

FPGA-based ORB Feature Extraction for Real-Time Visual SLAM

Weikang Fang^{*†}, Yanjun Zhang^{*}, Bo Yu[†], Shaoshan Liu[†],

^{*} Institute of Application Specific Instruction Set Processor,

School of Electronics and Information, Beijing Institute of Technology, Beijing, 100081, China

[†] PerceptIn, Shenzhen, 518046, China

Abstract—Simultaneous Localization And Mapping (SLAM) is the problem of constructing or updating a map of an unknown environment while simultaneously keeping track of an agent’s location within it. How to enable SLAM robustly and durably on mobile, or even IoT grade devices, is the main challenge faced by the industry today. The main problems we need to address are: 1.) how to accelerate the SLAM pipeline to meet real-time requirements; and 2.) how to reduce SLAM energy consumption to extend battery life. After delving into the problem, we found out that feature extraction is indeed the bottleneck of performance and energy consumption. Hence, in this paper, we design, implement, and evaluate a hardware ORB feature extractor and prove that our design is a great balance between performance and energy consumption compared with ARM Krait and Intel Core i5.

Index Terms—ORB, feature extraction, SLAM, FPGA

I. INTRODUCTION

Simultaneous Localization And Mapping (SLAM) [7]-[9] is the core enabling technology behind applications such as autonomous vehicles, robotics, virtual reality (VR), and augmented reality (AR). In detail, SLAM is the problem of constructing or updating a map of an unknown environment while simultaneously keeping track of an agent’s location within it. Fig. 1 shows a simplified version of our production-level Visual Inertial SLAM system which utilizes design concepts from MSCKF [1] and ORB-SLAM [2].

We implemented this SLAM technology on a quad-core ARM v8 mobile SoC. Based on our profiling results, the feature extraction stage is indeed the most computation-intensive, consuming >50% of the CPU resources. Even with such heavy computation resource consumption, it still takes about 50 ms to extract features from a VGA-resolution (640×480) image, leading to a frame rate only at about 20 FPS.

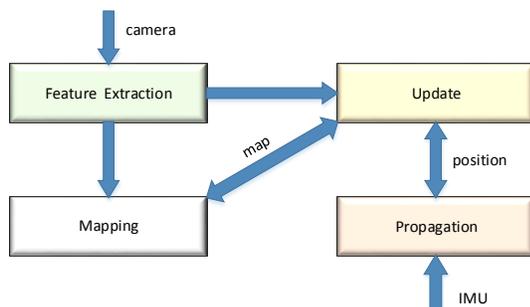


Fig. 1: Overview of a visual inertial SLAM system.

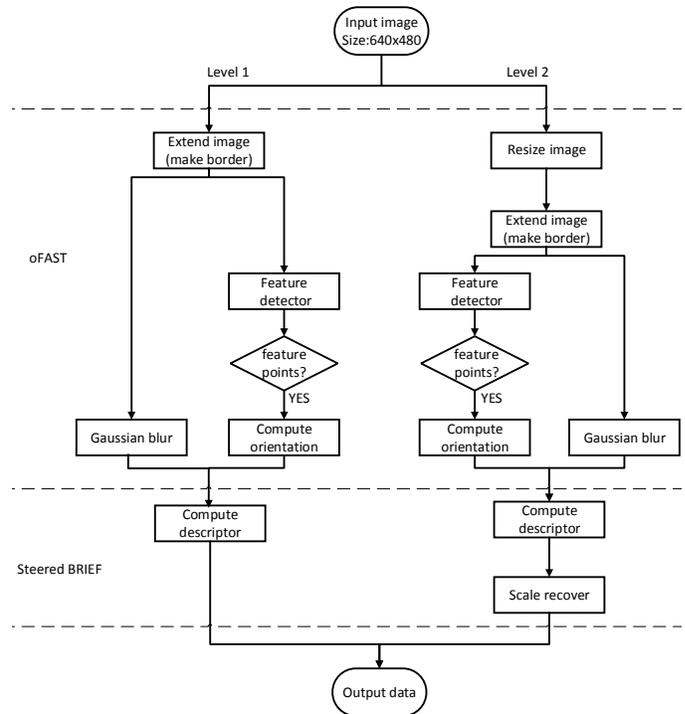


Fig. 2: ORB based feature extraction algorithm.

Therefore, in this paper, we aim to address these practical problems down to the hardware level, by implementing a hardware accelerator of ORB feature extractor. ORB is widely used in robotics and it is proven to provide a fast and efficient alternative to SIFT [3]. In addition, we examine its performance, resource occupation, and energy consumption of the design in order to achieve further improvements.

