# HcveAcc: A High-Performance and Energy-Efficient Accelerator for Tracking Task in VSLAM System

Renwei Li[1,2], Junning Wu[2], Meng Liu[1,2,3] , Zuding Chen[1,2], Shengang Zhou[1,2], Shanggong Feng[2]
[1]Intelligent Robot-Processor Laboratory, Research Center for Brain-inspired Intelligence
Institute of Automation, Chinese Academy of Sciences, Beijing, China
[2]Beijing Haawking Technology Co., Ltd.
[3]Email: liumeng2013@ia.ac.cn

***Abstract*—Visual SLAM (vSLAM) is a critical computer vision technology that is able to build a map of an unknown environment and perform location, simultaneously leveraging the partially built map. While existing several software SLAM processing frameworks, underlying general-purpose processors still hardly achieve the real-time SLAM at a reasonably low cost. In this paper, we propose *HcveAcc*, the first specialized CMOS-based hardware accelerator to help optimize the tracking task in the vSLAM system with high-performance and energy-efficient. Our HcveAcc targets to solve the time overhead bottleneck in the tracking process—high-density feature extraction and high-precision descriptor generation, and provides a configurable hardware architecture that handles higher resolution image data. We have implemented the HcveAcc in a 28nm CMOS technology using commercial EDA tools and evaluated it for the EuRoC and TUM dataset to demonstrate the robustness and accuracy in the SLAM tracking procedure. Our results show that HcveAcc achieves 4.3X speedup while consuming much less energy compared with state-of-the-art FPGA solutions.**

## I. Introduction

Simultaneous Localization and Mapping (SLAM) has been a hot research topic in the last two decades in navigation, robotic mapping and odometry for virtual reality or augmented reality [1]. To realize the real-time vSLAM, the long-term way forward is to bring sensors, algorithms, and processors together, thus, ASIC is a promising direction. ORB-SLAM is the most widely adopted feature-based method which aims to solve the real-time operation problem in vSLAM fields [1], [2]. Although ORB-SLAM is a complete and reliable solution to monocular, stereo and RGB-D cameras on CPU type of processing units, there is still room to enhance its performance on some specialized designs. The proposals for exploiting ORB-SLAM acceleration on FPGA platforms can be found in recent pieces of literature [3], [4]. To make ORB descriptor realization hardware-friendly, eSLAM uses the low precision angular quantization [3]. In the eSLAM work, hardware implementation details of feature extraction are not discussed, although the integration of the NMS (non-maximum suppression) computing process and how to efficiently utilize bandwidth resources while reducing large-scale pixel data copying interactions are still design difficulties. As stated in this work, eSLAM could achieve up to 3X speedup in frame rate and 71X improvement in energy efficiency when compared with the Intel i7 CPU. However, evaluations are only done on the easy-level TUM dataset, which means guaranteeing tracking accuracy is still hard based on the hardware accelerator. Fang et al. also explored an ORB extractor realization on FPGA [4]. They proposed a hardware architecture of ORB feature extraction based on the number of line buffers regardless of NMS (non-maximum suppression) function which is important for ORB-SLAM framework but did not provide the detailed architecture for calculating the centroid, rotating the sampling patterns, and generating the key points. In addition, SLAM experiments lack in this work [4] at which the extractor aims.

Very few of previous feature extraction related hardware works [5], [6] target at non-streaming architectures which can be directly integrated into SoC design. By using the streaming architecture [5], Lam et al. illustrated a FAST-BRIEF architecture that was integrated with pixel line buffers, FAST corner detectors, and BRIEF descriptors without rotational consistency to save runtime and area cost. The rBRIEF-based feature extraction method was then proposed to improve the feature matching quality [6], however, streaming architecture cannot be used to solve the SLAM problem with dynamic pixel flow from sensors. Compared with these previous works, we provide the HcveAcc accelerator design that is compatible with ORB-SLAM system requirements and can be integrated into any SoC architecture. We architect, design and evaluate a domain-specific accelerator for vSLAM workloads. The key contributions are summarized as follows.

- We propose HcveAcc, an accelerator that can help perform high-density feature extraction and high-precision descriptor generation for real-time operations within 200$mW$ power consumption.
- HcveAcc is integrated with our proposed PE (Process Element) blocks parallelly to balance storage computation overhead with the pipeline operation.
- HcveAcc is the first specialized CMOS-based hardware architecture targeted to solve the ORB-SLAM problems, and we have evaluated it for the EuRoC and TUM dataset to demonstrate the robustness and accuracy in the SLAM tracking procedure.

