A Hardware Accelerated Scale Invariant Feature Detector for Real-time Visual Localization and Mapping

Zunquan Zhou, Rendong Ying, Rongdi Sun, Zhenqi Wei, Ke Jin, Peilin Liu
Department of Electronics
Shanghai Jiao Tong University
Shanghai, China
zqzhou.363@gmail.com

***Abstract*—Scale Invariant Feature Transform (SIFT) has drawn attention in the field of computer vision, recently. SIFT has been adopted in many visual localization and mapping applications, for its robustness to scale, rotation and illumination changes. However, the high computational cost limits its use in practical scenarios. In this paper, we present a real-time FPGA-based hardware accelerator of SIFT. The design is composed of two main parts: key-point detection component and feature generation component. The key-point detection component applies an octave-interleaved scale-parallel pipeline structure, as a tradeoff between frame rate and resource consumption. The feature generation component works in task-level burst mode for each key-point. The buffer together with buffer management logic enables quasi-parallelism between the two components, and also enables task-level quasi-parallelism between main orientation generation and local descriptor generation in the feature generation component. Our proposal can perform feature extraction of 720p video with real-time efficiency of 42fps at a clock frequency of 100MHz.**

Keywords—visual localization and mapping; scale-invariant feature transform; hardware accelerator; real-time; octave-interleaved; scale-parallel; pipeline; task-level parallelism

I. INTRODUCTION

Image matching is a fundamental process of many computer vision applications, including object or scene recognition, motion tracking, visual localization and 3D scene reconstruction. The performance of these applications to some degree relays on high-quality feature description and matching. Among feature extraction algorithms, Scale-Invariant Feature Transform (SIFT) [1] is one of the most famous ones, due to its robustness. And SIFT has been recognized to be invariant to image scale and rotation, and to provide robust matching across a substantial range of affine distortion, change in 3D viewpoint, addition of noise, and change in illumination. Many visual localization and mapping applications have shown SIFT to provide reliable feature matching results [2, 3]. In addition, Random Sample Consensus (RanSAC [4]) based algorithm is used to filter out mismatched feature pairs. The remaining feature pairs can be regarded as correct matching results. And then Iterative Closed Points (ICP [5]) is used in registration of two feature point clouds.

However, many feature extraction algorithms, especially SIFT, can have many complex computations such as convolution, division, square root, arc-tangent, and exponential operations. For application such as Simultaneous Location and Mapping [6], and for platforms such as micro unmanned aerial vehicle (micro-UAV), usually low-power processor is provided and real-time algorithm is required. In contrast to CPU and/or GPU acceleration methods, FPGAs are more suitable for this kind of work, because of the huge parallel resources for processing and low-power consumption.

There have been many hardware accelerators for SIFT algorithm [7-12]. Most of them claim to be real-time, and the size of processed images keeps growing. But computational cost and processing time also grow sharply with the total number of pixels. In most cases, real-time requirement is hard to realize, so the algorithm has to be modified and simplified.

The SIFT accelerator in [7] is one of the close-to-original implementations. It can process a VGA image within 33ms if the number of feature points is fewer than 890. We have learned from their project. The system is also divided into two components. However, the key-point detection component has to stall when the feature generation component keeps processing. The processing time of the key-point detection is only 3.4ms for a VGA image. But for larger images, like 720p image, the processing time also grows with the number of pixels and key-points. So it is not reasonable to stop either component. Besides, the average processing time of local descriptor generation is more than 3000 clock cycles. And the local descriptor generation can start only after getting the key-point's main orientation. Since the first component is designed in highly parallel, the second component becomes bottleneck in the system.

There has been another effort to achieve a faster architecture of SIFT hardware accelerator. SIFT hardware implementation in [8] also divides the system into the same two components, while two components process different frames of image at the same time. In this way, task-level parallelism between the two components can be achieved, at the price of double buffer size. The processing time is proportional to the image size. Besides, they use a modified

978-1-5090-2879-5/16/$31.00 ©2016 IEEE

descriptor generation method: main orientation is generated from 16-direction bins; the data window is divided into 16 sub-regions along the 16 directions and one in the center; and descriptor is generated by reordering the sub-regions and the histograms of each sub-region. They accomplish task-level parallelism between main orientation generation and descriptor generation by modifying the original algorithm. Besides, the data window for descriptor generation is 15x15 pixels, which is far smaller than the suggested size from Lowe.

There are many other accelerators trying to achieve real-time performance with algorithm modification [9-12]. The feature generation component is implemented with a DSP processor in [9], or replaced by binary robust independent elementary feature (BEIEF[13]) in [10]. A layer parallel SIFT is proposed in [11], based on a box kernel from integral image rather than Gaussian filter. A NIOS processor is used to generate features in [12]. Besides, window size is rather small in these solutions.

