]	[n, n+15]	[n, n+15]	[n, n+15]	[n, n+15]	[n, n+15]
e	O1	n vs [n, n+15]	n-16 vs [n, n+15]	n-32 vs [n, n+15]	n-48 vs [n, n+15]	n-64 vs [n, n+15]	n-80 vs [n, n+15]	n-96 vs [n, n+15]	n-112 vs [n, n+15]
a	O2	don't care	n-16 vs [n-16, n-1]	n-32 vs [n-16, n-1]	n-48 vs [n-16, n-1]	n-64 vs [n-16, n-1]	n-80 vs [n-16, n-1]	n-96 vs [n-16, n-1]	n-112 vs [n-16, n-1]
t	O3	don't care	don't care	n-32 vs [n-32, n-15]	n-48 vs [n-32, n-15]	n-64 vs [n-32, n-15]	n-80 vs [n-32, n-15]	n-96 vs [n-32, n-15]	n-112 vs [n-32, n-15]
u	O4	don't care	don't care	don't care	n-48 vs [n-48, n-33]	n-64 vs [n-48, n-33]	n-80 vs [n-48, n-33]	n-96 vs [n-48, n-33]	n-112 vs [n-48, n-33]
r	O5	don't care	don't care	don't care	don't care	n-64 vs [n-64, n-49]	n-80 vs [n-64, n-49]	n-96 vs [n-64, n-49]	n-112 vs [n-64, n-49]
e	O6	don't care	don't care	don't care	don't care	don't care	n-80 vs [n-80, n-65]	n-96 vs [n-80, n-65]	n-112 vs [n-80, n-65]
n^{th}	O7	don't care	don't care	don't care	don't care	don't care	don't care	n-96 vs [n-96, n-81]	n-112 vs [n-96, n-81]
	O8	don't care	don't care	don't care	don't care	don't care	don't care	don't care	n-112 vs [n-112, n-97]
	No new feature								
	clocks	clk							
		clk8-1	clk8-2	clk8-3	clk8-4	clk8-5	clk8-6	clk8-7	clk8-8
	MUX1: 8 to 1	n + 1	n-15	n-31	n-47	n-63	n-79	n-95	n-111
f	Mov Window	[n+1, n+16]	[n+1, n+16]	[n+1, n+16]	[n+1, n+16]	[n+1, n+16]	[n+1, n+16]	[n+1, n+16]	[n+1, n+16]
e	O1	n+1 vs [n+1, n+16]	n-15 vs [n+1, n+16]	n-31 vs [n+1, n+16]	n-47 vs [n+1, n+16]	n-63 vs [n+1, n+16]	n-79 vs [n+1, n+16]	n-95 vs [n+1, n+16]	n-111 vs [n+1, n+16]
a	O2	don't care	n-15 vs [n-15, n]	n-31 vs [n-15, n]	n-47 vs [n-15, n]	n-63 vs [n-15, n]	n-79 vs [n-15, n]	n-95 vs [n-15, n]	n-111 vs [n-15, n]
t	O3	don't care	don't care	n-31 vs [n-31, n-14]	n-47 vs [n-31, n-14]	n-63 vs [n-31, n-14]	n-79 vs [n-31, n-14]	n-95 vs [n-31, n-14]	n-111 vs [n-31, n-14]
u	O4	don't care	don't care	don't care	n-47 vs [n-47, n-32]	n-63 vs [n-47, n-32]	n-79 vs [n-47, n-32]	n-95 vs [n-47, n-32]	n-111 vs [n-47, n-32]
r	O5	don't care	don't care	don't care	don't care	n-63 vs [n-63, n-48]	n-79 vs [n-63, n-48]	n-95 vs [n-63, n-48]	n-111 vs [n-63, n-48]
e	O6	don't care	don't care	don't care	don't care	don't care	n-79 vs [n-79, n-64]	n-95 vs [n-79, n-64]	n-111 vs [n-79, n-64]
$(n+1)^{th}$	O7	don't care	don't care	don't care	don't care	don't care	don't care	n-95 vs [n-95, n-80]	n-111 vs [n-95, n-80]
	O8	don't care	don't care	don't care	don't care	don't care	don't care	don't care	n-111 vs [n-111, n-96]
	Detection of n+16 new features								
	clocks	clk							
		clk8-1	clk8-2	clk8-3	clk8-4	clk8-5	clk8-6	clk8-7	clk8-8
	MUX1: 8 to 1	n+16	n	n-16	n-32	n-48	n-64	n-80	n-96
f	Mov Window	[n+16, n+31]	[n+16, n+31]	[n+16, n+31]	[n+16, n+31]	[n+16, n+31]	[n+16, n+31]	[n+16, n+31]	[n+16, n+31]
e	O1	n+16 vs [n+16, n+31]	n vs [n+16, n+31]	n-16 vs [n+16, n+31]	n-32 vs [n+16, n+31]	n-48 vs [n+16, n+31]	n-64 vs [n+16, n+31]	n-80 vs [n+16, n+31]	n-96 vs [n+16, n+31]
a	O2	don't care	n vs [n, n+15]	n-16 vs [n, n+15]	n-32 vs [n, n+15]	n-48 vs [n, n+15]	n-64 [n, n+15]	n-80 vs [n, n+15]	n-96 vs [n, n+15]
t	O3	don't care	don't care	n-16 vs [n-16, n-1]	n-32 vs [n-16, n-1]	n-48 vs [n-16, n-1]	n-64 vs [n-16, n-1]	n-80 vs [n-16, n-1]	n-96 vs [n-16, n-1]
u	O4	don't care	don't care	don't care	n-32 vs [n-32, n-17]	n-48 vs [n-32, n-17]	n-64 vs [n-32, n-17]	n-80 vs [n-32, n-17]	n-96 vs [n-32, n-17]
r	O5	don't care	don't care	don't care	don't care	n-48 vs [n-48, n-33]	n-64 vs [n-48, n-33]	n-80 vs [n-48, n-33]	n-96 vs [n-48, n-33]
e	O6	don't care	don't care	don't care	don't care	don't care	n-64 vs [n-64, n-49]	n-80 vs [n-64, n-49]	n-96 vs [n-64, n-49]
$(n+16)^{th}$	O7	don't care	don't care	don't care	don't care	don't care	don't care	n-80 vs [n-80, n-65]	n-96 vs [n-80, n-65]
	O8	don't care	don't care	don't care	don't care	don't care	don't care	don't care	n-96 vs [n-96, n-81]
	Detection of n+112 new features								
	clocks	clk							
		clk8-1	clk8-2	clk8-3	clk8-4	clk8-5	clk8-6	clk8-7	clk8-8
	MUX1: 8 to 1	n+112	n+96	n+80	n+64	n+48	n+32	n+16	n
f	Mov Window	n+112, n+127	n+112, n+128	n+112, n+129	n+112, n+130	n+112, n+131	n+112, n+132	n+112, n+133	n+112, n+134
e	O1	n+112 vs [n+112, n+127]	n+96 vs [n+112, n+127]	n+80 vs [n+112, n+127]	n+64 vs [n+112, n+127]	n+48 vs [n+112, n+127]	n+32 vs [n+112, n+127]	n+16 vs [n+112, n+127]	n vs [n+112, n+127]
a	O2	don't care	n+96 vs [n+96, n+111]	n+80 vs [n+96, n+111]	n+64 vs [n+96, n+111]	n+48 vs [n+96, n+111]	n+32 vs [n+96, n+111]	n+16 vs [n+96, n+111]	n vs [n+96, n+111]
t	O3	don't care	don't care	n+80 vs [n+80, n+95]	n+64 vs [n+80, n+95]	n+48 vs [n+80, n+95]	n+32 vs [n+80, n+95]	n+16 vs [n+80, n+95]	n vs [n+80, n+95]
u	O4	don't care	don't care	don't care	n+64 vs [n+64, n+79]	n+48 vs [n+64, n+79]	n+32 vs [n+64, n+79]	n+16 vs [n+64, n+79]	n vs [n+64, n+79]
r	O5	don't care	don't care	don't care	don't care	n+48 vs [n+48, n+63]	n+32 vs [n+48, n+63]	n+16 vs [n+48, n+63]	n vs [n+48, n+63]
e	O6	don't care	don't care	don't care	don't care	don't care	n+32 vs [n+32, n+47]	n+16 vs [n+32, n+47]	n vs [n+32, n+47]
$(n+112)^{th}$	O7	don't care	don't care	don't care	don't care	don't care	don't care	n+16 vs [n+16, n+31]	n vs [n+16, n+31]
	O8	don't care	don't care	don't care	don't care	don't care	don't care	don't care	n vs [n, n+15]