This paper is organized as follows. Section II presents an overview of our framework. Section III describes the proposed architecture. Section IV demonstrates the experimental results. Section V concludes the paper.

978-3-9819263-4-7/DATE20/©2020 EDAA

## II. Overview of Proposed Framework

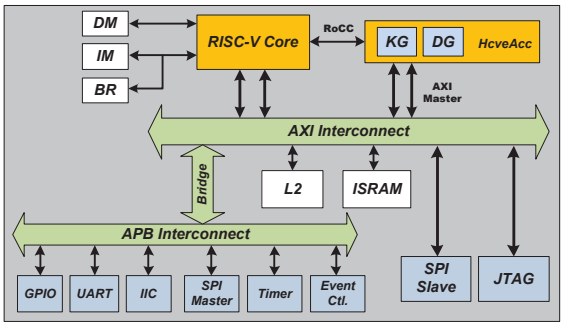


Fig. 1: The overall architecture of our SoC. The white areas represent DM (data memory), IM (instruction memory), BR (boot ROM), L2 cache, and ISRAM (image SRAM), respectively. The image data is stored in ISRAM. The green areas represent the bus resources, which are AXI interface, APB interface and Bridge. The orange areas include the HcveAcc and one RISC-V core which is connected by using RoCC interface. The blue area indicates the integrated peripheral modules, including GPIO, JTAG, etc.

The overall SoC architecture with our proposed HcveAcc is illustrated in Fig. 1. HcveAcc is designed as the high-performance computer vision engine accelerator which includes the key-point (feature) generation (KG) module and the descriptor generation (DG) module. The RISC-V core is not only responsible for task scheduling assignment, but also for sharing the feature filtering tasks of HcveAcc.

## III. Proposed Architecture

### A. KG Module

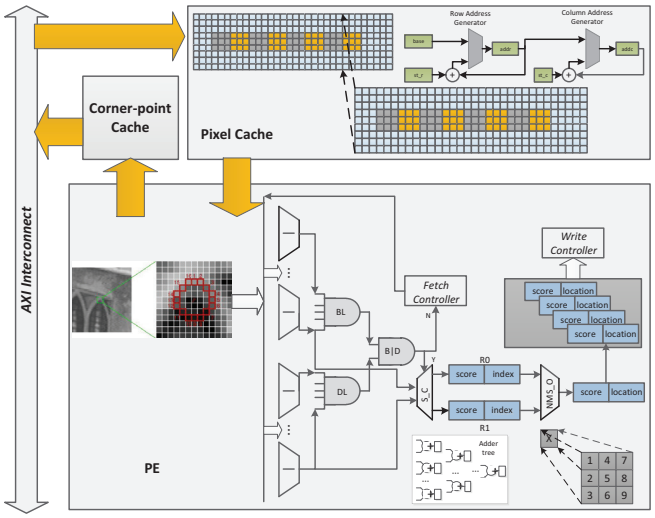


Fig. 2: The overall architecture of our KG module. The pixel cache is responsible for receiving pixel data requested from AXI bus. The PE is composed of the corner detection module and NMS (Non-maximal Suppression) module. The corner detection module includes the FAST12-corner detector and NMS filter.

The architecture of KG module is illustrated in Fig. 2. It consists mainly of pixel cache, PE (Process Element) and corner cache. The basic principle of the FAST detector [7] is: if there is a certain attribute difference between a pixel and enough neighboring pixels in the surrounding area, and the difference of metric is greater than the specified threshold, it can be concluded that the pixel and its neighboring pixels can be identified. Thus, the difference is caused by the center pixel which can be regarded as a key-point (corner point). The evaluation value of each pixel is parsed from grayscale images, so the attribute examined by the FAST detector is the gray level difference between the pixel and its neighborhood. As shown in the PE of Fig. 2, for all the pixels on this image patch, the difference is between the total of 16 pixels with red color and the center point, FAST12-corner Detector relies on the circumference of the 7*7 neighborhood pixels and the radius of 3 is considered. If there is a continuous 12 pixel and the absolute difference of the gray level difference of the center point is greater than a given threshold, the pixel point is detected as a corner point.

$$S_{p\to x}=\begin{cases} d, I_{p\to x} \le I_p - t\,(darker) \\ s, I_p - t \le I_{p\to x} \le I_p + t\,(similar) \\ b, I_p + t \le I_{p\to x}\,(brighter) \end{cases} \tag{1}$$

For each location on the circle $x \in \{1..16\}$, the pixel at that position relative to $p$ (denoted by $p \to x$) can have one attribute possibility which is dark, similar or bright. Then, choose an $x$ and compute $S_{p\to x}$ for all $p \in P$ (the set of all pixels in all training images) partitions $P$ into three subsets, $P_d$, $P_s$, $P_b$, where each $p$ is assigned to $P_{S_{p\to x}}$.