In this paper, we propose a real-time FPGA hardware accelerated architecture of SIFT algorithm for 720p video. It is close to the original SIFT algorithm. The architecture is divided into two main components: the key-point detection (KPD) component and the feature generation (FG) component. The KPD component exploits high-level parallelism and is designed in octave-interleaved pipeline structure. The processing time equals to the total number of pixels (excluding the delay in buffers and logics). The FG component works in task-level burst mode. It starts once a key-point is detected and the window data have been fully buffered. The data are still processed in serial. But the buffer management logic between the two components enables the following properties. First, the KPD component runs independently from the FG component. Quasi-parallelism can be achieved between them. Second, the two components work on asynchronous clocks. The FG component is speeded up by faster clock to avoid memory overriding and key-point skipping as much as possible. Third, the main orientation generation (MOG) module and the local descriptor generation (LDG) module work in task-level quasi-parallel for different key-points.

The structure of this paper is organized as follows. Section II reviews the SIFT algorithm proposed by Lowe [1]. Section III presents the architecture of our proposed SIFT accelerator. Section IV shows the experimental results, and Section V concludes this paper.

II. SIFT ALGORITHM

The SIFT algorithm is divided into five main computational procedures. Details are described as follows.

A. Construction of Gaussian Pyramid

Gaussian pyramid is constructed from convolving images with Gaussian filters. Firstly, we convolve a given source image, denoted as I(x, y), with a series of Gaussian filters, denoted as G(x, y, σ). The results are a series of Gaussian blurred images, denoted as L(x, y, σ) shown in (1), and together they form the so-called Gaussian pyramid.

$$L(x,y,\sigma) = G(x,y,\sigma) * I(x,y) \tag{1}$$

Where

$$G(x,y,\sigma) = \frac{1}{2\pi\sigma^2}\exp\left[\frac{-(x^2+y^2)}{2\sigma^2}\right] \tag{2}$$

And σ of the Gaussian filters is shown in (3).

$$\sigma = 2^i k^s \sigma_0; i = 0,1,...,o-1; s = 0,1,...,S+2; k = 2^{1/S} \tag{3}$$

Where o is the number of octaves, S is the number of scales.

The DoG image, denoted as D(x, y, σ), is defined as the difference of adjacent Gaussian blurred images, as shown in (4).

$$D(x,y,\sigma) = L(x,y,k\sigma) - L(x,y,\sigma) \tag{4}$$

The construction of DoG pyramid can be illustrated as Fig. 1. The number of octave is 2, and the scale is 3. So there are 2 octaves of Gaussian and DoG images, while the Gaussian octave has 6 scales of image and the DoG octave has 5 scales. In the first octave, Gaussian blurred images is denoted from L_0 to L_5. And DoG images is denoted from D_0 to D_4.

The second octave is down-sampled from the fourth image in the first octave. Gaussian filtering and differentiation is applied in the same way.

B. Key-point detection

By using the data from DoG pyramid, key-points can be detected from three adjacent Dog images. Firstly, a local maximum or local minimum, denoted as local extremum, is chosen out of its 26 neighboring pixels. Then each extremum candidate has to pass a procedure of stability test. After that, unstable extrema with low contrast or with strong edge response is rejected.

C. Gradient computation

The gradient histogram is used to generate feature for every key-point. So the gradient pixels from the neighborhood of the key-point in the corresponding Gaussian blurred image should be computed.

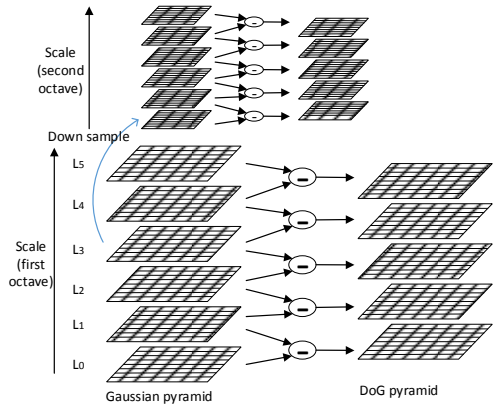


Fig. 1. Construction of Gaussian pyramid and DoG pyramid

The gradient of a given pixel (x, y) has two components: the gradient magnitude and orientation, denoted as (5) and (6).

$$m(x,y)=\sqrt{\left(L(x+1,y)-L(x-1,y)\right)^2+\left(L(x,y+1)-L(x,y-1)\right)^2} \tag{5}$$

$$\theta(x,y)=\arctan\left(\left(L(x,y+1)-L(x,y-1)\right)/\left(L(x+1,y)-L(x-1,y)\right)\right) \tag{6}$$

D. Main orientation generation

The main orientation for a given key-point is generated from the gradient histogram from neighborhood of the key-point. This neighborhood is denoted as MOG region. An orientation histogram is formed from the gradient of pixels within that region. The orientation histogram has 36 bins representing 36 directions around the key-point. The gradient magnitude of every pixel is weighted by a Gaussian-weighted filter, and then accumulated to the orientation histogram. The max bin in the orientation histogram corresponds to dominant directions of local gradients. The corresponding direction is chosen as main orientation for the key-point.