Fig. 3 Matching sequence between incoming features of the current frame and stored features of the previous frame

last case is when the moving window reaches the end of the "descriptor FIFO buffer" while features are still being detected in the current frame. In such a case, the last 16 features of the previous frame will be compared continuously with every new feature of the current frame. This occurs when the current frame has far more features than the previous frame.

4 RANSAC algorithm hardware module

As it was mentioned previously, RANSAC is used in order to remove the false correspondences from the set of matches that comes out from the matcher scheme. The image transformation matrix is calculated using the required number of random matches. The transformations taken into account are rotation, scaling and translation. Based on each produced matrix, the rest of the matches are examined and classified as inliers or outliers. The transformation matrix which gives the higher number of inliers is considered to describe best the image transformation and is selected to produce the consensus set of the true matches. It makes sense, however, to consider as true matches the inliers of other transformation matrices as well, if those are supported by adequately many matches, according to a predefined threshold. Such inliers are attributed to different scales within the same image. This is possible due to the existence of objects at different distances from the camera.

4.1 Computation of the Transformation Matrix

Let us consider a set of n matches between the current and the previous frame as $U = \{\{(x_{1a}, y_{1a}), (x_{1b}, y_{1b})\}, \{(x_{2a}, y_{2a}), (x_{2b}, y_{2b})\}, \ldots, \{(x_{na}, y_{na}), (x_{nb}, y_{nb})\}\}$, where index a denotes the current frame and b the previous frame. By taking into account rotation, scaling and translation between frames, the set of matches should satisfy (1).

$$\begin{bmatrix} x_a \\ y_a \end{bmatrix} = R_{\text{tra}} \times R_{sca} \times R_{rot} \times \begin{bmatrix} x_b \\ y_b \end{bmatrix} \tag{1}$$

where $\boldsymbol{R_{rot}}$, $\boldsymbol{R_{sca}}$ and $\boldsymbol{R_{tra}}$ are defined in Homogenous space [13], as in (2).

$$R_{rot} = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}, R_{sca} = \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix}, R_{tra} = \begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix} \tag{2}$$

Equation (1) assumes that rotation about the point with coordinates (0,0) is performed first, then scaling is applied and finally, translation is added. This does not mean that the formula will fail if a transformed image is produced with different order. The formula will fit the independent variables appropriately to describe best the final transformation.

We also assume isotropic scaling, resulting in $s_x = s_y = s$. Combining (1) and (2), we end up with the following equations:

$$\left\{ \begin{array}{l} x_{ra} = s \times \cos\theta \times x_{rb} - s \times \sin\theta \times y_{rb} + t_x \\ y_{ra} = s \times \sin\theta \times x_{rb} + s \times \cos\theta \times y_{rb} + t_y \end{array} \right\} \tag{3}$$

In (3), there are two equations with four unknowns, which are s, θ, t_x and t_y. Therefore, two samples from the set of matches are required to compute the model. By selecting two random

matches, e.g. r_1 and r_2, and setting $A = s{\cdot}\cos\theta$, $B = s{\cdot}\sin\theta$, the following equations hold:

$$\left\{\begin{array}{c} x_{r1a} = A \times x_{r1b} - B \times y_{r1b} + t_x \\ y_{r1a} = B \times x_{r1b} + A \times y_{r1b} + t_y \\ x_{r2a} = A \times x_{r2b} - B \times y_{r2b} + t_x \\ y_{r2a} = B \times x_{r2b} + A \times y_{r2b} + t_y \end{array}\right\} \tag{4}$$