The rest of this paper is organized as follows. In Section II, we review the background information of ORB algorithm. In Section III we describe the architecture of the proposed hardware feature extractor. In Section IV, we share the experimental methodologies and results. Finally, we summarize our conclusions in Section V.

II. ORB FEATURE EXTRACTION ALGORITHM FOR VISUAL SLAM

The overview of ORB based feature extraction algorithm is shown in Fig. 2. It consists of two parts, oFAST (Oriented Feature from Accelerated Segment Test) [5] based feature

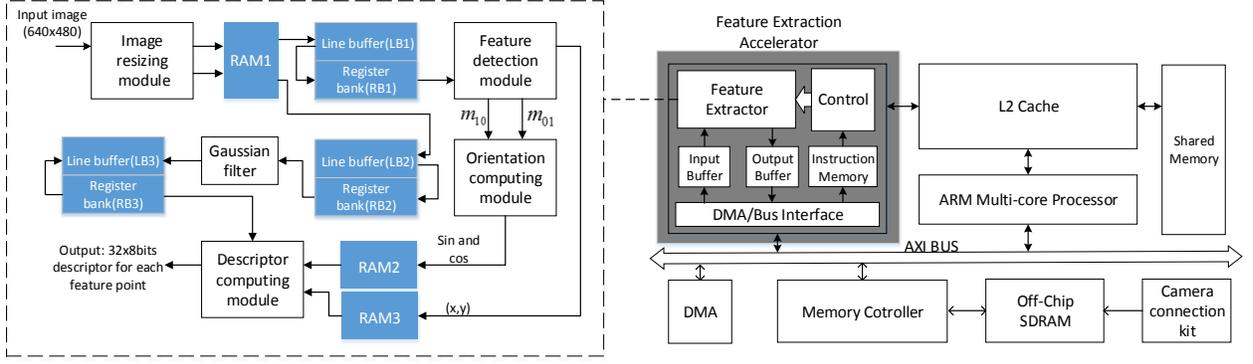


Fig. 3: Hardware architecture of real-time SLAM system (right) and hardware architecture of ORB feature extraction accelerator (left).

detection and BRIEF (Binary Robust Independent Elementary Features) based feature descriptors computation [6].

A. Oriented Feature from Accelerated Segment Test (oFAST)

In general, given an image, oFAST helps find out feature points and then calculates orientations of feature points to ensure rotation invariant. Details of oFAST are shown in Fig. 2. First, resize the original image using bilinear interpolation [4]. Second, compute feature points of both original image and resized image. Third, determine the orientation of feature points. The orientation of feature points are computed as follows.

Define the patch of a feature point be the circle centered at the feature point. The moments of the patch, m_{pq} , are defined as,

$$m_{pq} = \sum_{x,y \in r} x^p y^q I(x,y) \quad p, q = 0 \text{ or } 1 \quad (1)$$

where $I(x,y)$ is the intensity value of the point (x,y) in the patch and r is the radius of the circle. The orientation of the feature point, θ , is obtained by, $\theta = \arctan(\frac{m_{01}}{m_{10}})$. $\sin \theta$ and $\cos \theta$ is calculated as follows.

$$\sin \theta = \frac{m_{01}}{\sqrt{m_{10}^2 + m_{01}^2}} \quad \cos \theta = \frac{m_{10}}{\sqrt{m_{10}^2 + m_{01}^2}} \quad (2)$$

B. Steered Binary Robust Independent Elementary Features (steered BRIEF)

In general, a feature point is represented by a set of descriptor. In ORB algorithm, steered BRIEF algorithm is employed to compute descriptors of feature points. The details of steered BRIEF algorithm is described as follows.

Firstly, consider the circular patch defined in oFAST. Select M pairs of points in the patch according to Gaussian distribution. Secondly, in order to ensure rotation invariant, rotate these pairs of points by the angle determined by equation 2. Thus, after rotation, M pairs of points could be labeled as $D_1(I_{A_1}, I_{B_1}), D_2(I_{A_2}, I_{B_2}), D_3(I_{A_3}, I_{B_3}) \dots D_M(I_{A_M}, I_{B_M})$, where A_i and B_i are the two points of a pair, I_{A_i} and I_{B_i} are intensity values of the point. Thirdly, an operator is defined as follows,

$$T(D_i(I_{A_i}, I_{B_i})) = \begin{cases} 1 & \text{if } I_{A_i} \geq I_{B_i} \\ 0 & \text{if } I_{A_i} < I_{B_i} \end{cases}, \quad (3)$$

For each pair of points, the operator, T , produces one bit of result. With M pairs of points, the operator, T , produces a bit vector with the length of M . For example, T produces the following results, $T(D_1(I_{A_1}, I_{B_1})) = 1$, $T(D_2(I_{A_2}, I_{B_2})) = 1$, $T(D_3(I_{A_3}, I_{B_3})) = 0 \dots T(D_M(I_{A_M}, I_{B_M})) = 1$, then the descriptor of the feature point P is 110...1. The bit vector is descriptor of the feature points.