E. Local descriptor generation

A 128-D feature vector is generated from the gradient histogram of the key-point's neighborhood region, denoted as local descriptor for the key-point. The neighborhood region is named as LDG region, which is then divided into 4 x 4 =16 square sub-regions, according to the main orientation of the key-point. This process is shown in Fig. 2. The gradient orientation of each pixel inside the LDG region will be rotated by the main orientation, and the gradient magnitude will also be weighted by another Gaussian-weighted filter. This process makes the descriptor invariant to rotation. An orientation histogram is computed in every sub-region, while each histogram has eight orientation bins. Overall, above 16 histograms form a feature vector with 128 elements (4 x 4 x 8). In addition, the 128-D feature vector is then normalized to reduce the effects of illumination change.

III. Proposed Hardware Architecture

A. Overall System Architecture

The overall system architecture of proposed SIFT hardware accelerator will be introduced in this section. The architecture for the first octave is shown in Fig. 3.

The key-point detection component includes three sub-modules: 1) Gaussian filtering and DoG images computation, 2) the key-point detection, 3) the gradient orientation and magnitude computation. These three sub-modules are all designed in pipeline structure and exploit high level parallelism. The pixel data of a given gray image is streaming into the Gaussian convolution engines, and then Gaussian blurred pixel data and DoG pixel data is streaming out to the second and third sub-modules. The key-point detection module checks key-points in each scale and streams out the key-point flags, location of key-points, and the coordinate offset of location. The gradient computation module calculates both

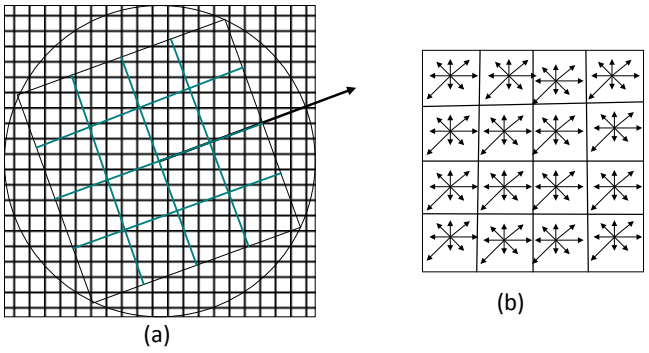


Fig. 2. Local descriptor generation. (a) Rotation and sub-region division. (b) Histogram generation in sub-regions.

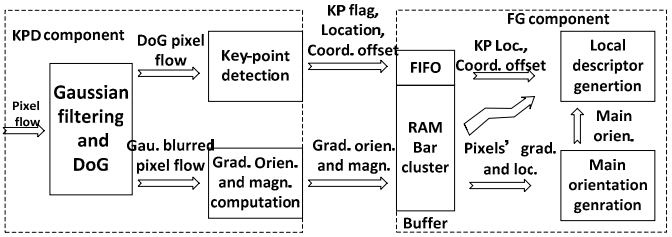


Fig. 3. Block diagram of overall system architecture

gradient orientation and magnitude for every pixel, and then streams out the gradients to the descriptor generation component.

The feature generation component is composed of three sub-modules: 1) the buffer and buffer management logic, 2) the main orientation generation, 3) the local descriptor generation. The buffer module takes responsibility for the storage of key-point information and the gradient of pixels, and the startup of the second or third module. This buffer logic is also described in section D. The second module generates the main orientation of every key-point, which is delivered to the third module and then buffered there. The third module generate the 128-D feature vector and packs the feature vector together with the location and orientation information of the key-point.

This paper uses a buffer structure same to segment buffer in [7]. We call it as RAM-bar cluster structure. Every sub-module in the KPD component, together with the buffer module in the FG component, is composed of a RAM-bar cluster as input buffer. In the first octave, the depth of RAM equals to the width of the image (1280 in this system). And the number of RAM bars equals to the least number of rows to be buffered for each sub-module. The size of RAM-bar clusters for the first octave is shown in Fig. 4.

The switching scheme of this RAM-bar cluster structure is described in [7]. There is a brief description. The input data flow is buffered to a given ram until the end of row (1280 pixels buffered), and a cyclic following RAM bar will take over. The input buffer logic takes control of this cyclic process. The ram bars can be accessed all at the same time, and a switching network is used for reordering the output data. This process is also shown in section C.

B. Two octave-interleaved structure

The implementation of the second octave is mostly the same to the first octave, except that the input pixels are down

sampled from the fourth scale of Gaussian blurred image in the first octave, and that the depth of ram bars is half-sized. In most hardware systems, the octaves work in parallel. We have implemented this structure, too. However, it is not that effective in resource consumption. Thus we propose a new octave-interleaved architecture shown in Fig. 5.