By setting the constraint $|x_{r1b} - x_{r2b}| + |y_{r1b} - y_{r2b}| \neq 0$ and solving (4), A, B, t_x, t_y are calculated as in (5).

$$\left\{\begin{array}{c} A = \dfrac{(y_{r1a} - y_{r2a}) \times (y_{r1b} - y_{r2b}) + (x_{r1a} - x_{r2a}) \times (x_{r1b} - x_{r2b})}{(x_{r1b} - x_{r2b})^2 + (y_{r1b} - y_{r2b})^2} \\ B = \dfrac{(y_{r1a} - y_{r2a}) \times (x_{r1b} - x_{r2b}) + (x_{r1a} - x_{r2a}) \times (y_{r1b} - y_{r2b})}{(x_{r1b} - x_{r2b})^2 + (y_{r1b} - y_{r2b})^2} \\ t_x = x_{r1a} - A \times x_{r1b} + B \times y_{r1b} \\ t_y = y_{r1a} - B \times x_{r1b} - A \times y_{r1b} \end{array}\right\} \tag{5}$$

When a valid model is produced, all matches are examined if they satisfy the model. Every matched feature in the previous frame is projected to the current frame using the transformation matrix. If the projection and the corresponding matched feature in the current frame have Manhattan distance less than 2, the sample pair is considered an inlier, i.e. a true match:

$$|A \times x_b - B \times y_b + t_x - x_a| + \left|B \times x_b + A \times y_b + t_y - y_b\right| \{£2 \tag{6}$$

In the proposed architecture, no floating point numbers are used. Instead, every arithmetic operation is performed using integer numbers. In order to calculate the Manhattan distance with satisfactory accuracy, A, B, t_x and t_y should be scaled up by at least 8 bits. The equations were applied in hardware as quoted in (7) and (8).

$$\left\{\begin{array}{c} A = \dfrac{[(y_{r1a} - y_{r2a}) \times (y_{r1b} - y_{r2b}) + (x_{r1a} - x_{r2a}) \times (x_{r1b} - x_{r2b})] \times 256}{(x_{r1b} - x_{r2b})^2 + (y_{r1b} - y_{r2b})^2} \\ B = \dfrac{[(y_{r1a} - y_{r2a}) \times (x_{r1b} - x_{r2b}) + (x_{r1a} - x_{r2a}) \times (y_{r1b} - y_{r2b})] \times 256}{(x_{r1b} - x_{r2b})^2 + (y_{r1b} - y_{r2b})^2} \\ t_x = 256 \times x_{r1a} - A \times x_{r1b} + B \times y_{r1b} \\ t_y = 256 \times y_{r1a} - B \times x_{r1b} - A \times y_{r1b} \end{array}\right\} \tag{7}$$

$$|A \times x_b - B \times y_b + t_x - 256 \times x_a| + \left|B \times x_b + A \times y_b + t_y - 256 \times y_b\right| \{£512 \tag{8}$$

4.2 RANSAC hardware scheme

The hardware implementation of the RANSAC algorithm is depicted in Fig. 4. The RANSAC module consists of 3 blocks, the "transformation matrix calculator", the "inliers count calculator" and the "RANSAC controller".

The "transformation matrix calculator" implements (7). It receives as inputs two samples from the set of matches and it stores them in the "Array of random samples". This is an 8-element array, with each element having a width of 11 bits. The "Combinational logic 1" includes the required multipliers and adders, in order to compute the two numerators as well as

the denominator of (7). The two dividers perform the divisions in (7). The dividers produce the longest propagation delay in the overall architecture and consequently, registering is used to increase the maximum clock frequency. The "Combinational logic 2" block calculates the t_x and t_y components. Since there is a delay of two clock cycles in computing A and B, the random sample $\{(x_{r1a}, y_{r1a}), (x_{r1b}, y_{r1b})\}$, which is required for the calculations, is also registered accordingly. The output of the block feeds the "inliers count calculator".