*1) Non-maximum Suppression:* The Non-maximum Suppression (NMS) is to make sure that in object detection, a particular object is identified only once. Consequently, NMS has a large positive impact on filtering corner points in each local NMS window. The design flow is described below. Firstly, a score function of $V$ is computed for each of the detected points. The $V$ is defined as the sum of the absolute difference between pixels in the contiguous arc and the center pixel. We calculate scores by using the adder tree structure. Secondly, two adjacent interest points are considered to compare their $V$ values. Thirdly, we need to discard the one with the lower $V$ value. The entire process can be summarized mathematically as Equation (2), where $p$ represents the center pixel, $t$ is the threshold for detection and pixel values correspond to the $N$ contiguous pixels in the circle. The $S_{bright}$ and $S_{dark}$ are both corner point sets which have been defined in Equation (1). The NMS is performed in a 3*3 mask which is marked in Fig. 2, the number represents our computing order in this mask window. After comparing among these nine potential corner points, the key-point would be chosen to represent the mask window. Compared with eSLAM work [3], we only need to design the suitable size of corner-point cache to store the location information without saving all corner scores.

$$V = \max\left(\begin{array}{c} \sum\limits_{x\in S_{bright}} |I_{p\to x} - I_p| - t, \\ \sum\limits_{x\in S_{dark}} |I_p - I_{p\to x}| - t \end{array}\right) \tag{2}$$

*2) PE and Pixel Cache Design:* We propose the Process Element (PE) design which is marked in Fig. 2. The corner detector and NMS modules are integrated into the PE unit. Assuming the pixel data is ready for computing, we send the input data into 16 sets of adder and subtractor combinations separately to perform the useful difference in the corner determination process, as illustrated in Equation (1). The BL (Bright Logic) and DL (Dark Logic) represent the bright corner decision logic and the dark corner decision logic, respectively. Since BL or DL will not be logically 1 at the same time, we only need to perform XOR operation on BL and DL. The above logic components are marked in gray in Fig. 2. If the above XOR operation value is logic 1, then the NMS related logic is enabled, otherwise, PE jumps directly to the data load controller. S_C (score calculation) components mean the score for the NMS process which can be calculated by the sum of intermediate results from the sets of adder and subtractor before. The NMS for ORB-SLAM is designed with 3*3 mask window, so we have customized coordinate position jump logic for this purpose. At most, we can get one optimal corner point for each group of NMS mask window. In Fig. 2, R0 means the register for storing temporary results of sore and its index. R1 means the current results of score and index. We only need to compare these two scores and determine if R0 can be replaced by the values of R1. After nine times of calculation, we can write this corner point information to memory. By combining corner detection and NMS process, a large number of temporary results of scores for each corner point are saved according to the traditional line buffer strategy.

To reduce the data interaction overhead through the AXI bus, the pixel cache module is designed which incorporates the size of 7*7 for FAST corner detection window and the size of 3*3 for NMS calculation window. By caching pixel data as much as possible and balancing the computation time with caching cost, we design the pixel cache module as shown in Fig. 2. It is combined with the data size ( 9*32, 9 rows and 32 columns ) through the AXI bus, and different colors mean each NMS mask window. For each NMS mask window, PE module is executed 9 times. In Fig. 2, 9 rows can help guarantee the complete pixel data to fill each FAST corner detection window. The size-related parameters of the Window Fusion module can be easily modified based on the AXI bus efficiency since we choose 32 here for the burst transaction to ensure aligned access to memory data. We also have designed the address jump logic in the horizontal and vertical directions which are marked with green color in Fig. 2. The pixel cache is designed as ping-pong mode. Additionally, pixel cache architecture can help avoid large amount pixel data caching, and data splicing between fetch operations as shown in Fig. 2, it illustrates a direct overlapping diagram of four pixel cache modules. To ensure address alignment and full coverage of PE execution for each pixel, the pixel cache architecture is designed with an address step of 24 instead of 32. Thus, for the adjacent calculation round between pixel cache modules, the repeated PE calculation can be avoided. The dashed lines in the figure identify the adjacent data locations, which guides the beginning of the next round of calculations. This is regarded as the horizontal direction. For vertical data splicing, we only need to span 3 rows of pixel data in the next fetch process. For instance, if the resolution of images is 752*480, 31 times of pixel cache movements are required for the horizontal direction, and 160 times of pixel cache movements are required for the vertical direction. In this way, any pixel is executed by the PE module except for some boundary regions of the current image which is at most 5 rows or columns on the boundary.