In this octave-interleaved architecture, every row of pixels from the second octave is processed after two rows from the first octave in the KPD component. Take the Gaussian filtering module in Fig. 6 as an example. In Fig. 6(a), two rows of pixels are streaming into the first octave's input buffer. The Gaussian filtering module is driven to stream out two rows of Gaussian blurred images for the first octave. At the same time, the down-sampled image from the fourth scale is stored in a FIFO. After that, in Fig. 6(b), one row of pixels from the second octave is accessed from the FIFO, and streaming into the second octave's input buffer. Pixels from the second octave are then streaming into the same Gaussian filtering module. But the Gaussian blurred images now belong to the second octave.

The processing time for one frame of image has increased by around a quarter, compared to the parallel solutions. But almost half logic resources for the key-point detection component have been saved. Besides, since the two octaves are interleaved, more processing time is spared for feature generation in each octave. In conclusion, the octave-interleaved architecture has reduced logic resource consumption and increased the feature capacity, at price of fps decrease.

C. *Gaussian Blur*

The 2D Gaussian filter kernel described in (2) can cover the whole image, theoretically. Instead, a window is used to mask the filters, and the window size varies with the value of σ. Since σ_0 is set to 1.6, and the filter factors are represented by Q16 fixed-point format, the window sizes are set to {9 x 9, 11 x 11, 13 x 13, 17 x 17, 21 x 21, 25 x25}.

Besides, due to the separability and the symmetry properties of the Gaussian filter kernel, the 2D filter kernel is broken into two 1D symmetrical filters [14], as denoted in (7).

$$G(x,y,\sigma)=\frac{1}{\sqrt{2\pi}\sigma}\exp\left(\frac{-x^2}{2\sigma^2}\right)*\frac{1}{\sqrt{2\pi}\sigma}\exp\left(\frac{-y^2}{2\sigma^2}\right) \quad (7)$$

The whole Gaussian filters are shown in Fig. 7. The ram-bar cluster structure makes it possible to conduct six Gaussian filtering concurrently. The data read out of RAM-bars is switched and then resized according to the special window size of each Gaussian filter.

In our proposed Gaussian filter architecture, the pixels are firstly convolved with the 1D filter along the y direction, and then the intermediate results are convolved with the 1D filter along the x direction. The block diagram of one 25 x 25 Gaussian filter is shown in Fig. 8. The process is described as follows. 1) pixels along the y direction are read out from the image RAM-bar bluster in parallel. And the pixels are switched, so that the center pixel is at the center of the 1D window mask. 2) if a pixel in the window mask falls outside the original image, the pixel will take the value of the

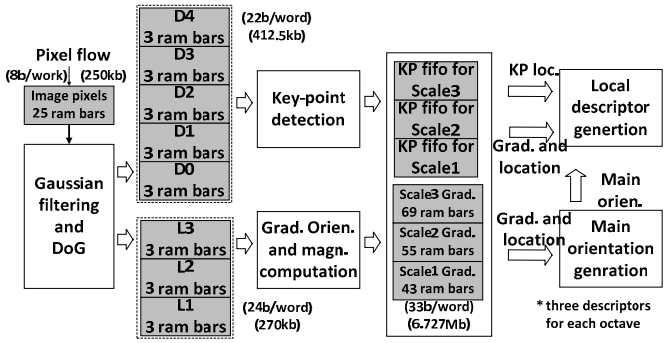


Fig. 4. Block diagram of buffers and computational modules

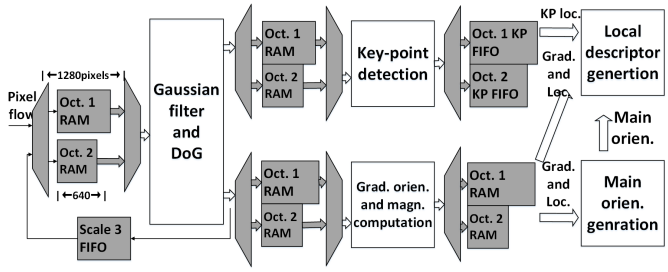


Fig. 5. Block diagram of octave-interleaved architecture for key-point detection component

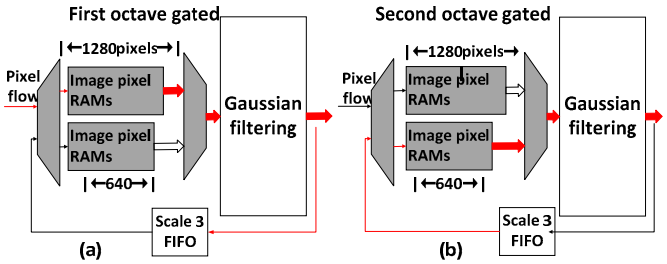


Fig. 6. Block diagram of octave-interleaved processes in Gaussian filtering module. (a) First octave is gated. (b) Second octave is gated.