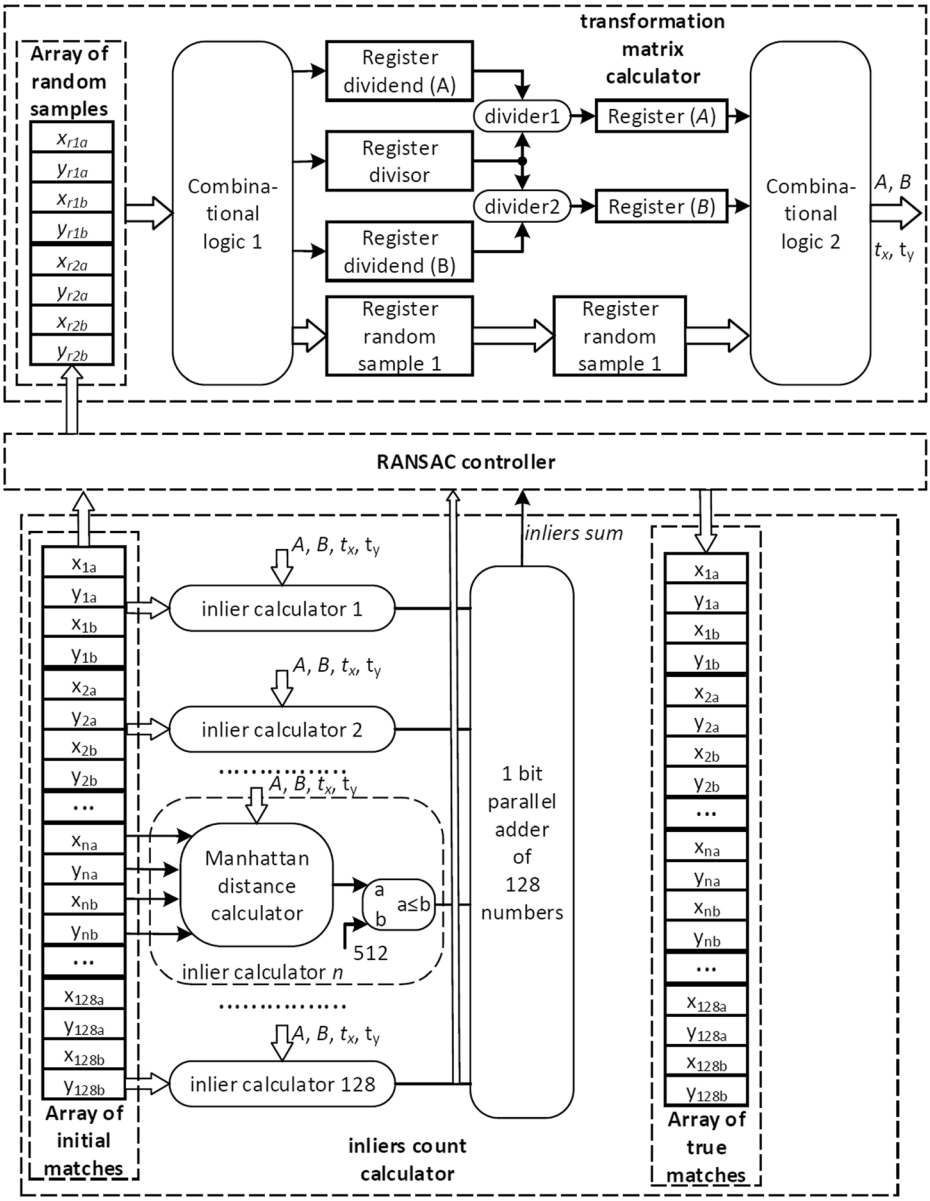


Fig. 4 The proposed RANSAC scheme

The "inliers count calculator" block is responsible for computing the number of inliers for the current transformation matrix. The "Array of initial matches" is an array of 128 elements in which all matches between two images have been stored. Each match inside the array is connected to combinational circuits, called "inlier calculator n", n = 1,2,…,128. Every inlier calculator circuit implements (8) and consists of the "Manhattan distance calculator" as well as a comparator. The output of the comparator indicates whether the corresponding match constitutes an inlier. All comparator outputs are summed in a parallel adder, producing the overall number of matches for the transformation matrix produced by the current sample pair.

The "RANSAC controller" block is responsible for reading samples from the "Array of initial matches" and storing to the "Array of random samples". In every clock cycle, a new sample pair is selected and after a propagation delay, the "*inliers sum*" provides the number of inliers. It should be noticed here that there is a delay of two clock cycles due to the registered data of the two dividers in the "transformation matrix calculator" block. Moreover, the "RANSAC controller" keeps track of the "*inliers sum*" signal and copies matches from the "Array of initial matches" to the "Array of true matches", when it is required. The "RANSAC controller" was implemented using state machines.

There is an issue that should be highlighted at this point. In real video frames, captured by a moving vehicle, objects in the foreground and in the background obey to transformations with different scaling, therefore different transformation matrices apply. When a random sample from the foreground is selected, it is quite possible to result in a transformation matrix, which renders true matches in the background outliers. The opposite is also possible. This phenomenon is getting stronger when an object is very close to the camera system and another one is very far. When addressing the problem of finding correspondences, such matches should not be rejected by RANSAC, since they are true matches. Therefore, we accept as true matches, inliers that come from different transformation matrices, under the condition that the resulting number of inliers is greater than the threshold of five matches. It is quite unlikely to locate five false matches that obey the same transformation. However, it could be quite common to detect five features that belong to a specific object and obey a unique transformation, due to its position in the scene.

According to [15], the minimum number of random samples that should be selected, can be obtained from (9).

$$R = \frac{\log(1-p)}{\log\left(1-(1-\varepsilon)^k\right)} \tag{9}$$

where p is the probability of at least one random sample to include matches that constitute inliers, ε is the proportion of outliers and k is the number of matches that can be inferred by one sample. In our case, $k = 2$. Assuming a reasonable proportion of outliers, approximately 40%, and setting $p = 0.99$, the minimum number of random matches is found to be 16. This means that from the moment the "Array of initial matches" has been filled with data, the RANSAC process will have been completed after 16 clock cycles. In the most demanding scenario of using all possible combinations, the required iterations are given by (10).

$$R_{\max} = \frac{m!}{(m-k)! \times k!} \tag{10}$$

where m is the number of matches and k = 2. For m = 128, RANSAC lasts for 8128 clock cycles.

5 Evaluation of the proposed architecture

The proposed architecture was evaluated in terms of accuracy, resource usage and execution speed. Furthermore, the development board DE2i-150 from Terasic, based on Cyclone IV technology, was used to verify the functionality of the system.

5.1 Matching accuracy

In order to evaluate the matching accuracy, the *recall* measure with respect to the *false rate* was used. *Recall* is defined as in (11), while *false rate* is defined as in (12). In (11), the overlapping features are defined as those which their Manhattan distance is less than 2 pixels, when the features of the first image are projected with homography to the second. The aforementioned criterion is widely used in the literature [24, 28, 31] and is considered as a safe way to draw conclusions in matching accuracy evaluation.

$$recall = \frac{true\_matches}{overlapping\_features} \tag{11}$$

$$false\_rate = 1 - precision = \frac{true\_matches}{total\_matches} \tag{12}$$

In general, the matching capability of the system depends on the adopted SIFT descriptor scheme. A detailed evaluation of the hardware optimized SIFT descriptor, which can be used in conjunction with the present matcher/RANSAC architecture, was presented in [28]. The use of moving window, in order to reduce comparisons between features, affects the matching accuracy. A large moving window provides higher *recall* values in comparison with a smaller one. Moreover, the number of detected features in each frame also affects matching. Higher numbers of detected features reduce the matching ability.

In order to evaluate the matching accuracy we fed the hardware pipeline with image data and executed the processing steps in an accurate ModelSim simulation of the datapath. We used a set of 20 images from the Middlebury data set [25]. Each image was transformed by a scale factor of 0.8. The shrunken images represented the current frame, while the initial images represented the previous frame. This procedure simulates backwards translation of the robotic vision system. SIFT was applied to the initial image and the extracted descriptors were saved in a memory initialization file. At the beginning of the simulation, the testbench reads descriptor data from the RAM which is pre-loaded with the previously saved memory initialization file and stores it to the "descriptor FIFO buffer". Subsequently, the testbench reads the pixel data of the current frame in a streaming manner from another file that is stored in the computer and saves the matches to the results file. In a next step, the results file is entered to a computer program, where the validation of the matches is examined. Since the image transformation is known, the computer software is able to identify all true and false matches.

