PATTERN COMPRESSION OF FAST CORNER DETECTION FOR EFFICIENT HARDWARE IMPLEMENTATION

Keisuke Dohi, Yuji Yorita, Yuichiro Shibata, Kiyoshi Oguri

Graduate School of Engineering, Nagasaki University
1-14 Bunkyo-machi, Nagasaki 852-8521, Japan
email: {dohi,yrtyj}@pca.cis.nagasaki-u.ac.jp,{shibata,oguri}@cis.nagasaki-u.ac.jp

ABSTRACT

This paper shows stream-oriented FPGA implementation of the machine-learned Features from Accelerated Segment Test (FAST) corner detection, which is used in the parallel tracking and mapping (PTAM) for augmented reality (AR). One of the difficulties of compact hardware implementation of the FAST corner detection is a matching process with a large number of corner patterns. We propose corner pattern compression methods focusing on discriminant division and pattern symmetry for rotation and inversion. This pattern compression enables implementation of the corner pattern matching with a combinational circuit. Our prototype implementation achieves real-time execution performance with 7∼9% of available slices of a Virtex-5 FPGA.

1. INTRODUCTION

Recently, Augmented Reality (AR) technology, which appends additional information to a view of a real-world environment such as image data input by a video camera, is getting a lot of attention. Obviously, fast and correct understanding of reality is a key for augmentation of a view of the reality. The basis of this technology is detection of points of interest or feature points. While special markers are employed to track feature points in some AR implementation [1], the Parallel Tracking and Mapping (PTAM) system enables marker-less detection using a corner detection algorithm executed on a multi-core CPU [2]. Among various corner detection algorithms proposed so far [3] [4], the PTAM system employs the Features from Accelerated Segment Test (FAST) corner detection algorithm [5].

The PTAM system, however, requires a desk-top multi-core CPU for real-time execution, and thus is not suited for direct implementation on embedded hardware. In this paper, we show highly-efficient FPGA implementation of the FAST corner detection algorithm used in the PTAM system, focusing on the fact that algorithm can be considered as a kind of a huge table look-up of corner patterns. Although the table is too large to be straightforwardly implemented on an FPGA, we propose logic compression approaches to enable the FPGA implementation. While FPGA implementation of other corner detection algorithms has been widely investigated [6, 7, 8], work on the FAST corner detection algorithm is relatively few [9]. As far as our knowledge goes, our work is the first attempt of FPGA implementation of the machine-learned FAST algorithm utilized in the PTAM, with was tailored for AR systems.

2. FAST CORNER DETECTION

The FAST corner detection algorithm was proposed to quickly detect corners appeared in given image data [10]. The original FAST-N algorithm compares the intensity of a corner candidate p with each point on a 16-point ring that surrounds p as Fig.1 shows. The candidate point is detected as a corner, if N contiguous points on the ring are all brighter or all darker than p.

In addition, in order to enable efficient corner detection especially for $N < 12$, use of corner patterns organized by machine learning rather than mere contiguous points has been proposed [5]. The machine-learned FAST corner detection gives a state for each point on the ring $x \in \{0, 1, \ldots 15\}$ that surrounds the corner candidate point p. The state of the point x relative to p takes one of three states according to the following manner:

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

where I_p shows the intensity of p, $I_{p\to x}$ shows the intensity of the point x, and t denotes a threshold for intensity. Patterns consisting of 16 state variables $\{S_{p\to 15}, ..., S_{p\to 0}\}$ were classified into the corner patterns and the others, using the ID3 algorithm [11]. A decision tree which detects corners based on these patterns was also generated in C source code and released in public [5].

Hence, the machine-learned FAST corner detection can be considered as a binary table look-up whose domain is all combinations of the states for 16 pixels of $S_{p\to x}$. In this paper, we generated the same pattern tables in HDL code.

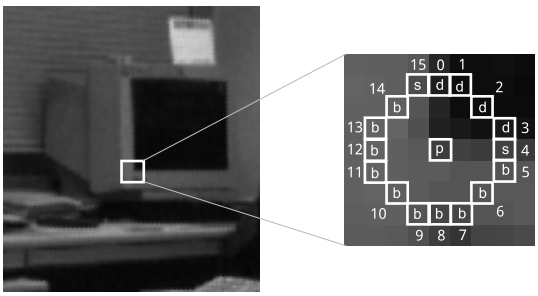

Fig. 1. Relations of center pixel p and surrounding pixels.