## B. DG Module

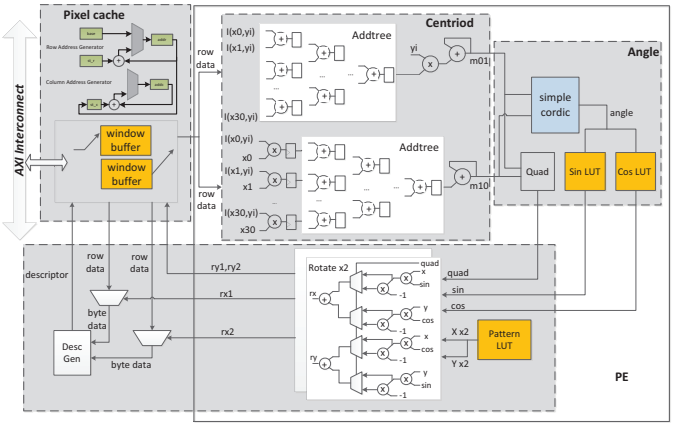


Fig. 3: Overall architecture of our DG module.

After detecting key-point we need to compute a descriptors for each one. Our DG module is responsible for generating descriptor according to the rBRIEF (rotation-aware BRIEF) algorithm [7] which overcomes the shortcomings of the BRIEF algorithm which is less adaptable to rotation and scale transformation. To make hardware friendly, we use a high-throughput pixel cache to interact with the bus and computational modules to create the image patches needed for descriptor calculations. The computing unit follows a pipeline design to reduce hardware overhead. By using the CORDIC module, we can quantify the calculation accuracy to one degree. The overall architecture of our DG module is shown in Fig. 3 which includes the pixel cache and PE module.

*1) rBRIEF Calculation:* The BRIEF descriptor is a bit string of an image patch from a set of binary intensity tests, as shown in Equation (3). The $p$ represents the feature point of the current image patch. The $a_i$ and $b_i$ are the pixel pair picked from the current image patch. The $p(a_i)$ and $p(b_i)$ separately means the value of pixel intensity. Then, we can easily compute the binary result for each pixel pair to consist of a vector of $n$ binary tests which is commonly recommended to 256 bits, as illustrated in Equation (4).

$$\varphi(p; a_i, b_i) = \begin{cases} 1, p(a_i) < p(b_i) \\ 0, p(a_i) \geq p(b_i) \end{cases} \tag{3}$$

$$f_n(p) = \sum_{i=1}^{n} 2^i \times \varphi(p; a_i, b_i) \tag{4}$$

Based on the BRIEF algorithm, the rBRIEF algorithm adds the centroid definition of the image patch, the representation process of the angle, and the rotation calculation of the pattern pairs in the image patch. The intensity centroid assumes that a corner's intensity is offset from its center, and this vector may be used to impute an orientation. The ORB feature defines the moments of a patch in two directions, as shown in Equation (5). We have used adder tree to sum these values which are marked in the Centroid window of Fig. 3. Using the patch orientation, we can construct the rotating locations for each pair in the current image patch which is described in Equation (6), then the new locations are transformed from $(x, y)$ to $(rx, ry)$.

$$\begin{aligned} m_{01} &= \sum_{y} y \times I(x,y) = \sum_{i=0}^{30}\sum_{j=0}^{30} y_i \times I(x_j, y_i) \\ m_{10} &= \sum_{x} x \times I(x,y) = \sum_{i=0}^{30}\sum_{j=0}^{30} x_j \times I(x_j, y_i) \end{aligned} \tag{5}$$

$$\begin{cases} rx = xcos\theta + ysin\theta \\ ry = -xsin\theta + ycos\theta \end{cases} \tag{6}$$

$$\begin{cases} x_{i+1} = x_i cos\theta + y_i sin\theta \\ y_{i+1} = y_i cos\theta - x_i sin\theta \\ \theta_{i+1} = \theta_i + \theta \end{cases} \tag{7}$$