The architecture was developed so that the active moving window is 128. Using that moving window, satisfactory matching ability is provided when the number of features is up to 600. The accuracy is also acceptable when the number of features approaches to 800. When the number of features is higher than 1000, then the recall degrades more but the architecture can still provide a considerable number of true matches. In Fig. 5, typical *recall* values with respect to *false rate* are presented, as obtained from the Cones image [25]. The tuning

parameters of SIFT algorithm were configured so that the number of detected features is 400, 600, 800, 1000 and 1200 respectively. In this typical graph, *recall* reaches almost a value of 0.3 when 800 features are detected. Considering that the repeatability of the SIFT extraction module is greater than 0.7 [28], the number of overlapping (repeatable) features is more than $0.7 \cdot 800 = 560$. This results in 168 true matches at least, which is sufficient for several robotic vision applications.

The matches file is also used in order to evaluate the RANSAC module. In the simulation step, the testbench filled the "Array of initial matches" by reading the matches file between two images, which is produced previously in the matching step. After the execution of the algorithm, the testbench produced an output file containing the inliers. Inliers were input to the computer program, by which they were drawn on the screen, together with the corresponding images, for visual verification. The output of the RANSAC module was compared against the expected inliers. In this case, all true matches were recognized and all false matches were rejected. When the transformation is artificial, all pixels verify the same transformation matrix. As a result, the phenomenon of different transformation matrices between pixels in the background and in the foreground does not apply.

Subsequently, real image pairs were captured, with arbitrary transformations. In Figs. 6 and 7, results from real-world images are depicted. Figure 6 shows the initial matches produced by the SIFT matching step, while Fig. 7 shows the matches kept by the proposed RANSAC hardware module. In this pair, most visually established true matches were kept, while all visually established false matches were rejected.

5.2 Resource usage

In Table 1, the required resources from the FPGA chip are presented. Each main module of the design is presented separately. The complete architecture is able to fit in a Cyclone IV FPGA device. The optimized SIFT matcher module demands only 1/3 of the LUTs of the overall design. This is due to the adopted multiplexing scheme. A similar multiplexing scheme could be also applied to the RANSAC module so that a further reduction of the required resources would be achieved. This reduction could make the FPGA chip capable to fit additional circuits, as the SIFT extraction module. Therefore, the complete architecture of SIFT extraction, SIFT

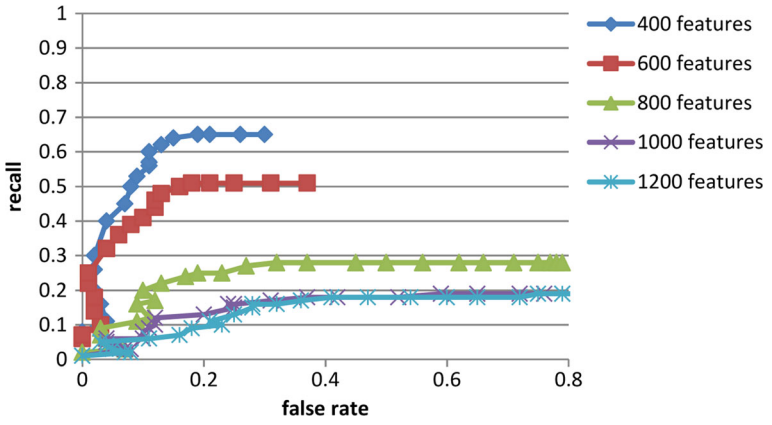


Fig. 5 *Recall* with respect to *false rate* for the Cones image

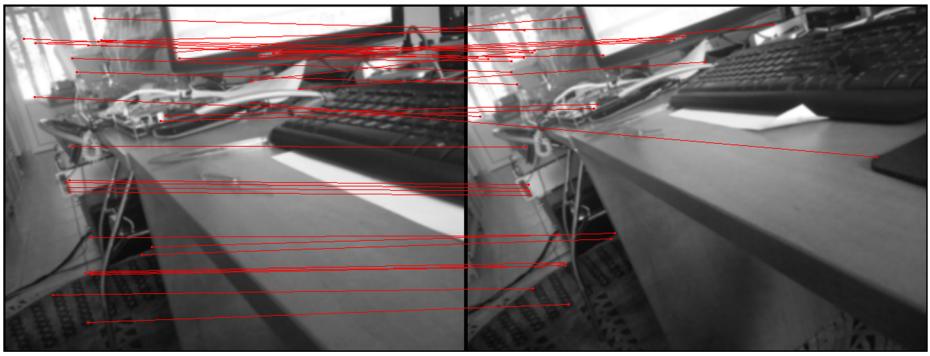


Fig. 6 Real images with initial matches: 34 matches were found by the SIFT matcher

matching and RANSAC would fit to a mid-range FPGA chip, which is considerer as a future work.

We should notice here that the combined SIFT matcher – RANSAC architecture is independent on the image resolution as it concerns the required resources. It deals only with the number of extracted features of a frame. For as long the number of extracted features are less than 2048 so that they can be stored in the "descriptor FIFO buffer", the SIFT matcher will work effectively no matter what the frame resolution is. Even if more than 2048 features is possible to be detected, the matcher scheme could be built with larger "descriptor FIFO buffer", increasing the RAM requirements from the FPGA. However, care should be taken in such a choice. If the number of features is increased excessively, the accuracy of the matcher will be decreased and a larger active moving window will be required. The RANSAC scheme is also independent of the image resolution and it was designed with the limitation of a maximum of 128 matches. The image resolution affects only the SIFT module block. In [28], details about how the resolution affects the SIFT module are presented. In general, let us notice that the number of columns affects the required RAM resources of the design.

