A Runtime Configurable Hardware Architecture for Computing Histogram-based Feature Descriptors

Murad Qasaimeh, Joseph Zambreno and Phillip H. Jones
Department of Electrical and Computer Engineering, Iowa State University, Ames, Iowa, USA 50010
Email: {qasaimeh, zambreno, phjones}@iastate.edu

***Abstract*—Feature description is an essential component of many computer vision applications. It encodes the visual contents of images in a manner that is robust against various image transformations. Computing these descriptors is computationally expensive, which causes a performance bottleneck in many embedded vision systems. Although many hardware architectures have been proposed to accelerate feature description computation, most target a single feature description algorithm under specific constraints. The lack of flexibility of such implementations increases development effort if deployed applications need to be modified or upgraded. In this paper, we propose a software configurable hardware architecture capable of computing different types of histogram-based feature descriptors without the need for re-synthesizing the hardware. The architecture takes advantage of data streaming to reduce the computational complexity of computing this class of descriptor. To illustrate the efficiency of our architecture, we deploy two of the most commonly used descriptors (SIFT and HOG) and compare their quality with software implementations. The architecture is also evaluated in terms of execution speed and resource usage and compared with dedicated hardware architectures. Our flexible architecture shows a speed up of 3× and 5× compared to state-of-the-art dedicated hardware architectures for SIFT and HOG, with resource usage overheads [LUTs, FFs, and DSPs] of [1.1×, 15×, and 1.6×] and [6.4×, 7×, and 32×] for SIFT and HOG, respectively.**

***Keywords*—*Local Feature, Descriptors, Histogram, FPGA*.**

I. Introduction

In computer vision, image features refer to salient points within a scene that are distinctive, repeatable and have enough intensity variation to be tracked reliably between images even under different image transformations and distortions, such as scaling, rotation, shearing, etc. Two processes are associated with image features: feature detection, and description. In feature detection, every pixel is examined to check if there is a feature of a given type (e.g. edges, corners, blobs) present at that pixel location. In feature description, the visual content around a detected feature is captured by computing robust numeric representations from the surrounding pixels' low-level information such as intensity, gradients, local binary pattern (LBP), etc. Image features are used in many applications, such as object detection and recognition, image retrieval, 3D reconstruction, and image mosaicing.

Histogram-based feature descriptors are the most commonly used feature description type due to their distinctiveness and robustness to various image transformations. However, computing these descriptors is computationally expensive and is the bottleneck of many vision systems [1]. For this reason, many hardware architectures have been proposed in the literature to accelerate this class of descriptor. Although these architectures achieve a high performance, most are hard-wired to support a specific histogram-based feature descriptor configuration, and cannot be modified without completely re-synthesizing and reprogramming the FPGA. These architectures are realized as static hardware pipelines optimized for only a single application. The lack of programmability in these architectures increases development effort when the deployed applications need to be modified or upgraded.

Implementing a flexible hardware architecture capable of computing a wide range of histogram-based descriptor variants has many advantages. For example, in robotic applications each feature description algorithm has been designed with strengths and weaknesses that make them more efficient in some scenarios compared to others [2]. Having a hardware accelerator that can be re-configured at runtime to transition between many of these algorithms when the environment changes is advantageous as compared to fixed hardware accelerators. A configurable hardware architecture can make prototyping new applications easier and faster for application developers. Changing the algorithmic structure for an existing application or designing a new application would not require writing HDL code, which can be prone to error and time consuming. Moreover, a configurable architecture can be used as an IP core to speed up prototyping embedded vision systems, thus reducing development effort.

Contributions. In this paper, we take advantage of common characteristics between different feature description algorithms to build a generic and flexible hardware accelerator. A major concern that arises when implementing a generic hardware architecture is hardware resources overhead. For this reason, we propose optimization techniques to reduce the computational complexity and hardware resource usage. The proposed architecture leverages data streaming to reduce the cost of computing and updating histogram values. The main contributions of this paper are: (1) Implementing a hardware architecture capable of computing several feature description algorithms using a single datapath. The architecture is configurable in terms of patch sizes, number of regions, and number of bins per region, and (2) Using optimization techniques to reduce the complexity of computing 2D histograms from O(n^2) to O(n).