III. HARDWARE ARCHITECTURE

A. Overall Architecture of FPGA-based Real-Time SLAM

Fig. 3 (right) illustrates the proposed FPGA-based architecture of real-time SLAM system. ARM Multi-core processors are used to perform control logic and other computations in SLAM. AXI bus is employed to connected memory systems, ORB accelerator and ARM processors.

The feature extraction accelerator consists of a data accessing part and a kernel part. The kernel part is composed of Instruction Memory, Control unit and Feature Extractor. The data accessing part is composed of Input Buffer, Output Buffer and DMA/Bus Interface. The DMA/Bus Interface directly accesses data stream captured by the camera via AXI Bus and store data in the Input Buffer for the Feature Extractor. The results of the Feature Extractor are stored in the Output Buffer and are finally sent to L2 cache. The ARM multi-core System would use these features for further processing.

B. Hardware Architecture of ORB Feature Extractor

Hardware architecture of ORB feature extractor is depicted in Fig. 3 (left). The architecture consists of the image resizing module, the feature detection module, the orientation computing module, and the descriptor computing module. Line buffers and registers are used to store intermediate results.

The resizing module implements bilinear interpolation and builds a 2-level image pyramid. The size of an input image is 640×480 and the size of a scaled image is 533×400 . RAM1 stores the image of each level.

The LB1 (line buffer 1) stores 31 lines of pixels. RB1 (register bank 1) stores 31×31 pixels of neighborhood that contains a circular patch for each point. The feature detection module detects the feature point and computes the moments of a patch, i.e. m_{10} and m_{01} . The coordinates (x,y) of the feature points are stored in RAM3.

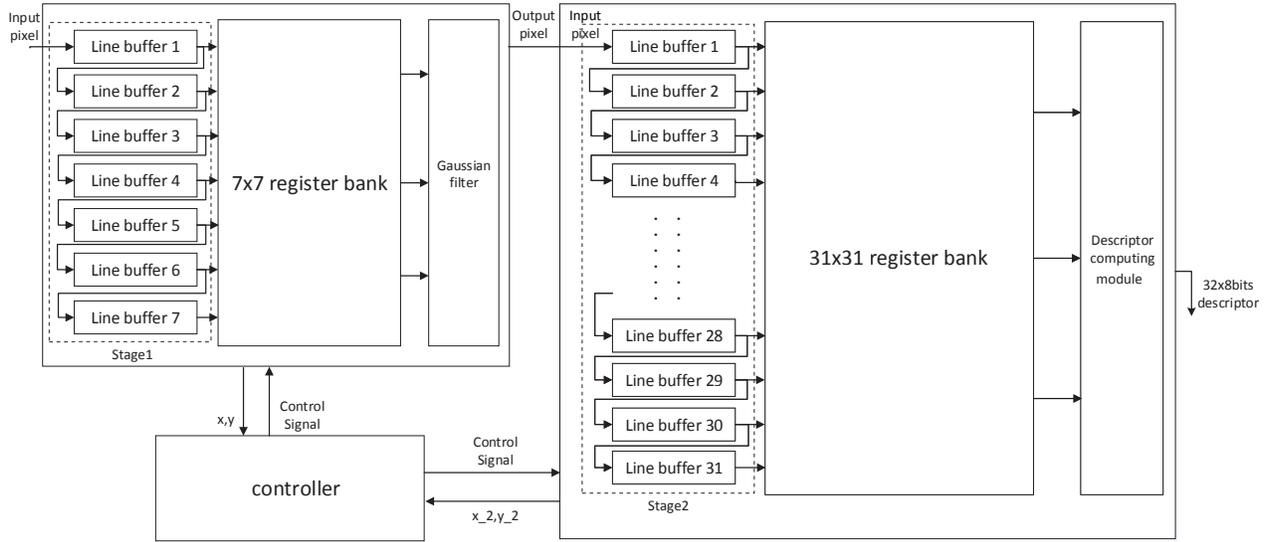


Fig. 4: Hardware architecture of the synchronized two-stages line shifting buffers

The orientation computing module computes $\sin \theta$ and $\cos \theta$ according to equation 2. $\sin \theta$ and $\cos \theta$ are stored in RAM2. A 7×7 hardware Gaussian filter is used to smooth images. LB2 and RB2 buffers 7 lines of pixels and a 7×7 patch of pixels for the Gaussian filter respectively. LB3 and RB3 are used to store results of Gaussian filter. Since the Descriptor computing module calculates descriptors in a 31×31 patch of pixels, LB3 stores 31 lines of pixels and RB3 buffers 31×31 bytes.