(a) (b)

Fig. 2. The results of FAST corner detection with (a) libcvd and (b) considering rotation and inversion.

The number of whole table entries is $3^{16} = 43,046,721$, while the numbers of valid corner patterns are 52,494 for FAST-9 and 15,554 for FAST-10.

3. CORNER PATTERN COMPRESSION

The corner pattern table is indexed by $\{S_{p\to 15}, ..., S_{p\to 0}\}$. According to Equation (1), each state can have one of the three states, thus the bit width required for a table index (address) is at least: $\lceil \log_2 3^{16} \rceil = 26$.

If we take this 26-bit approach, we firstly need to convert 16 state variables into a 26-bit pattern and also need $2^{26} = 64$-Mbit memory to store the patterns. Common FPGAs are not able to accommodate this memory size with on-chip RAMs. An alternative way is to implement this mapping as combinational circuits on an FPGA instead of the on-chip RAM table. However, direct implementation attempt of the combinational circuits on a Virtex-5 XC5LVX50 with the ISE 12.2 tool resulted in failure due to an explosive growth in logic complexity.

To cope with this problem, we propose techniques to compress the corner patterns so as to make it possible to implement the FAST corner detector as combinational logic circuits on an FPGA. In this paper, we propose and evaluate two approaches. The first approach splits Equation (1) into two pattern tables and carries out matching processes twice. This approach is able to detect exactly the same points with the original machine-learned FAST corner detector for any given image data, thus this method corresponds to loss-less compression. The second compression approach is based on the potential symmetry of the corner patterns for rotation and inversion. However, this approach may correspond to a lossy compression method. Note that these two methods can be used at the same time.

3.1. Equation splitting

This method was originally based on our assumption that only either pixel x that has $S_{p\to x} = d$ or $S_{p\to x} = b$ is important in the corner decision process with Equation (1). Thus we split the equation into two patterns as follows:

$$SD_{p\to x} = \begin{cases} \text{darker}, & I_{p\to x} \leq I_p - t \\ \text{other}, & \text{else} \end{cases} \quad (2)$$

$$SB_{p\to x} = \begin{cases} \text{brighter}, & I_p + t \leq I_{p\to x} \\ \text{other}, & \text{else} \end{cases} . \quad (3)$$

The corner pattern matching is executed twice; one for each equation. The focused pixel p is classified as a corner, if the pixel is detected as a corner with one of the two matching processes.

In order to verify the assumption behind this method, we exhaustively compared the table of this method with the original methods, the fast_corner_detect_9 function and the fast_corner_detect_10 function implemented in the libcvd library [12] used in the PTAM system. As a result, it was shown that the proposed and the original method were completely consistent, which means that the proposed FAST corner detection consisting of twice matching for 16-bit patterns $\{SD_{p\to 15}, ..., SD_{p\to 0}\}$ and $\{SB_{p\to 15}, ..., SB_{p\to 0}\}$ is equivalent to the original detection presented in [5].

Although this method effectively reduced the number of the corner patterns, the number of matching procedures was doubled in compensation for the patten reduction. Fortunately, however, our observations revealed there was room for further optimization. First, we have found the contents of the two corner pattern tables for SD and SB are completely the same. Therefore, we need only one table for each matching process. Second, we have found that the two pattern tables for SD and SB never detect the corner for the same candidate point, and that it is enough to perform the table look-up only for either SD or SB which contains more "1" bits for $N \geq 9$. By making the best use of these observations, both of the size of the pattern table and the number of matching processes can be halved.

3.2. Introduction of rotation and inversion

The second compression approach is based on another intuitive assumption that if a pixel p is a corner in a given image data, it should remain a corner even when the image is rotated or inverted (flipped). According to the way of expression of patterns described in the previous section, a 90-degree left rotated pattern can be written as: $\{S_{p\to 3}, ..., S_{p\to 0}, S_{p\to 15}, ...$

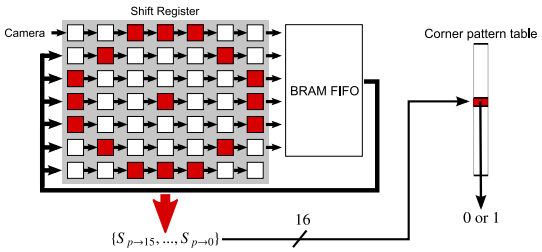

Fig. 3. The system diagram.