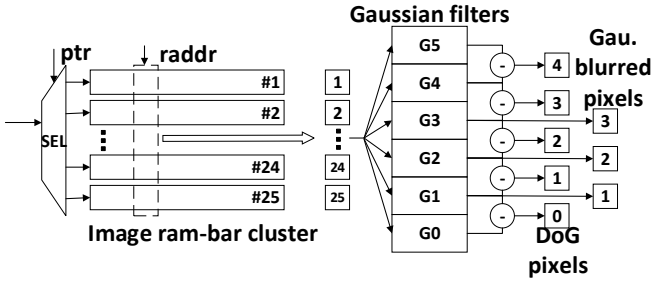


Fig. 7. Buffer management and data flow for Gaussian filtering

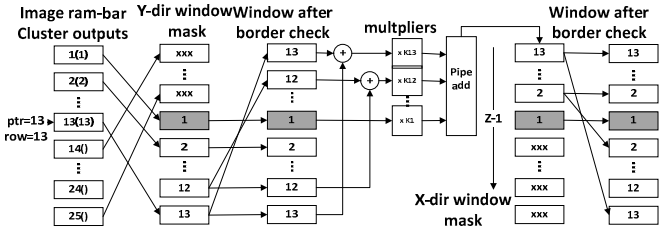


Fig. 8. Computational process for one Gaussian filter (25x25)

symmetric pixel in the window. 3) the symmetric pixels in the window are added up, and the sums are then multiplied with the corresponding filter factors. The results are added up. 4) results of y-direction filtering are streaming into the 1D window mask along the x direction. And pixels outside the

image border will also be revised in a same way of step 2. 5) x-direction filtering is the same to step 3.

The symmetric strategy along the image boundary helps to prevent sharp changes along the image borders, especially in the second octave.

D. Buffer Between Two Components

The buffer between the key-point detection component and the feature generation component serves an important role in system architecture. The buffer module consists of a RAM-bar cluster, FIFOs, and a control logic. This buffer separates the key-point detection component from the feature generation component. The key-point information is buffered in FIFOs. Gradients are streaming into the RAM bars. And no feedback is transmitted from the second component to the first one. Compared to the hardware accelerator described in [7], pipeline in the first component cannot be disturbed, whenever the second component starts working. In this way, the two main components work in quasi-parallel. The processing time per frame mainly depends on the KPD component in this solution. This buffer logic also simplifies the control logic between two components, at the risk of gradient information overridden by new coming ones. When the number of overridden pixels reaches a pre-set threshold, the key-point is skipped.

This proposed buffer logic also enables asynchronous clocks for the two components, so that the feature generation component can work on a higher frequency. In this way, more feature points can be generated. It also means less overriding and skipping cases.

Last but not least, the buffer logic can start the main orientation generation module and the local descriptor generation module simultaneously. In this way, the two sub-modules work in task-level quasi-parallel for two different key-points, and the processing time of main orientation is saved.

As mentioned above, this buffer logic introduces allowable errors, because pixels near the border of window can be overridden. These pixels usually do not contribute to the histogram or have little weight, as shown in Fig. 2. However, the performance decreases much if great number of key-points exceeds the system's capacity.

Fig. 9 shows that main orientation generation and local descriptor generation for different key-points can be processed simultaneously. The two sub-modules both have exclusive FIFO as key-point buffer. When the descriptor for key-point L is under processing, the main orientation for key-point M can be computed at the same time. Generally, for a given key-point, the main orientation generation finishes much earlier than the local descriptor generation. This strategy makes it possible that the descriptor generation module does not wait for main orientation results. We call this task-level quasi-parallelism.

IV. Experiment Results

In this section, we present the experiment results of our hardware accelerator for SIFT algorithm.

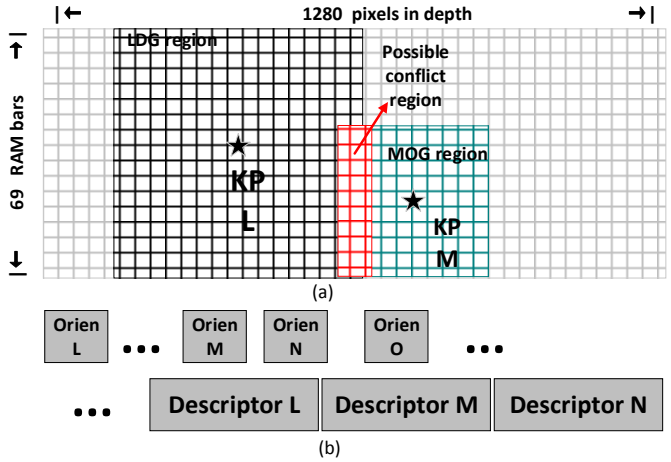


Fig. 9. Task-level quasi-parallelism between main orientation generation and local descriptor generation for two different key-points. (a) Parallel memory access (b) Task-level quasi-parallelism

The first part introduces matching results on a set of images. The second part is about the implementation results of our hardware accelerator.