The descriptor computing module takes smoothed images, orientations and coordinates of feature points in the image as inputs and calculates descriptors of feature points. Since the descriptor computing module depends on the results of the Gaussian filter, a straightforward implementation is to buffer the whole image before starting to compute descriptors. However, this straightforward design that needs to store the whole image requires substantial on-chip memory resources. Furthermore, starting to compute descriptors after Gaussian filtering on the whole image will stall the stream processing of the feature detection and orientation computation, which will increase the latency of the system.

The synchronized two-stages shifting line buffers, as shown in Fig. 4, is proposed to compute Gaussian filtering and descriptors in a streaming way. The first stage and the second stage of line buffers store pixels for the Gaussian filter and the descriptor computing module respectively. Line buffers in each stage are organized as a shifter. The control unit synchronizes the data movement in the line buffers. If no feature point is detected, the two-stages line buffers shifts data in the buffers. If a feature point is detected, the control unit stops shifting the two-stages line buffers and starts to compute the descriptor for the feature point. By utilizing the proposed synchronized two-stages line buffers, 575K bytes of on-chip memory are saved.

IV. EVALUATION RESULTS AND DISCUSSION

A. Word length optimization

According to equation 2, the orientation of a feature point is determined by the moments of a circular patch. In our design, the radius of a patch is 15. The range of the value of m_{10} and m_{01} is from -624750 to 624750 . 21 bits are needed to represent m_{10} and m_{01} . The orientation computing module requires even more bits to keep precision, which will consume substantial hardware resources.

Word length optimization is used to reduce hardware consumption. By computing m_{10} and m_{01} of all possible circular patches, we find that many significant bits in m_{10} and m_{01} are zeros in most cases. Besides, the influence of low bits in the calculation is quite small. Therefore, the proposed operation to shorten the word length is described as follow.

First, starting from the highest data bit without considering sign bit, find the overlapping 0s of m_{10} and m_{01} then remove them. Second, in the remaining data bits, take the higher N bits. If remaining part is less than N bits, fill it with zeros. Finally, splice the sign bit and obtained N bits together. In this way, the word length would be shorten to N+1 bits.

Truncating the word length introduces truncation errors in computing rotation angles and further affects the coordinates of points after rotating. For a particular point (x, y) , the coordinates after rotating is (x', y') that is determined by,

$$\begin{aligned} x' &= x \cdot \cos \theta + y \cdot \sin \theta, \\ y' &= y \cdot \cos \theta - x \cdot \sin \theta. \end{aligned} \quad (4)$$

We define the following metric to quantitatively evaluate the error introduced by word length truncation,

$$Error = \sqrt{(x' - x'_N)^2 + (y' - y'_N)^2} \quad (5)$$

where (x', y') is the coordinate calculated with original word length while (x'_N, y'_N) is calculated with the word length of N.

Pixels in a patch have various sensitivity to word length truncations. When rotating two different points in a patch by an identical angle, the point that is further from the original point will move further. Hence, the points that is further from the original point will be more likely to be in a wrong location. The maximum errors would occur at points $(\pm 15, \pm 15)$ which are the farthest points from the original point in a patch.

Fig. 5 illustrates the relationship between the word length and the maximum and mean errors. The results shows that the maximum error increases exponentially as the word length decreases. Considering both the precision and hardware consumption, the word length of the orientation computing unit is 8 bits in our design.

Table I compares the hardware consumption of the orientation computing module with 21 bits and 8 bits word length. The results show that the numbers of registers and LUTs are significantly reduced by 65% and 83% after word length optimization.

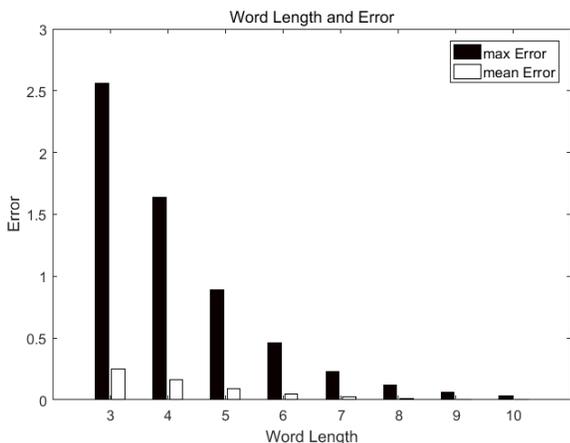


Fig. 5: The relationship between word length and the maximum Error and mean Error.