$S_{p\to4}\}$. In the same manner, the 180-degree rotated pattern and 270-degree rotated pattern can also be described by rotating the state variables in the original pattern. Also, the inverted pattern can be written as: $\{S_{p\to1}, S_{p\to2}, ..., S_{p\to14}, S_{p\to15}, S_{p\to0}\}$. Thus, we have a total of eight pattern expressions with rotation and inversion for each corner candidate pixel p. Admitting our assumption, the same detection result should be obtained for these eight patterns.

Using this method and the previously mentioned method at the same time, every pattern can be described as binary data, and thus we can chose the pattern which has the smallest binary as a representative pattern of corresponding eight isomorphic patterns. While the number of corner patterns is reduced, this compression method needs to convert a given state pattern into a corresponding representative isomorphic pattern, which will require additional logic resources.

Unfortunately, our verification experiments revealed that this compression method does not retain the original corner detection functionality. Therefore, we evaluated how the quality of results was affected. We used the standard test image the "Lenna" as a benchmark, and set the intensity threshold t to 50. For the benchmark image, the FAST-10 algorithm produced the same detection results even after this compression, but some differences were observed for the FAST-9. Fig.2(a) and Fig.2(b) show corner detection results produced with the original fast_corner_detect_9 function and this method, respectively. The true positive rate, the false positive rate, and the false negative rate were 100 %, 0.26 %, and 0 %, respectively. Although our second compression method was not equivalent to the original detection algorithm shown in [5] since the assumption on corner symmetry was not the case for the FAST-9, the difference of results was relatively small.

4. IMPLEMENTATION

To evaluate the proposed compression methods, we implemented an FPGA-based real-time FAST corner detection system. The data-flow of the system is fully pipelined and thus the input image data are processed at the same throughput with the frame rate of the camera. All Block RAMs (BRAMs) are used as FIFO modules; no random accesses are performed on these memory modules. The system needs intensity data for the focused pixel and surrounding 16 pixels at the same time. Therefore, we employed a streamed architecture consisting of shift registers made of flip-flops and line buffers using BRAM FIFOs as shown in Fig.3.

Since the shift registers illustrated in the region covered with a gray hatching consist of flip-flops, all of the data in this region can be accessed in parallel. The size of the region is set to seven-by-seven so that the intensity data for the focused pixel p and the 16-pixel ring can be kept. The focused pixel and the corresponding ring are moved on to the next on every clock cycle synchronized with the camera device and the DVI output. The corner pattern table compressed with aforementioned methods is implemented as computational logic circuits and indexed by the state pattern calculated from intensity data on the shift registers.

Table 1. Table size and # of corner patterns.

	Size	FAST-9	FAST-10
basic	2^{26}bits	52,494	15,554
split	2^{16}bits	1,026	513
symmetry	2^{16}bits	466	117

5. EVALUATION

The evaluation system was implemented on a ML501 prototype board equipped with a Xilinx Virtex-5 XC5VLX50 and a OmniVison OV9620 CMOS camera device. We used ISE 12.2 as development environment. Since the evaluated system employs the fully pipelined, it achieves the performance of 62.5 fps, provided that the maximum frequency of the circuits configured on the FPGA exceeds 25 MHz.

5.1. Number of corner patterns

At first, we evaluated how the number of corner patterns was reduced by the proposed compression methods. The size of the corner pattern table and the numbers of corner patterns for the FAST-9 and FAST-10 are summarized in Table 1.

The "basic" method shows the original corner patten table generated from the PTAM libcvd functions. The "split" corresponds to the method described in Section 3.1 in which patten equations are split. The "symmetry" method takes into account the pattern symmetry for rotation and inversion in addition to the "split" method. The "split" method significantly reduced the size of the corner pattern table from 64 Mbits to 64 Kbits. Meanwhile, the "symmetry" method reduced the number of the corner patterns not the table size.

Both "split" and "symmetry" methods produced positive results in terms of reduction in the corner patterns. While the "split" method is loss-less and is completely equivalent to the "base" method, it effectively eliminated 98% and 97%

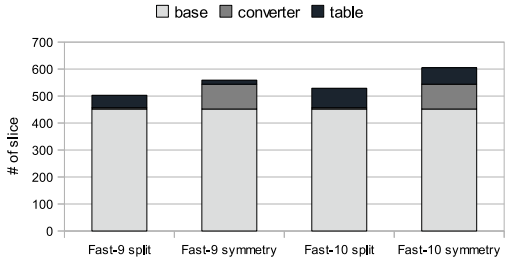

Fig. 4. Slice usage for each method.