5.3 Execution speed and real time performance discussion

The proposed architecture was built using the Quartus Prime software. The TimeQuest Timing Analyzer was employed in order to determine the maximum propagation delay inside the architecture. The design uses two clock signals, which are the *clk* and the *clk8*. The latter is

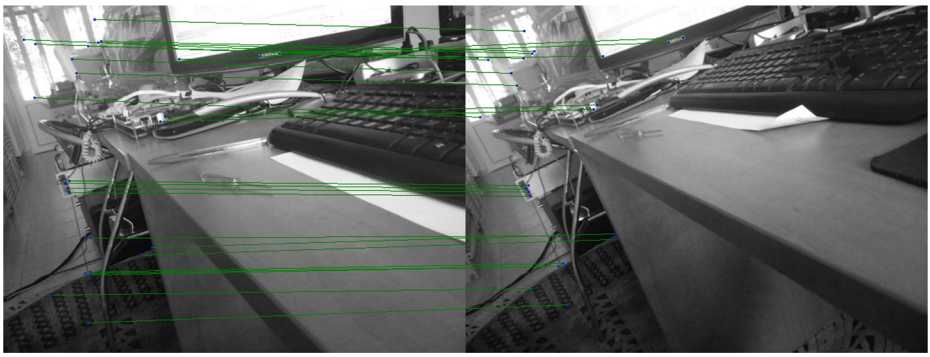


Fig. 7 Real images with extracted number of inliers: 26 matches were established as true matches by RANSAC

Table 1 Required resources (Cyclone IV EP4CGX150DF31C7)

Module	LUTs	Registers	Multipliers 9 × 9 bits	RAM (bits)
SIFT matcher	51,068	13,646	0	1,737,522
RANSAC	90,326	11,430	[a] 528	5632
Complete architecture	141,394	25,076	[a] 528	1,743,154
	(94%)	**(17%)**	**(73%)**	**(26%)**

[a] Half of the required multipliers are implemented using the dedicated multiplier circuitry and the rest using the available logic elements (LE)

eight times faster than the former. As a consequence, this should be taken into consideration in the search for the maximum clock frequency. Each module was analyzed separately. The maximum delay in the SIFT matcher module was found 10 ns. Thus, the maximum applied frequency is 100 MHz. This module is clocked by both *clk* and *clk8* clocks. This means that the *clk8* can be 100 MHz, while the *clk* can be 12.5 MHz. In the RANSAC module, the maximum propagation delay was found 66 ns and this means that the maximum applied frequency is 15 MHz. The *clk* clock is used for this module so the applied clock is limited to 12.5 MHz.

Considering that the *clk* clock is 12.5 MHz, the supported frame processing rate by the architecture is about 40fps, when images with resolution 640 × 480 pixels are processed. The frame rate is determined by the SIFT extraction module. On each *clk* cycle, one pixel is read. The required time to detect, extract and match a pixel (if it constitutes a feature) is 80 ns when *clk* is 12.5 MHz. In order to process the complete frame, the required time is about 640·480·80 ns = 24.5 ms when the frame has resolution 640 × 480. If the resolution was 1280 × 720, the required time to process the frame would be about 73 ms, meaning that the system would have lost its real-time performance. The execution speed is dependent on the low speed *clk* clock. The multiplexing scheme affects the frame rate since it constraints the low speed clock. Since the multiplexer is 8 to 1, the high speed clock should be 8 times faster than the *clk*. For that reason it is named *clk8*. If the multiplexing scheme was 4 to 1, the high speed clock should be 4 times faster than the low speed clock. In that case, the constraint for the *clk* comes from the RANSAC circuit. It should be 15 MHz, which means that the supported frame rate could be 48fps for images with VGA resolution. However, there is always the option to use a different clock to the RANSAC circuit and the SIFT module clock. In such a case, the low speed clock could be 25 MHz, meaning that the maximum propagation delay could be 40 ns. This has been analyzed in detail in [28, 30]. Considering VGA frames as well as multiplexer 4 to 1, the architecture could support 81fps. If the multiplexing scheme was 2 to 1, then the constraint between the two speed clocks would be even looser. However, schemes with multiplexers of 4 to 1 and 2 to 1 cannot fit in Cyclone IV devices. If a multiplexing Scheme 16 to 1 was used, the high speed clock should be 16 times faster. As a result, the low speed clock should be 6.25 MHz. That would lead to a supported frame rate at 20fps, meaning that the real-time performance requirements are not met. In conclusion, the final selection of a multiplexer 8 to 1 and the low speed clock at 12.5 MHz seems to be the optimum choice.

As it has been already mentioned, the proposed vision system uses previously published work [28] for SIFT extraction and targets robotic vision applications. The vision system is going to be used in a mobile vehicle which is moving in smooth terrain. Furthermore, the vehicle is not intended to be moving fast. For these conditions a speed about 30fps is considered sufficient for real-time operation.

In conclusion, we would say the factors that affect the real-time performance are the low speed clock *clk*, the multiplexing scheme and the image resolution. In order to improve performance, the *clk* clock can be increased by reducing the maximum propagation delay in the datapath. For example, asynchronous dividers have been employed to perform calculations in the RANSAC module as well as in SIFT extraction module [28]. The deviders are responsible for the maximum propagation delay of these circuits. Optimization of asynchronous divisions could significantly increase the supported frame rate. Another way to speed up the architecture would be to host it in last generation technology FPGAs, which are faster and consists of more resources. The selection of multiplexing scheme of 4 to 1 or 2 to 1 in conjunction with a more advanced FPGA could also offer significant improvement in performance. In such a case, where the plenty of hardware resources are available, the multiplexing scheme could not be adopted [30]. However, the power of this design is that it can be implemented using mid-range FPGA devices.

5.4 Verification and power consumption

The Cones image was used for the verification of the proposed system. A shrunken image was produced from the initial image and the feature descriptors of each image were extracted by the hardware block described in [28] and were stored in a file. The descriptors of the initial image were saved in a memory initialization file and loaded to the "descriptor FIFO buffer" by a starting process. The descriptors of the shrunken image were saved to a second memory initialization file and were read in order to feed the SIFT matcher circuits. The second initialization file was used since the Cyclone IV chip does not suffice to host the SIFT extraction module concurrently with the SIFT matcher and RANSAC. Therefore, for verification purposes the SIFT extraction module was replaced by RAM, in which the descriptors are preloaded. In Fig. 8, the system is illustrated using the DE2i-150 development board in operation. The true matches that are output of the RANSAC module are marked with crosses.