B. Hardware evaluation

The proposed ORB feature extractor is implemented and evaluated on an Altera Stratix V FPGA. The proposed hardware consumes 25648 ALUTs, 21791 registers, 1.18M bytes of BRAMs and 8 DSP Blocks. The clock frequency of the proposed hardware is 203MHz. The latency of the hardware for processing an image is 14.8ms, and the throughput of the hardware is 67 frames per second.

We compared the proposed hardware with multi-core ARM processors and Intel CPU. Table II shows the performance comparison between the proposed hardware and ORB implementations on ARM Krait and Intel Core i5 CPU. Compared with ARM Krait, the latency and energy consumption is

TABLE I: Hardware consumption of the Orientation computing unit

	Word length = 21 bits	Word length = 8 bits	Reduction 8 bits/21 bits
Number of ALUT	2416	848	65%
Number of Register	3114	576	82%
Number of DSP	3	3	0%

TABLE II: Hardware performance comparison

	Clock Freq. (GHz)	Latency (ms)	Throughput (FPS)	Energy (mJ/frame)
Proposed design	0.203	14.8	67	68
ARM Krait	2.26	30	33	75
Intel Core i5	2.9	25	40	400
Improvement vs ARM	–	51%	103%	9%
Improvement vs Intel	–	41%	68%	83%

reduced by 51% and 9%, and throughput is improved by 103%. Compared with Intel i5 CPU, the latency and energy consumption is reduced by 41% and 83%, and throughput is improved by 68%.

V. CONCLUSION

Feature extraction is the most computation-intensive part in a visual inertial SLAM system. In our production-level system, we utilize ORB as our feature extractor as ORB is widely used in robotics and it is proven to provide a fast and efficient alternative to SIFT. Based on our profiling results, ORB feature extraction takes over 50% of the CPU resources as well as energy budget. In this paper, we aimed to solve this problem and we have designed, implemented, and evaluated a hardware ORB feature extractor. Our design runs only at about 203 MHz to reduce energy consumption. In the computation latency category, it outperforms ARM Krait by 51% and Intel Core i5 by 41%; in the computation throughput category, it outperforms ARM Krait by 103%, and Intel Core i5 by 68%; and most importantly, in the energy consumption category, it outperforms ARM Krait by 10% and Intel Core i5 by 83%. Thus this design is proven to be a great balance between performance and energy consumption.

REFERENCES

- [1] A. Mourikis and S. Roumeliotis, "A multi-state constraint Kalman filter for robot-aided inertial navigation", IEEE international conference on Robotics and automation, 2007.
- [2] R. Mur-Artal, J. Montiel and J. Tardos, "ORB-SLAM: a versatile and accurate monocular SLAM system", IEEE Transactions on Robotics, 31(5), pp.1147-1163, 2015.
- [3] E. Rublee, V. Rabaud, K. Konolige and G. Bradski, "ORB: An efficient alternative to SIFT or SURF", IEEE international conference on Computer Vision (ICCV), 2011.
- [4] S. Se, D. Lowe and J. Little, "Mobile Robot Localization and Mapping with Uncertainty using Scale-Invariant Visual Landmarks", International Journal of Robotics Research 21.8(2002):735-760.
- [5] Y. Biadgie and K. Sohn, "Feature Detector Using Adaptive Accelerated Segment Test", International Conference on Information Science and Applications (ICISA), 2014.
- [6] M. Calonder, V. Lepetit, C. Strecha and P. Fua, "BRIEF: Binary Robust Independent Elementary Features", European Conference on Computer Vision Springer-Verlag, 778-792, 2010.
- [7] A. Davison, I. Reid, N. Molton and O. Stasse, "MonoSLAM: Real-Time Single Camera SLAM", IEEE Transactions on Pattern Analysis and Machine Intelligence, Volume, 29, Issue, 6, June 2007.
- [8] T. Bailey and H. Durrant-Whyte, "Simultaneous localization and mapping (SLAM): part I", IEEE Robotics and Automation Magazine, Volume, 13, Issue: 2, 2006.
- [9] M. Dissanayake, P. Newman, S. Clark, H. Durrant-Whyte and M. Csorba, "A solution to the simultaneous localization and map building (SLAM) problem", IEEE Transactions on Robotics and Automation, Volume, 17, Issue, 3, Jun. 2001.