Organization. The rest of the paper is organized as follows. Section II presents related work and compares it to our approach. In Section III, we present the histogram binning technique. Section IV provides a detailed description of our hardware architecture and its main building blocks. In Section V, we discuss the verification and evaluation methods used to evaluate the proposed architecture and compare it with related works. Finally, Section VII concludes the paper with outlooks for future work.

II. Related Work

Generality and efficiency of computing systems are often inversely related to each other. The more general a computing system is, in terms of executing a wide range of tasks, the less efficient it tends to be in performing a specific task as compared to a system designed for only one task. It is important to find the right balance between these often competing characteristics. A general-purpose computing system such as CPUs are highly programmable but usually cannot achieve the highest performance. While dedicated hardware architectures can achieve high performance, but they are not programmable. Designers might be able to compromise small degradation in performance to increase system programmability.

Many dedicated hardware architectures for histogram-based feature descriptors with real-time performance have been proposed in the literature [3][4][5][6][7], but a configurable hardware architecture capable of supporting more than one descriptor algorithm with real time performance has not been proposed. However, efforts for implementing flexible hardware architectures for other image processing algorithms have been made [8][9][10]. In [8], a generic architecture for image feature detectors was presented. They proposed a generic hardware architecture that integrates two image feature detectors (Harris and SUSAN) in a single datapath. Another example of a generic architectures appears in [9]. They proposed a generic VHDL template for fast window-based stereo block matching correlation. In [10], the authors proposed a parallel hardware architecture for computing integral histogram images. The architecture was generic in terms of its input types: grayscale intensity, gradient, or binary pattern.

III. Histogram Binning Technique

Image gradients refer to the directional change in pixel intensity and can be represented using two values: a magnitude (*M*) and orientation (θ). The histogram binning technique used in our architecture take advantage of overlapping between windows as a sliding window moves across images. This technique reduces the number of operations required to compute histograms by avoiding redundant operations between consecutive windows. To compute histograms of gradients, the entire range of gradient orientation ($0° \leq \theta < 360°$) is divided into classes (also called bins) from θ_1 to θ_B, where *B* is number of bins. The first histogram, H_1, of a window of size N×N pixels is computed as shown in Equation (1). Figure 1 shows an example for a 3×3 window. It shows computing histograms for two consecutive windows as the window slides one pixel to the right. The new histogram values can by computed by subtracting the left-most column and adding the right-most column to the previously computed histogram. This concept is applicable not only for regions with uniform shape, but also can be applied to any connected regions. Using this technique, we can compute histograms for a window of size N×N using only N addition and N subtraction operations O(2N) instead of using N×N addition operations $O(N^2)$ in the straightforward approach. Equations (1-3) show the mathematical formulation for this technique, where *i* and *j* represent the index of the upper left corner pixel. Equation (1) shows the equation to compute histograms of the first window $H_1(\theta_k)$. Equations (2-3) show how the histogram of the next window $H_2(\theta_k)$ can be computed by only updating the previous histogram $H_1(\theta_k)$. $\forall \theta_k \in \{\theta_1, \theta_2, ..., \theta_n\}$

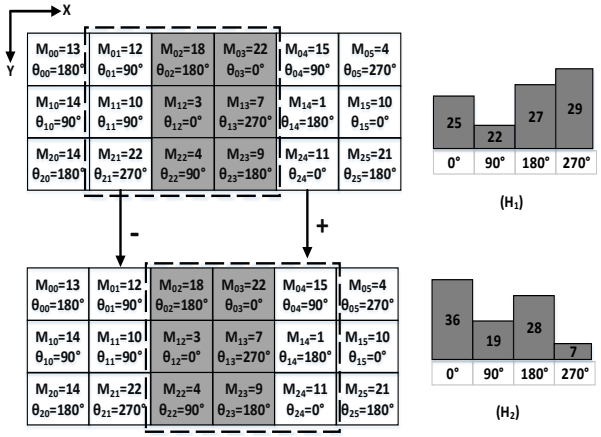

Fig. 1: Computing 2D Histogram of Gradients

$$H_1(\theta_k) = \sum_{x=i}^{i+N} \sum_{y=j}^{j+N} M_{x,y}, \quad \text{if } \theta_k \leq \theta_x < \theta_{k+1} \tag{1}$$