The power consumption was estimated using the PowerPlay Early Power Estimator tool for Cyclone IV. It was difficult to measure the power consumption directly from the chip since the DE2i-150 development board consists of several other external circuits that contribute to the overall consumption. The power consumption was estimated to be about 2 W. The use of the high speed *clk8* clock in the architecture increases the dynamic consumption in general. On the other side, the use of the multiplexing scheme and consequently, the reduction of the employed LE from the FPGA device reduce the static consumption.

5.5 Comparison with other systems

It is not safe to have a direct comparison with other systems presented in the literature. Different image sizes, feature types, target applications, octaves/scales configuration are met in a variety of architectures and as a result it is very difficult to draw safe conclusions. We have limited our comparison only to those papers which present a hardware matcher, regardless of the adopted features type. In Table 2, a rough comparison with other systems is presented. In comparison with our previous work [30], the present architecture outperforms the former in resource usage maintaining satisfactory matching accuracy. This is achieved by using the

Table 2 Comparison with other systems

	[31]	[7]	[30]	Proposed
Image size	1280 × 720	1024 × 1024	640 × 480	640 × 480
Configuration	[a] Sd + BD + Bm	[a] H + CC	[a] Sd + SD + Sm + R	Sm + R
Octaves/scales	2/6	–	¼	1/4
Technology used	Virtex-5	Virtex-4	Stratix IV	Cyclone IV
LUTs	18,437	20,576	494,201	51,068 + 90,326
Registers/FF	13,007	4733	105,423	13,646 + 11,430
DSP/Multipliers	52	[b] 0	960	[c] 528
RAM (Kbits)	4932	[d] 630	1886	1697 + 5.5
Power Consumption	4.5 W	2 W	4 W	2 W
Frame rate	[e] 60fps	33fps	[f] 155fps	40fps

[a] Sd: SIFT detector, SD: SIFT descriptor, Sm: SIFT matcher, BD: BRIEF Descriptor, Bm: Brief matcher, R: RANSAC, H: Harris detector, CC: cross correlation metric used in matching

[b] Muliplications are performed without the use of DSPs

[c] Half of the required multipliers are implemented using the dedicated multiplier circuitry and the rest using the available logic elements (LE)

[d] The authors used 35 Block RAMs

[e] This frame rate is supported when the number of features does not exceed 2000

[f] This frame rate is supported considering that the descriptor processing time is 21 ns for Stratix IV FPGAs

Fig. 8 Verification of the proposed architecture using the DE2i-150 development board

descriptors multiplexing scheme. The supported frame rate is reduced by half; however the system maintains its real-time operation. In [7, 31] other feature types than SIFT are used in the matcher. In [10], the BRIEF descriptor is used for matching which leads to a better usage of the FPGA resources but it also degrades the accuracy of the matcher. In [18], no SIFT features are used at all. Harris corner detector and cross correlation metric are used in the design. Let us note that the matcher presented here and in [30] is the first fully paralellized SIFT matcher implemented in hardware.

6 Conclusion

In this paper, an FPGA-based architecture for real-time SIFT matching and RANSAC algorithm for robotic vision applications is presented. A moving window of 16 elements is employed in order to reduce the concurrent comparisons between features of previous and current frames. A multiplexing scheme is used to increase the active moving window to 128, achieving satisfactory matching ability. The architecture uses 2 clocks, the fast *clk8* clock and the slower by 8 times *clk* clock. On each new detected feature, a match is produced in one *clk* cycle. The RANSAC algorithm has been also implemented and the overall architecture is able to fit in mid-range FPGA devices. Image transformations supported by the RANSAC module are scaling, translation and rotation. Each RANSAC run lasts for as many *clk* cycles as the number of the random samples. The maximum supported frequencies for *clk* and *clk8* clocks are 12.5 MHz and 100 MHz respectively, considering Cyclone IV technology. The proposed system is capable of processing 40fps, meeting real-time requirements.

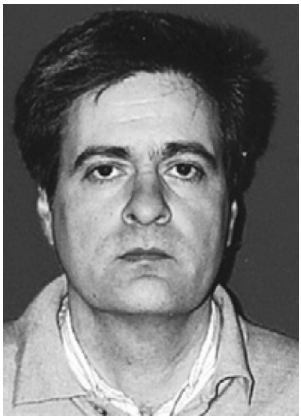
References

1. Alahi A, Ortiz R, Vandergheynst P (2012) FREAK: Fast Retina Keypoint. In: 2012 I.E. Conf. Comput. Vis. Pattern Recognit. IEEE, pp 510–517
2. Bay H, Tuytelaars T, Van Gool L (2006) SURF: Speeded Up Robust Features. In: Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics). Berlin, Heidelberg, pp 404–417
3. Bay H, Ess A, Tuytelaars T, Van Gool L (2008) Speeded-Up Robust Features (SURF). Comput Vis Image Underst 110:346–359. doi:10.1016/j.cviu.2007.09.014
4. Boulekchour M, Aouf N, Richardson M (2014) Robust L∞ convex optimisation for monocular visual odometry trajectory estimation. Robotica:1–20. doi:10.1017/S0263574714001829
5. Calonder M, Lepetit V, Ozuysal M et al (2011) BRIEF: Computing a Local Binary Descriptor very Fast. IEEE Trans Pattern Anal Mach Intell 34:1281–1298. doi:10.1109/TPAMI.2011.222
6. Condello G, Pasteris P, Pau D, Sami M (2013) An OpenCL-based feature matcher. Signal Process Image Commun 28:345–350. doi:10.1016/j.image.2012.06.002
7. Di Carlo S, Gambardella G, Prinetto P et al (2015) SA-FEMIP: A Self-Adaptive Features Extractor and Matcher IP-Core Based on Partially Reconfigurable FPGAs for Space Applications. IEEE Trans Very Large Scale Integr Syst 23:2198–2208. doi:10.1109/TVLSI.2014.2357181
8. Dohi K, Hatanaka Y, Negi K, et al (2012) Deep-pipelined FPGA implementation of ellipse estimation for eye tracking. In: 22nd Int. Conf. F. Program. Log. Appl. IEEE, pp 458–463
9. Dung L-R, Huang C-M, Wu Y-Y (2013) Implementation of RANSAC Algorithm for Feature-Based Image Registration. J Comput Commun 1:46–50. doi:10.4236/jcc.2013.16009
10. Fassold H, Rosner J (2015) A real-time GPU implementation of the SIFT algorithm for large-scale video analysis tasks. Proc SPIE - Int Soc Opt Eng. doi:10.1117/12.2083201
11. Fischler MA, Bolles RC (1981) Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. Commun ACM 24:381–395. doi:10.1145/358669.358692
12. Fürntratt H, Rosner J, Stiegler H, Fassold H (2013) GPU-accelerated SIFT descriptor matching. GPU Technol. Conf