A. Matching results

The hardware accelerator is compared with the software SIFT program in OpenCV 2.4.11. There are two octaves and three scales for each octave. The initial σ_0 is 1.6 and we do not double image size in the initial step. For extrema detection, the contrast threshold is set to 0.04 and the edge threshold equals to 10. Besides, key-points which are within 5 pixels next to image border are ignored.

Both software program and our hardware accelerator are tested with a set of benchmark images provided by Mikolajczyk and Schmid [15]. Firstly, the key-points in each image is detected and the descriptors are generated. Then, features from different images are matched. Finally, the correctly matched points are detected based on the RanSAC algorithm. Some bad situations are shown from Fig. 10 to Fig. 14. The matching results are shown in Table I.

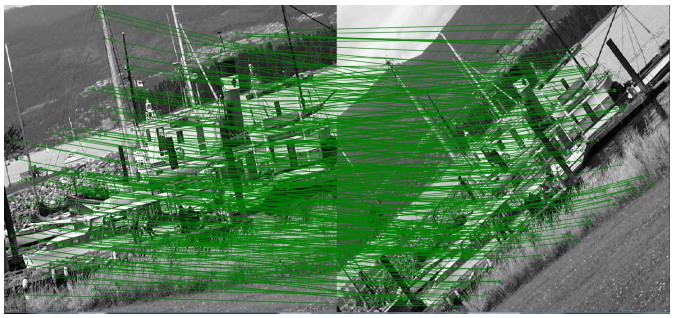
The performance degradation results mainly from the fix-point format, window size, the simplification of descriptor logic and the skipped key-points (exceeding the capacity).

The word length for main data is shown in Table II. The window size is almost the same as software program. The average error for Gaussian filtered images is around 0.2%. The average error for gradient orientation is around 0.4%, and the error for gradient magnitude is around 0.3%. In experiments, fix-point format and window size brings small errors to key-point location.

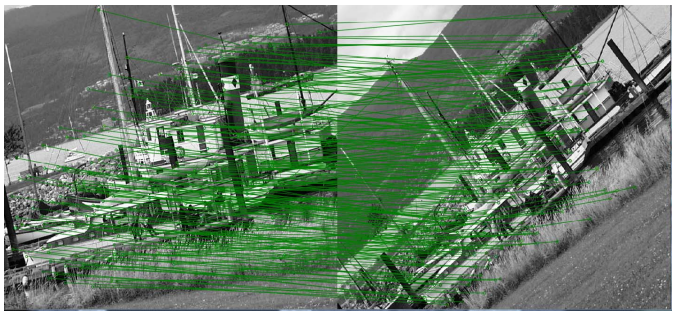
The number of features decreases in hardware accelerator, mainly because of skipped key-points. And there are much more skipped key-points in blurred images, as is shown in Fig. 12. In this case, the performance degrades much.

Theoretically, our architecture can achieve similar accuracy to the software program, if the main orientation generation module and the local descriptor generation module are well implemented. A simplified feature generator is implemented to

save logic resources, because a full implementation of local descriptor generator consumes 555K transistors in [16]. The performance degradation results mainly from the simplification of FG components.



(a)



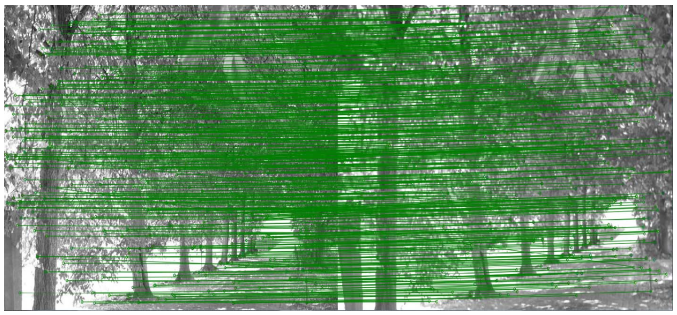
(b)

Fig. 10. Zoom and rotation. (a) software matching results. (b) hardware matching results.

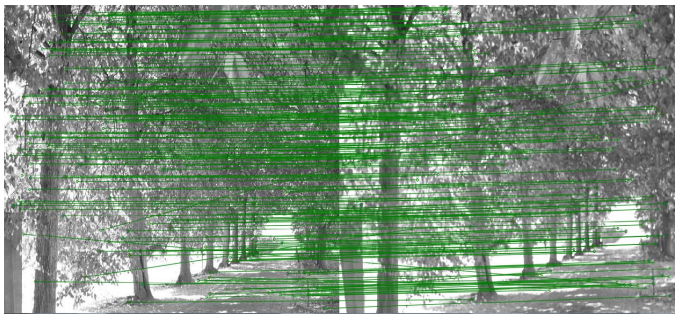
(a)

(b)

Fig. 11. Change of viewpoint. (a) software matching results. (b) hardware matching results.