$$H_2(\theta_k) = \sum_{x=i}^{i+N} \sum_{y=j}^{j+N} M_{x,y} - \sum_{y=j}^{j+N} M_{i,y} + \sum_{y=j}^{j+N} M_{i+N+1,y} \tag{2}$$

$$H_2(\theta_k) = H_1(\theta_k) - \sum_{y=j}^{j+N} M_{i,y} + \sum_{y=j}^{j+N} M_{i+N+1,y} \tag{3}$$

IV. Proposed Hardware Architecture

A 2D array of processing elements arranged in a systolic structure is the main building block of our architecture. It is used to buffer gradient pixels generated by the gradient computation block, quantizes their orientations, and compute histograms. The internal architecture of one PE is shown in Figure 2. Each PE sends and receives data from/to its four neighbors using four input ports: R(in), L(in), U(in), and D(in), and four output ports: R(out), L(out), U(out), and D(out). The size of these ports equal $NBins \times 2 \times 8$, where NBins is number of histogram bins, each bin has 2 values for gradient magnitude (8 bits) and orientation (8 bits). Each PE stores one gradient magnitude and orientation (M and θ) locally. Every time a new gradient is computed, the PE updates its gradient value with the values coming from $pixel(in)$ and sends the old gradient value to its $pixel(out)$ port. It also contains a memory array (H) of size $NBins$, which is used to store the computed histogram values.

For every new gradient $pixel(in)$, a PE quantizes its gradient orientation into the nearest two angular bins. For example, if the PE is configured to compute histograms with 4 bins, that means the bin angles are : 0°, 90°, 180°, and 270°. If the input gradient orientation is 215°, the orientation will be quantized into the closest two angles, which are 180°, and 270°. The gradient magnitude will be divided into two values using linear interpolation. Because 215° is closer to 180°, the gradient magnitude for this bin will be higher than the 270° bin. The quantitation unit is shown in Figure 2 with symbol (Q). It reads gradient angle θ and selects two bins of the *DEMUX0* output. 8 bits ($S_7 - S_0$) are used to configure each PE. These bits configure four MUXs and one DEMUX. It allows PEs to compute different permutations of its four inputs and send the result to one of its four output ports. For example, S_0 controls MUX0 output to be D(in) or 0. The same concept applies to bits S_{5-1}. We used -R(in) to subtract the values that leave each region (algorithm proposed in Section

IV), because we assume that the data is streaming in razer-scan (left to right). Finally, S_{7-6} select the output port. For example, when $S_{7-6} = 00$, the computed value will be sent to the D(out) port, and when $S_{7-6} = 01, 10, 11$, the computed value will be sent to U(out), L(out), R(out), respectively.