13. Gentle JE (2007) Matrix Transformations and Factorizations. Matrix Algebr Theory, Comput Appl Stat. doi:10.1007/978-0-387-70873-7
14. Haiyang L, Hongzhou H, Yongge W (2013) A Fast Image Matching Algorithm Based on GPU Parallel Computing. Inf Technol J 12:1449–1453. doi:10.3923/itj.2013.1449.1453
15. Hartley R, Zisserman A (2004) Estimation - 2D Projective Transformations. In: Mult. View Geom. Comput. Vis. Cambridge University Press, pp 87–127
16. Hidalgo-Paniagua A, Vega-Rodríguez MA, Pavón N, Ferruz J (2014) A Comparative Study of Parallel RANSAC Implementations in 3D Space. Int J Parallel Prog 43:703–720. doi:10.1007/s10766-014-0316-7
17. Jiang J, Li X, Zhang G (2014) SIFT Hardware Implementation for Real-Time Image Feature Extraction. IEEE Trans Circuits Syst Video Technol 24:1209–1220. doi:10.1109/TCSVT.2014.2302535
18. Kapela R, Gugala K, Sniatala P et al (2015) Embedded platform for local image descriptor based object detection. Appl Math Comput 267:419–426. doi:10.1016/j.amc.2015.02.029
19. Ke Y, Sukthankar R (2004) PCA-SIFT: A more distinctive representation for local image descriptors. Proc. IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit. 2
20. Leutenegger S, Chli M, Siegwart RY (2011) BRISK: Binary Robust invariant scalable keypoints. In: 2011 Int. Conf. Comput. Vis. IEEE, Barcelona; Spain, pp 2548–2555
21. Liu H, Shen H (2014) Application of improved SIFT algorithm on stitching of UAV remote sensing image. Bandaoti Guangdian/Semiconductor Optoelectron 35:108–112
22. Lowe DG (1999) Object recognition from local scale-invariant features. In: Proc. IEEE Int. Conf. Comput. Vis. IEEE, Kerkyra, Greece, pp 1150–1157
23. Lowe DG (2004) Distinctive Image Features from Scale-Invariant Keypoints. Int J Comput Vis 60:91–110. doi:10.1023/B:VISI.0000029664.99615.94
24. Mikolajczyk K, Schmid C (2005) Performance evaluation of local descriptors. IEEE Trans Pattern Anal Mach Intell 27:1615–1630. doi:10.1109/TPAMI.2005.188
25. Scharstein D, Szeliski R (2003) High-accuracy stereo depth maps using structured light. Proc. IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit. 1
26. Song H, Xiao H, He W, et al (2013) A fast stereovision measurement algorithm based on SIFT keypoints for mobile robot. In: 2013 I.E. Int. Conf. Mechatronics Autom. IEEE, Takamastu, Japan, pp 1743–1748
27. Tang JW, Shaikh-Husin N, Sheikh UU (2013) FPGA implementation of RANSAC algorithm for real-time image geometry estimation. In: 2013 I.E. Student Conf. Res. Dev. IEEE, pp 290–294
28. Vourvoulakis J, Kalomiros J, Lygouras J (2016) Fully pipelined FPGA-based architecture for real-time SIFT extraction. Microprocess Microsyst 40:53–73. doi:10.1016/j.micpro.2015.11.013
29. Vourvoulakis J, Lygouras J, Kalomiros J (2016) Acceleration of RANSAC algorithm for images with affine transformation. In: 2016 I.E. Int. Conf. Imaging Syst. Tech. IEEE, pp 60–65
30. Vourvoulakis J, Kalomiros J, Lygouras J (2017) FPGA accelerator for real-time SIFT matching with RANSAC support. Microprocess Microsyst 49:105–116. doi:10.1016/j.micpro.2016.11.011
31. Wang J, Zhong S, Yan L, Cao Z (2014) An Embedded System-on-Chip Architecture for Real-time Visual Detection and Matching. IEEE Trans Circuits Syst Video Technol 24:525–538. doi:10.1109/TCSVT.2013.2280040

John V. Vourvoulakis received his Diploma in 2002 and his MSc Degree in 2004 from the department of Electrical and Computer Engineering of Democritus University of Thrace. His research interests are in the field of embedded systems based on FPGAs and/or microcontrollers including hardware design and low level

programming. He is also working as adjunct teaching staff in the Technological Educational Institute of Lamia, Greece.



John A. Kalomiros received the degree of Physics, a MS degree in Electronics and a PhD from the Aristotle University of Thessaloniki, Greece. He also received a PhD on the development of vision systems for robotic applications from the Democritus University of Thrace. His research interests include electronics, hardware design and machine vision. He is associate Professor at the Department of Informatics Engineering in the Technological Educational Institute of Central Macedonia, Greece.

John N. Lygouras received the Diploma degree and the Ph.D. in Electrical Engineering from the Democritus University of Thrace, Greece in 1982 and 1990, respectively, both with honors. From 2012 he is a Professor at the Department of Electrical & Computer Engineering in DUTh. His research interests are in the field of robotic systems trajectory planning and execution. His interests also include the research on analog and digital electronic systems implementation and controller design for underwater remotely operated vehicles (ROVs).