We have applied the CORDIC block to perform a coordinate rotation using the coordinate rotation digital computer algorithm [8]. It calculates the trigonometric functions of sine, cosine, magnitude, and phase (arctangent) to any desired precision. A CORDIC algorithm is useful to avoid using the hardware multiplier because the only operations it requires are addition, subtraction, bit shift, and lookup. The angle calculation uses a simplified CORDIC calculation module to achieve one-degree angular quantization, which ensures accuracy and efficiency. The calculation of the quadrant division reduces the storage space overhead of the angle-dependent sin/cos LUT. With multiple rotations, the angle value can gradually approach the final result. Assume that the current angle value is $\theta$, the next value is acquired as in Equation (7). Besides, the Equation (7) can be simplified to Equation (8). Considering the complexity of the hardware implementation, if the angle of each rotation is set to $\theta = \tan^{-1}\left(2^{-i}\right)$, the above equation can then be simplified to Equation (9). For instance, assuming that the initial vector is the first quadrant arbitrary vector, the vector is rotated in the x-axis direction until it approaches the x-axis (y=0), and the accumulated angle value $\theta$ in the process is the final angular output value. Then we can form the angle calculation module by using the LUT resource and regular combinatorial logic which are marked with **Angle** in Fig. 3.

$$\begin{cases} \widetilde{x_{i+1}} = x_i + y_i tan\theta \\ \widetilde{y_{i+1}} = y_i - x_i tan\theta \\ \theta_{i+1} = \theta_i + \theta \end{cases} \tag{8}$$

$$\begin{cases} \widetilde{x_{i+1}} = x_i + y_i \times 2^{-i} \\ \widetilde{y_{i+1}} = y_i - x_i \times 2^{-i} \\ \theta_{i+1} = \theta_i + tan^{-1}\left(2^{-i}\right) \end{cases} \tag{9}$$

*2) PE and Pixel Cache Design:* The PE in the DG module is marked in Fig. 3, including the angle calculation and descriptor formulation process. The pixel cache design in DG is like the KG pixel cache which is described in the above sections. By forming PEs parallelly, we can additionally accelerate the DG module with increasing the cost of pixel cache which is a trade-off strategy.

## IV. Experimental Results

### A. Hardware Implementation

TABLE I: Result comparisons between FAST-BRIEF and HcveAcc.

| ***Design Name*** | | FAST-BRIEF [5] | | HcveAcc | |
|---|---|---|---|---|---|
| ***CMOS Tech.*** | | 65nm | | 28nm | |
| ***Components*** | | Detector | Descriptor | Detector | Descriptor |
| ***Quantization*** | | N/A | *256bits (N)* | FAST12 | *256bits (Y)* |
| ***Number of PEs*** | | N/A | | 2 | 2 |
| ***Area cost (sq um)*** | ***Comb*** | 10146.4 | 11923.9 | 2837.6 | 13345.6 |
| | ***Seq*** | 9066.3 | 42154.9 | 968.1 | 7175.4 |
| | ***Total*** | 73291.5 | | 24326.7 | |
| ***Freq.(GHz)*** | | N/A | | 1.0 | |
| ***Power(mW)*** | | 1131 | | 50 | |

*1) Synthesized Results:* The proposed HcveAcc is developed in the Verilog language. The baseline for comparison is the only existing synthesized work [5] which is called FAST-BRIEF using 65nm technology. Detailed results are shown in Table I. We have synthesized our proposed modules with TSMC 28nm technology. The timing constraint is based on worst corner conditions with the frequency of 1GHz, while this synthesized condition is not mentioned in the FAST-BRIEF work. The area results in this table represent the data of the combinational logic and the sequential logic, respectively, and the total area of HcveAcc can be saved up to 66.8% compared with the cost of FAST-BRIEF. We can also find that the power consumption of HcveAcc is much less than FAST-BRIEF. For quantization, we have adopted the descriptor generation with angle calculation which is marked with *Y*, however, FAST-BRIEF can not solve image rotation problem with *N* marked. Besides, the streaming-architecture without storing pixel data [5] cannot avoid dropping feature points which are not suitable for high-precision requirements of ORB-SLAM system. The Fig. 5 (a) and Fig. 5 (b) separately represents the layout of our proposed KG module and DG module.