(a)



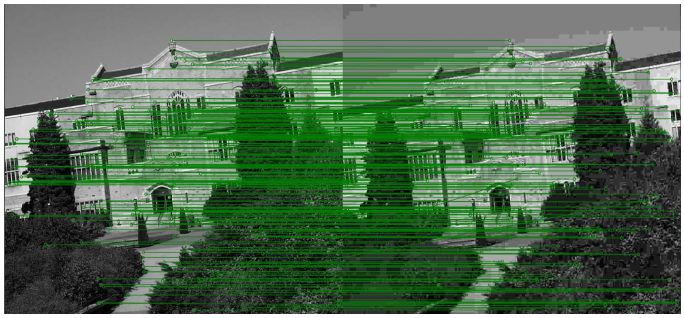
(b)

Fig. 12. Blur. (a) software matching results. (b) hardware matching results.

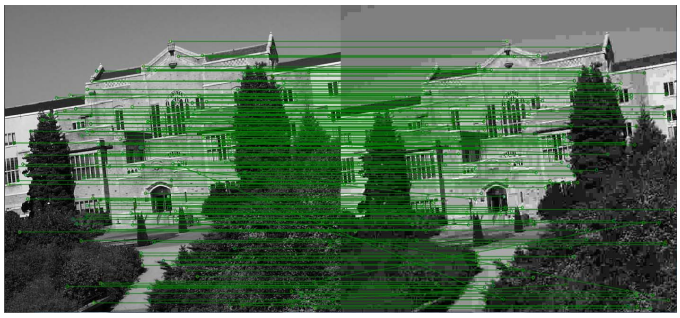
(a)

(b)

Fig. 13. Illumination. (a) software matching results. (b) hardware matching results.



(a)



(b)

Fig. 14. JPEG compression. (a) software matching results. (b) hardware matching results.

TABLE I. MATCHING RESULTS

Test scenario	Software(features)			Hardware(features)		
	Left	*Right*	*Match*	*Left*	*Right*	*Match*
Zoom and rotation	1040	885	366	1000	873	248
Change of viewpoint	761	951	176	716	935	88
Blur	2413	3438	521	1934	2172	212
Illumination	259	653	118	246	733	100
JPEG compression	898	973	301	715	909	203

TABLE II. FIX-POINT FORMAT FOR MAIN DATA

Main data	Word length (M Q N format)
Source image	8 Q 0
Gaussian filtered pixel	8 Q 16
DoG pixel	6 Q 16
Gradient orientation	9 Q 0
Gradient magnitude	8 Q 16
KP location (x-coord)	11 Q 0
KP location (y-coord)	10 Q 0
Main orientation	9 Q 0
Feature vector	12 Q 16

B. Implementation results

The hardware SIFT algorithm accelerator is implemented on Xilinx Virtex-7 690t device. The main implementation results are listed in Table III.

In this paper, we implement a SIFT algorithm similar to the original OpenCV implementation and the hardware implementation in [7]. Our architecture shares the same window size for Gaussian filtering and feature generation. In comparison, we propose an octave-interleaved architecture to save computational logics in key-point detection component. And a buffer management logic separates the two components and achieves quasi-parallelism between them. The task-level quasi-parallelism between main orientation generation and local descriptor generation also reduces feature generation time.

TABLE III. IMPLEMENTATION RESULTS

Items	Architecture in [7]	Our architecture
Resolution	640x480	1280x720
FPGA device	- (TSMC 0.18μm CMOS)	Virtex-7
Slice LUTs	- (1320K gate count)	97592
Slice registers		117889
DSPs		354
Working memory(Mbit)	5.73	11.46
Max freq for key-point detection	100MHz (operation freq.)	168.55MHz (50Mhz operation freq.)
Max freq for feature generation		179.18MHz (100Mhz operation freq.)
Fps	30	42
Max key-point capacity	890	1148, 743, 468 (for scale 1, 2, 3 in each octave)

The processing cycles for every key-point in each scale is proximately as follows (except for key-points near image border): 2000, 3100, 4900. And the maximum bearable key-points for each scale is around 1148, 743 and 468. This key-point capacity for one octave is acceptable for most cases.