Table 2. Maximum frequency.

FAST-9		FAST-10	
split	symmetry	split	symmetry
81.6 MHz	57.3 MHz	83.1 MHz	59.3 MHz

of the original corner patterns for FAST-9 and FAST-10, respectively. Additionally, the "symmetry" method reduced approximately the half of the corner patterns for FAST-9, while it reduced approximately three quarters for FAST-10. The difference in effectiveness is attributed the robustness of the "basic" method for rotation and inversion. The evaluation results suggest that the table of the original FAST-10 has more isomorphic patterns compared to the original FAST-9 and thus is more robust for rotation and inversion.

5.2. Hardware resource usage

Next, we evaluated the hardware amount utilized for FPGA implementation. Fig.4 shows the hardware resource usage and their breakdowns; "basic", "converter", and "table" correspond to areas for image pipelines, hardware that converters intensity data to a table index, and the corner pattern table implemented as combinational circuits.

The "split" was better than "symmetry" for both of the FAST-9 and FAST-10, since the converter to select a representative isomorphic state pattern required in the "symmetry" method occupied a large amount of hardware, while the hardware resources for the reduced pattern table did not make much difference. As a result, the compression method based on the pattern symmetry for rotation and inversion described in Section 3.2 was not effective for the FAST-9 and FAST-10. On the other hand, the "split" method was significantly effective, making it possible to directly implement the FAST corner detector on an FPGA, which was not feasible with the "basic" method.

5.3. Frequency

Table 2 shows the maximum frequencies of the circuits for each method. The "symmetry" was slower than the "split", since the logic that compares rotated and inverted patterns dominated the critical path. The performance of the "split" circuits corresponds to more than 200 fps, assuming the camera device can input image at that rate.

6. CONCLUSION

This paper presented FPGA implementation of the machine-learned FAST corner detection used in the PTAM system. Although the corner detection was considered as a kind of a table look-up, its corner detection table was too huge to directly implement with an FPGA. To cope with this problem, we proposed two table compression approaches; one for loss-less compression and the other for lossy compression. As a result, the number of corner pattern entries in the original FAST-10 table was reduced from 15,554 to 513 when hardware usage was minimum, consuming only approximately 9% of slices (529 slices) in Virtex-5 XC5LVX50. The FPGA implementation of the FAST corner detection achieved real-time throughput, showing effectiveness of our design approach with table compression. Our future work includes the FPGA implementation of a whole functionality of the PTAM as an embedded marker-less AR system.

7. REFERENCES

[1] "ARToolKit," http://www.hitl.washington.edu/artoolkit/.

[2] G. Klein and D. Murray, "Parallel tracking and mapping for small AR workspaces," in *Proc. ISMAR*, 2007.

[3] S. Smith and J. Brady, "SUSAN - A new approach to low level image processing," *International journal of computer vision*, vol. 23, no. 1, pp. 45–78, 1997.

[4] C. Harris and M. Stephens, "A combined corner and edge detector," in *Alvey vision conference*, vol. 15, 1988, p. 50.

[5] E. Rosten and T. Drummond, "Machine learning for high-speed corner detection," in *Proc. ECCV (1)*, 2006, pp. 430–443.

[6] C. Torres-Huitzil and M. Arias-Estrada, "An FPGA architecture for high speed edge and corner detection," 2000, pp. 112–116.

[7] C. Cabani and W. J. MacLean, "A proposed pipelined-architecture for fpga-based affine-invariant feature detectors," in *Proc. CVPRW*, 2006, p. 121.

[8] C. Claus, R. Huitl, J. Rausch, and W. Stechele, "Optimizing the susan corner detection algorithm for a high speed fpga implementation," in *Proc. FPL*, 2009, pp. 138–145.

[9] M. Kraft, A. Schmidt, and A. J. Kasinski, "High-speed image feature detection using fpga implementation of fast algorithm," in *Proc. VISAPP (1)*, 2008, pp. 174–179.

[10] E. Rosten and T. Drummond, "Fusing points and lines for high performance tracking." in *Proc. ICCV (2)*, 2005, pp. 1508–1511.

[11] J. Quinlan, "Induction of decision trees," *Machine learning*, vol. 1, no. 1, pp. 81–106, 1986.

[12] "CVD Projects," http://mi.eng.cam.ac.uk/~er258/cvd/.