*2) Performance and Power Evaluation:* Since ORB-SLAM only needs a certain number of high-quality feature points, we introduce the feature refiltering procedure based on the RISC-V core according to the sorting algorithm (***Top-K ranking problem***). To guarantee the runtime of feature creation procedure, we have compared the three working modes, namely case1, case2, and case3, which are shown in Fig. 6. The case1 represents the serial working mode of KG and DG without refiltering task-sharing of RISC-V core. The case2 represents the serial working mode of KG and DG with refiltering task-sharing of RISC-V core. The case3 represents the pipelined working mode of KG and DG with refiltering task-sharing of RISC-V core, as shown in Fig. 4. In this way, the whole

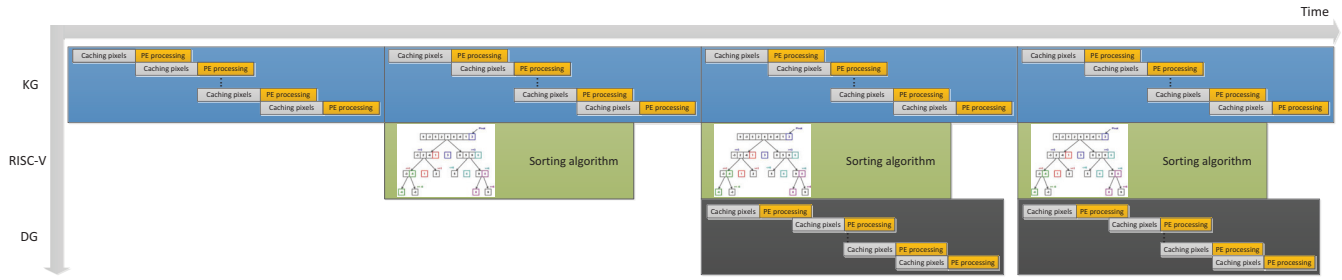


Fig. 4: Computation scheme. The blue frame, the green frame and the black frame represent the task sharing flow diagrams of KG, RISC-V core and DG, respectively. The blue frame includes PE processing and caching pixels processes and is identified as yellow and gray, respectively. The green frame describes the sorting algorithm which is the top k ranking problem, in which case the algorithm complexity is much smaller than O(nlogn). The black frame includes PE processing and caching pixels processes and is identified as yellow and gray, respectively. Therefore, not only KG and DG have realized pipeline processing, but also the entire KG, RISC-V core and DG have realized pipeline processing.

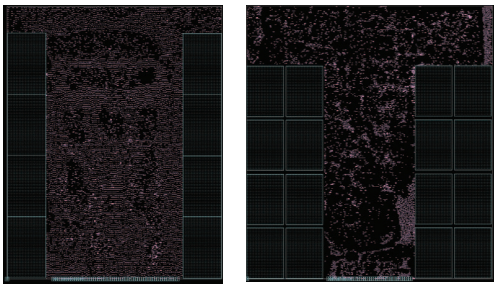


Fig. 5: (a) The layout design of KG; (b) The layout design of DG.

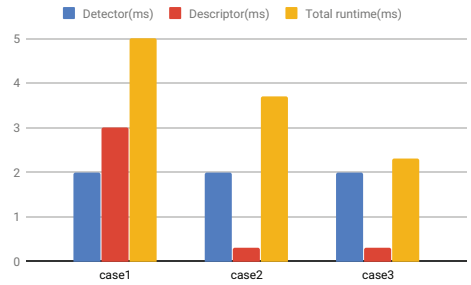


Fig. 6: The runtime comparisons of case1, case2 and case3. The total runtime colored with yellow includes the combined runtime of Detector(KG), Descriptor(DG), and the RISC-V core.

feature creation task can be optimized the most. The total runtime results of case1, case2 and case3 are 5ms, 3.7ms and 2.3ms respectively. We compare the HcveAcc performance and power consumption results with the implementations on an Intel i7 CPU processor, the ARM platform, the FPGA [4], and the eSLAM accelerator [3], as shown in TABLE II. The results of speedup are according to the ***Throughput*** column data. Compared with eSLAM, the speedup of HcveAcc can be up to 4.3X. The power consumption of HcveAcc with one RISC-V core and related memory is 0.181 *W* according to the EDA tool which is much more energy-efficient.