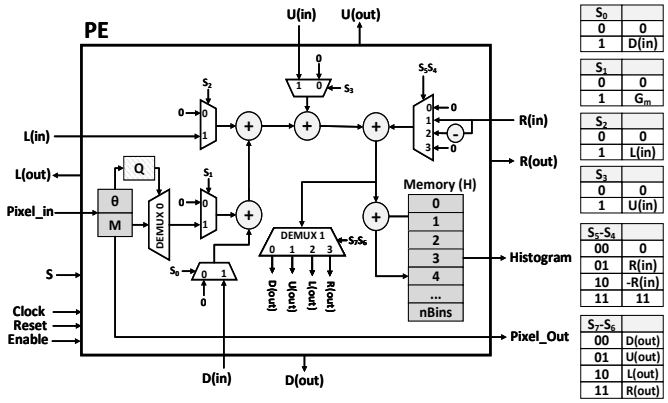

Fig. 2: Processing Element (PE) Architecture and Its Configurations

To compute histograms for a window of size N×N, we construct the 2D array architecture with N×N PEs arranged in a systolic structure, as shown in Figure 3. It shows an example of 4×4 array of PEs, where each PE is connected to its four neighbors. For simplicity, in this example, we configured PEs to compute histograms of three regions with number of bins equal to 1. The upper right section of Figure 3 shows the PEs' configuration for computing histograms of three regions. PEs located on the right edge of each region are configured to send their gradient pixels to L(out). In our example, PEs (PE_{01}, PE_{11}, PE_{21}, PE_{03}, PE_{13}, PE_{23}, and PE_{33}) are configured to send their gradient pixels to L(out). The architecture needs to keep track of these pixels, because they need to be subtracted from the total histogram to keep the computed histograms correct. PEs located within regions such as (PE_{31}, and PE_{32}) are responsible of passing the values coming from their R(in) to L(out). PEs located at the left edge of each region such as (PE_{00}, PE_{10}, PE_{20}, PE_{30}, PE_{02}, PE_{12}, and PE_{22}) are responsible for subtracting the values coming to their R(in) ports from their gradient pixel. The computed results are then passed to their U(out). For each region, one PE is needed to keep track of the computed histogram value. In our example, we used (PE_{00}, PE_{02}, and PE_{20}) to compute histograms of the three regions (H_1, H_2, and H_3).

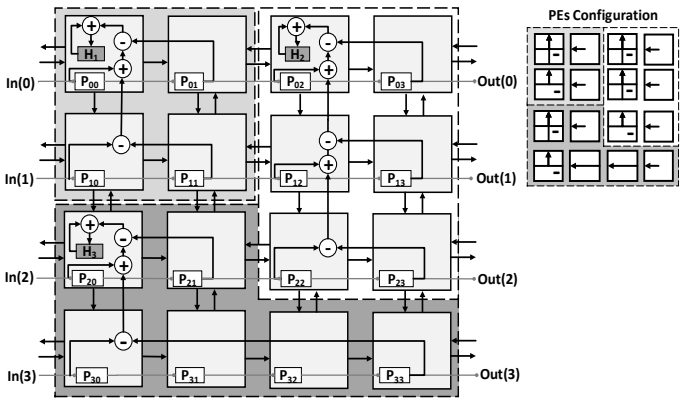

Fig. 3: An Example of 4×4 PEs (3 Regions, 1 Bin). The Upper Right Section Shows each PE's Configuration Key

V. Evaluation and Verification

The proposed architecture is evaluated in terms of accuracy, execution speed, and resource usage. The accuracy of the proposed architecture is evaluated for two applications: (1) SIFT feature matching, and (2) Pedestrian detection using HOG and an SVM classifier.

A. SIFT Feature Matching

To validate our architecture, we compared the quality of SIFT descriptors computed by software implementation (MATLAB) and our architecture using a feature matching problem. We used the 8 benchmarks of Oxford dataset [11] in this experiments. Matching ability of a descriptor is measured by the area under the ROC curve. An area of 1 represents a perfect descriptor, and an area of 0.5 represents random guesses. Table 1 also shows the computed areas under these curves. To compare ROC curves, we used a statistical significance test (Hanley method). It shows the difference in area under the ROC curves are insignificant for the matching problem. The results of hardware and software do not exactly match due to using fixed point representation for gradient magnitude and orientation instead of using floating point and using linear interpolation instead of trilinear interpolation. An exact match with software can be achieved at the cost of additional hardware resources.

TABLE I: Area Under ROC (AUC) for Software and Hardware

Benc	ubc	leu.	bikes	wall	graf	trees	boat	bark
Soft.	0.94	0.88	0.85	0.79	0.56	0.82	0.67	0.58
Hard.	0.94	0.88	0.84	0.79	0.54	0.81	0.65	0.56

B. Pedestrian Detection

In this experiment, we used a HOG descriptor with a linear SVM classifier to detect pedestrians in a diverse set of images from the CVC-02 dataset [12]. To validate our architecture, we implemented a MATLAB model that represents precisely the behavior of the implemented hardware. We ran the experiment on 250 images containing 587 annotated pedestrians, the proposed hardware achieved detection accuracy of 86.2% with 4.7% false negative, while the software implementation (floating point) achieved accuracy of 86.2% with 4.7% false negative. This shows that our hardware implementation achieved the same accuracy as the software implementation for the pedestrian detection application.