V. CONCLUSION

Visual location and mapping applications have progressed with the improvement of hardware and software development. In order to achieve real-time performance in many micro vehicles, it's a promising solution to design a hardware accelerator for heavy computational algorithms. And SIFT has been shown as one of robust feature detection algorithm. In this paper, we present a hardware accelerator for SIFT algorithm. The system contains two main components: the key-point detection and the feature generation. The key-point detection component is designed in octave-interleaved scale-parallel pipelined architecture. The parallel and pipeline structure achieves highest throughput (one pixel each scale per clock cycle). The octave-interleaved structure makes it possible that both octaves share the same key-point detection logic. The buffer management logic between the two components enable quasi-parallelism between them. And task-level quasi-parallelism is achieved between main orientation generation and local descriptor generation. In this way, the processing time for main orientation is saved. The proposed hardware architecture is able to perform feature extraction at 42 fps for 720P video.

VI. ACKNOWLEDGMENT

This research work is supported by the Important National Science and Technology Specific Project of China under Grant No. 2016ZX03001022-006, the National Natural Science

Foundation of China under Grant Nos. 91438113, 61402283, 61573242, 61401501 and 61304225, the National Science and Technology Major Project under Grant No. GFZX0301010708, and the Shanghai Science and Technology Committee under Grant Nos. 16DZ1100402 and 15511105100.

REFERENCES

[1] D. G. Lowe, “Distinctive image features from scale-invariant key points,” Int. J. Comput. Vis., vol. 60, no. 2, pp. 91–110, Jan. 2004.

[2] P. Henry, M. Krainin, E. Herbst, X. Ren, and D. Fox, “RGB-D mapping: Using depth cameras for dense 3D modeling of indoor environments,” In the 12th International Symposium on Experimental Robotics (ISER), 2010.

[3] X. Li, W. Guo, M. Li, and C. Chen, “Generating Colored Pointcloud Under the Calibration between TOF and RGB Cameras,” Information and Automation (ICIA), 2013 IEEE International Conference on, pp. 483-488, Aug, 2013.

[4] MA fischler, and RC Bolles, “Random sample consensus: a paradigm for model fitting with application to image analysis and automated cartography,” Communication of the ACM,24(6),1981, pp.381-395.

[5] P. J. Besl, and N. D. McKay, “A method for registration of 3-D shapes. IEEE Transactions on Pattern Analysis and Machine Intelligence,” 14(2), pp. 239–256, 1992.

[6] A. J. Davison, N. D. Molton, I. Reid, and O. Stasse, “MonoSLAM: Real-time single camera SLAM,” IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI), 29(6), pp. 1052–1067, 2007.

[7] F. C. Huang, S. Y. Huang, J. W. Ker, and Y. C. Chen, “High-performance SIFT hardware accelerator for real-time image feature extraction,” IEEE Trans. Circuits Syst. Video Technol., vol. 22, no. 3, pp. 340–351, Mar. 2012.

[8] J. Jiang, X. Y. Li, and G. J. Zhang, "SIFT hardware implementation for real-time image feature extraction," IEEE Trans. Circuits Syst. Video Technol., vol. 24, no. 7, pp. 1209-1220, July 2014.

[9] S. Zhong, J. Wang, L. Yan, L. Kang, and Z. Cao, “A real-time embedded architecture for SIFT,” J. Syst. Archit., vol. 59, no. 1, pp. 16–29, Jan. 2013.

[10] J. Wang, S. Zhong, L. Yan, and Z. Cao, "An embedded system-on-chip architecture for real-time visual detection and matching," IEEE Trans. Circuits Syst. Video Technol., vol. 24, no. 3, pp. 525-538, July 2014.

[11] L.-C. Chiu, T.-S. Chang, J.-Y. Chen, and N. Y.-C. Chang, “Fast SIFT design for real-time visual feature extraction,” IEEE Trans. Image Process., vol. 22, no. 8, pp. 3158–3166, Aug. 2013.

[12] V. Bonato, E. Marques, and G. A. Constantinides, “A parallel hardware architecture for scale and rotation invariant feature detection,” IEEE Trans. Circuits Syst. Video Technol., vol. 18, no. 12, pp. 1703–1712, Dec. 2008.

[13] M. Calonder, V. Lepetit, M. Oezuysal, T. Trzcinski, C. Strecha, and P. Fua, “BRIEF: Computing a local binary descriptor very fast,” IEEE Trans. Pattern Anal. Mach. Intell., vol. 34, no. 7, pp. 1281–1298, Jul. 2012.

[14] L. Chang and J. Hernández-Palancar, “A hardware architecture for SIFT candidate keypoints detection,” in Progress in Pattern Recognition, Image Analysis, Computer Vision, and Applications. New York, NY, USA: Springer-Verlag, 2009, pp. 95–102.

[15] K. Mikolajczyk and C. Schmid, “An affine invariant interest point detector,” in Proc. 7th ECCV, May 2002, pp. 128–142.

[16] Y. Lin, C.Yeh, S. Yen, C. Ma, P. Chen, and C.-C.J. Kuo, “Efficient VLSI design for SIFT feature description,” Next-Generation Electronics (ISNE), 2010 International Symposium on , pp.48-51, Nov. 2010.