### *B. Accuracy Correlation*

*1) KG Accuracy:* We use the high-level modeling C language to describe the model, which ensures the consistency with the hardware implementation accuracy, including FAST12 calculation and NMS function module integration, as shown in Fig. 7. Since there is no quantitative selection, the hardware implementation results can be consistent with the calculation results of the OpenCV software library. The Fig. 7 (a) shows an example of the feature detection procedure without NMS module, and the key-point number is 4208. The Fig. 7 (b) shows the feature detection procedure with NMS module, and the key-point number is 1270. By introducing an NMS module, the complexity of subsequent computational processes is reduced.

TABLE II: The comparison of performance and power results.

| *Platform* | *Image Resolution* | *Frame Latency (ms)* | *Throughput (MPix/s)* | *Speedup Ratio* | *Power (W)* |
|---|---|---|---|---|---|
| CPU [3] | 640*480 | 32.5 | 9.5 | 15.4 | 47 |
| ARM [3] | 640*480 | 291.6 | 1.1 | 133.2 | 1.574 |
| FPGA [4] | 640*480 | 14.8 | 20.1 | 7.3 | 4.6 |
| eSLAM [3] | 640*480 | 9.1 | 33.8 | 4.3 | 1.936 |
| HcveAcc | 752*480 | 2.3 | 146.5 | N/A | 0.181 |


Fig. 7: The comparison of feature detection with or without NMS module. All of key-points are marked with green color.

*2) DG Accuracy:* The DG module is responsible for generating a 256bits descriptor with angle calculation. The actual calculation employs double-precision floating-point that is costly and is not suitable for hardware implementation. To evaluate the accuracy of DG hardware implementations, we use different angle discretization to run matching experiments between the original image and rotated image. In TABLE III, we have listed the max Hamming distance (***Max Distance***), the total key-point number (***Total***) for matching, the

(***Matched***) key-point number, and the error rate (***Error Rate***) for matching. The eSLAM work [3] uses the 12 degrees as the minimum angle discretization to reduce the computation cost significantly, as shown in the ***Angle Discretization*** column, the error rate evaluated is 70.20% for this experiment. We have adopted the one degree as the discretization parameter to reduce the error rate as much as possible which is also hardware-friendly. As the value of angle discretization gets smaller, the computational overhead of the DG hardware increases. The DG resource overhead with one-degree discretization in the synthesis process is about 3X of 12 degrees discretization. In the case where the number of PEs is consistent, the runtime overhead of the DG is about 50X that of the KG. Therefore, we have the motivation to design the whole feature creation for tracking task in the pipelined mode.

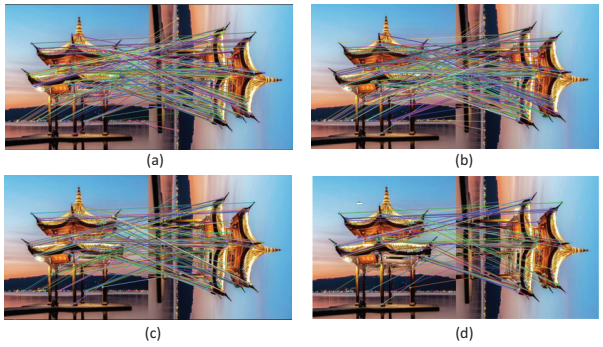


Fig. 8: The matching graphs with using different precision of angle calculation.

TABLE III: The comparison of hardware utilization.

| *Angle Discretization* | *Max Distance* | *Total* | *Matched* | *Error Rate* |
|---|---|---|---|---|
| 20 | 76 | 500 | 149 | 70.20% |
| 12 | 60 | 500 | 278 | 44.40% |
| 5 | 5 | 500 | 493 | 1.40% |
| 1 | 4 | 500 | 498 | 0.40% |

*3) ORB-SLAM Accuracy:* To evaluate the ORB-SLAM accuracy correlation, we have established the high-level programming language description of the entire HcveAcc task and integrated it into the ORB-SLAM system. We run experiments based on the EuRoC and TUM dataset [1], [2]. The absolute translation root-mean-square error (*RMSE*) metric is used to quantify the trajectory error of ORB-SLAM system, and we also list maximum trajectory error (*max*) results. In Fig. 9, we have picked the medium difficulty dataset in EuRoC (V1_02_medium), and the relatively difficult dataset in TUM (fr1/room) which is also selected in eSLAM work [3]. The Fig. 9 (b) and Fig. 9 (d) are both the trajectory error graphs between the ground truth trajectory and the estimated trajectory by using our HcveAcc solution. We can see that the results of *RMSE* are acceptable and the *max* error can be controlled within $1m$. The *max* value is 4.9% worst than the original ORB-SLAM, while the error percentage is about 30.9% for the same dataset in eSLAM work [3]. In this way, the hardware design flow for ORB-SLAM can be considered to have the consistency of accuracy within a certain range.