C. Comparison with Dedicated Hardware Architectures

To measure the performance of the proposed architecture, we compared the execution time of our implementation with OpenCV implementations of SIFT and HOG algorithms running on an ARM Dual-Core Cortex-A9. We also compare execution times with other hardware architectures implementing the SIFT and HOG algorithms. Table II shows the time required to compute one SIFT descriptor, frame sizes, frame rates, and maximum number of SIFT descriptors that can be computed within the frame rate mentioned (KP Ratio), where KP represents the percentage of pixels that can be processed at a specific frame rate. Table II shows that our architecture with window sizes, 16×16, 8×8, and 4×4 achieved a speed-up of 4-5 order of magnitudes compared to the ARM processor and 3-4 order of magnitudes compared to the Intel Core i7. When the fully parallel architecture P(16×16) is used, our architecture outperforms the work done in [5] by 3×, because our architecture can run at 167 MHz while the work in [5] is running at

21.7 MHz (one cycle= 46ns) only. However, our architecture shows an overhead in resource usage of 1.1×, 15×, and 1.6× for LUT, FF, and DSPs, as shown in Table III. This is due to the generality of our architecture to support many algorithms instead of only one. For the sequential implementaion P(8×8), our architecture can compute one SIFT descriptor in $0.37\mu s$ (64 cycles) which outperforms other hardware architectures by 6.1×, but with a resource overhead of 1.2×, 2.7×, and 1.5× for LUT, FF, and DSPs compared to the fastest sequential implementation in [4]. In the sequential implementation of P(4×4), our architecture can compute one SIFT descriptor in $0.75\mu s$ (128 cycles), which achieved a speed up of 2.9×, with resource overhead of 0.8×, 0.68×, and 1.6× for LUT, FF, and DSPs.

TABLE II: SIFT Descriptor Extraction Time

	Exec. time	Frame Size	Frame Rate
ARM A9	5.13 ms	800×640	0.11 fps
Jiang [4]	2.23 μs	512×512	150 fps, feat.<2900
John [5]	46 ns	640×480	70 fps (parallel)
P(16×16)	5.9 ns	800×640	195 fps (parallel)
P(08×08)	0.37 μs	800×640	60 fps, feat.<2600
P(04×04)	0.75 μs	800×640	30 fps, feat.<2600

TABLE III: SIFT Hardware Architectures

Paper	FPGA	Max Freq	LUTs	FFs	DSPs
Jiang [4]	Virtex-5	79.4MHz	26,398	10,310	89
John [5]	CycloneIV	21.7MHz	120,917	6,719	77
P(16×16)	ZYNQ-7	167 MHz	128,731	102,092	130
P(08×08)	ZYNQ-7	167 MHz	32,871	28,004	130
P(04×04)	ZYNQ-7	167 MHz	10,093	9,542	130

Table IV compares the performance of the proposed architecture with an OpenCV implementation of HOG running on an ARM Dual-Core Cortex-A9, Intel Core i7 processors, and dedicated FPGA architectures proposed in the literature. It shows frame sizes and the maximum frame rates achieved by each work. It order to find a common metric, we computed pixels/second from frame rates and sizes. To compute our architecture frame rate, we multiply the number of HOG cells in one frame by the number of clock cycles needed to finish one cell, and divide the result by the maximum operational frequency. Table IV shows that our architecture achieved a speed-up of 1-2 order of magnitude compared to the ARM processor and 1 order of magnitude compared to the Intel Core i7. It also shows that our architecture P(16×16) outperforms other works by 3-46×, because it can compute one HOG cell in one clock cycle, but at the expense of resources. P(16×16) has resource usage overheads of 6.4×, 17×, and 32× for LUT, FF, and DSPs compared to the latest implementation in [7]. Our sequential architectures, P(8×8) and P(4×4), shows a compromise in terms of performance to reduce the resource usage overhead as shown in Table V.

TABLE IV: HOG Descriptor Extraction Time

	Frame Size	Frame Rate	Pixels/Second
ARM A9	1920×1080	0.63	1.3×10^6
Hahnle [6]	1920×1080	64	132.7×10^6
Jens [7]	1920×1080	40	82.9×10^6
P(16×16)	1920×1080	192	398.1×10^6
P(08×08)	1920×1080	48	99.5×10^6
P(04×04)	1920×1080	12	24.8×10^6

TABLE V: HOG Hardware Architectures

Paper	FPGA	Freq.	LUTs	FFs	DSPs
Hahnle[6]	Virtex-5	270MHz	5,188	5,176	49
Jens [7]	XC7z020	82.2MHz	21,297	5,942	4
P(16×16)	ZYNQ-7	167MHz	128,731	102,092	130
P(08×08)	ZYNQ-7	167MHz	32,871	28,004	130
P(04×04)	ZYNQ-7	167MHz	10,093	9,542	130

VI. Conclusion

In this paper, we proposed a configurable hardware architecture for computing different histogram-based feature description algorithms. The proposed architecture is configurable in terms of window size, number of regions, number of bins per region, and the pattern of these regions. We implemented different optimization techniques to reduce hardware resources and computational complexity for computing histograms of gradients. The experimental results show that our architecture can be used in a wide range of computer vision applications. This architecture can also be used to compute histograms for other low-level information such as local binary pattern (LBP) or grayscale intensity without any modifications. Our future work will investigate the possibility of extending the proposed architecture to compute other kinds of window-based algorithms in computer vision.

References

[1] K. Mikolajczyk and C. Schmid, "A performance evaluation of local descriptors," *IEEE transactions on pattern analysis and machine intelligence*, vol. 27, no. 10, pp. 1615–1630, 2005.

[2] S. Krig, "Interest point detector and feature descriptor survey," in *Computer Vision Metrics*, pp. 187–246, Springer, 2016.

[3] J. Wang, S. Zhong, L. Yan, and Z. Cao, "An embedded system-on-chip architecture for real-time visual detection and matching," *IEEE transactions on Circuits and Systems for Video Technology*, vol. 24, no. 3, pp. 525–538, 2014.

[4] J. Jiang, X. Li, and G. Zhang, "SIFT hardware implementation for real-time image feature extraction," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 24, no. 7, pp. 1209–1220, 2014.

[5] J. Vourvoulakis, J. Kalomiros, and J. Lygouras, "Fully pipelined FPGA-based architecture for real-time SIFT extraction," *Microprocessors and Microsystems*, vol. 40, pp. 53–73, 2016.

[6] M. Hahnle, F. Saxen, M. Hisung, U. Brunsmann, and K. Doll, "Fpga-based real-time pedestrian detection on high-resolution images," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, pp. 629–635, 2013.

[7] J. Rettkowski, A. Boutros, and D. Göhringer, "Real-time pedestrian detection on a xilinx zynq using the hog algorithm," in *ReConFigurable Computing and FPGAs (ReConFig), 2015 International Conference on*, pp. 1–8, IEEE, 2015.

[8] A. Hernandez-Lopez, C. Torres-Huitzil, and J. J. Garcia-Hernandez, "Fpga-based flexible hardware architecture for image interest point detection," *International Journal of Advanced Robotic Systems*, vol. 12, no. 7, p. 93, 2015.

[9] K. Häublein, M. Reichenbach, and D. Fey, "Fast and generic hardware architecture for stereo block matching applications on embedded systems," in *ReConFigurable Computing and FPGAs (ReConFig), 2014 International Conference on*, pp. 1–6, IEEE, 2014.

[10] S. Yin, P. Ouyang, T. Chen, L. Liu, and S. Wei, "A configurable parallel hardware architecture for efficient integral histogram image computing," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 24, no. 4, pp. 1305–1318, 2016.

[11] K. Mikolajczyk and C. Schmid, "A performance evaluation of local descriptors," *IEEE transactions on pattern analysis and machine intelligence*, vol. 27, no. 10, pp. 1615–1630, 2005.

[12] D. Gerónimo, A. D. Sappa, D. Ponsa, and A. M. López, "2d–3d-based on-board pedestrian detection system," *Computer Vision and Image Understanding*, vol. 114, no. 5, pp. 583–595, 2010.