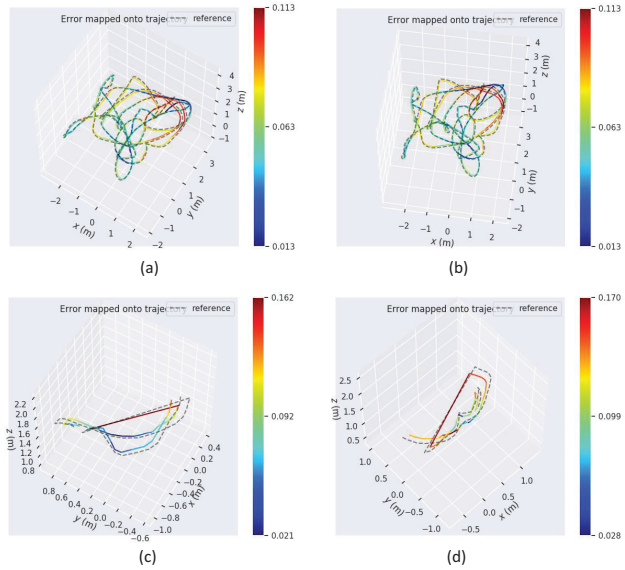


Fig. 9: (a) The original ORB-SLAM trajectory of EuRoC V1_02_medium ($RMSE = 0.064, max = 0.113(m)$) ; (b) The modified ORB-SLAM trajectory of EuRoC V1_02_medium ($RMSE = 0.069, max = 0.498(m)$) ; (c) The original ORB-SLAM trajectory of TUM fr1/room ($RMSE = 0.090, max = 0.162(m)$) ; (d) The modified ORB-SLAM trajectory of TUM fr1/room ($RMSE = 0.118, max = 0.170(m)$).

## V. Conclusion

In this paper, a hardware accelerator, the *HcveAcc*, is proposed for exploring tracking task optimization in visual SLAM system. Compared with state-of-the-art FPGA solutions, the HcveAcc can help perform high-density feature extraction and high-precision descriptor generation with 4.3X speedup and $181mW$ power cost.

## References

[1] R. Mur-Artal, J. M. M. Montiel, and J. D. Tardos, "Orb-slam: a versatile and accurate monocular slam system," *IEEE transactions on robotics*, vol. 31, no. 5, pp. 1147–1163, 2015.

[2] R. Mur-Artal and J. D. Tardós, "Orb-slam2: An open-source slam system for monocular, stereo, and rgb-d cameras," *IEEE Transactions on Robotics*, vol. 33, no. 5, pp. 1255–1262, 2017.

[3] R. Liu, J. Yang, Y. Chen, and W. Zhao, "eslam: An energy-efficient accelerator for real-time orb-slam on fpga platform," in *Proceedings of the 56th Annual Design Automation Conference 2019*. ACM, 2019, p. 193.

[4] W. Fang, Y. Zhang, B. Yu, and S. Liu, "Fpga-based orb feature extraction for real-time visual slam," in *2017 International Conference on Field Programmable Technology (ICFPT)*, Dec 2017, pp. 275–278.

[5] S.-K. Lam, G. Jiang, M. Wu, and B. Cao, "Area-time efficient streaming architecture for fast and brief detector," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 66, no. 2, pp. 282–286, 2018.

[6] T. H. Pham, P. Tran, and S.-K. Lam, "High-throughput and area-optimized architecture for rbrief feature extraction," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 27, no. 4, pp. 747–756, 2018.

[7] E. Rublee, V. Rabaud, K. Konolige, and G. R. Bradski, "Orb: An efficient alternative to sift or surf." in *ICCV*, vol. 11, no. 1. Citeseer, 2011, p. 2.

[8] P. K. Meher, J. Valls, T.-B. Juang, K. Sridharan, and K. Maharatna, "50 years of cordic: Algorithms, architectures, and applications," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 56, no. 9, pp. 1893–1907, 2009.