An efficient FPGA implementation of the Harris Corner feature detector

Tak Lon Chao
The Dept. of Electronic Engineering
The Chinese University of Hong Kong
chaotaklon@gmail.com

Kin Hong Wong
The Dept. of Comp. Sci. and Engin.
The Chinese University of Hong Kong
khwong@cse.cuhk.edu.hk

Abstract

In computer vision, the Harris corner feature detector is one of the most essential early steps in many useful applications such as 3-D reconstruction. However, if it is implemented in software, the resulting code is probably not able to be executed in real time under low cost mobile processors. This paper proposes an efficient hardware approach that offloads the repetitive feature extraction procedures into logic gates hence the solution is low cost to produce and low power to operate compared to its software counterpart. In this project, the system is built and tested on a popular prototyping FPGA (Field programmable Gate Arrays) platform (Zedboard) with a small FPGA device. The experiments and demos show that the speed and accuracy of the feature detector are good enough for many real world applications.

1. Introduction

The Harris corner detector [1] for detecting sharp corners in images is useful in many computer vision applications, such as pose estimation and 3-D structure from motion (*SfM*). It finds sparse corner points of an image, which can be used for establishing key corner correspondences of two consecutive images of an image sequence. Using these correspondences, for examples, in *SfM*, the pose and structure of the rigid objects in the scene can be calculated. Modern desktop computers can perform corner detection in real-time, however, it is still too complex for a low cost embedded system. In this paper, we propose an efficient hardware implementation to obtain the Harris corners using FPGAs programmed in Verilog. We would like to show that we can design a small and efficient hardware for corner feature extraction which is suitable to be used in mass produced products such as the mobile phone.

1.1 Contributions of this paper

We propose a hardware Harris corner detector using a minimal hardware configuration but retaining all the functionalities of the Harris algorithm. We achieve this using the following approaches. (1) We developed an alternative method for "non-maximum suppression" in the traditional Harris corner detector using less hardware resources with no extra time delay. (2) We developed an efficient mechanism to control the number of features to be obtained. We found that most existing systems do not implement this or using too much hardware to achieve the same goal.

The structure of the paper is as follows. In section 2, we will discuss the theory of the Harris corner detector and the related work. In section 3, we will discuss the implementation details. In section 4, we will talk about the experiment results and in Section 5, we will conclude our work.

2. Theory and related work

2.1 The Harris algorithm

The Harris detector [1] detects a position in the image where both image gradients in two orthogonal axes are high. In appearance it is a corner. It is useful to establish correspondences of 2-D corner points of two images (either stereo images or two consecutive images of a sequence) coming from a 3-D point feature. In sparse field 3D reconstruction (*SfM*), these are the 2-D features that are used in the reconstruct process. The Harris algorithm takes a small window of an image (e.g. 6 × 6 pixels) and determines whether the window contains a corner feature or not. Assume I(x,y) is the image point, we first calculate the gradient matrix M.

$$\mathrm{M} = \begin{bmatrix} \sum_{i,j}^{W} \left(\frac{\partial I(i,j)}{\partial x}\right)^2 & \sum_{i,j}^{W} \left(\frac{\partial I(i,j)}{\partial x}\right)\left(\frac{\partial I(i,j)}{\partial y}\right) \\ \sum_{i,j}^{W} \left(\frac{\partial I(i,j)}{\partial x}\right)\left(\frac{\partial I(i,j)}{\partial y}\right) & \sum_{i,j}^{W} \left(\frac{\partial I(i,j)}{\partial y}\right)^2 \end{bmatrix}$$

where i, j are pixel indexes of a window of range *W*. If the two eigenvalues of M are high, it is a corner feature point. Finding the eigenvalue may be time consuming, so an approximation formula is used instead:

$$\mathrm{R} = \mathrm{Det}(\mathrm{M}) - \mathrm{k} * \mathrm{Trace}(\mathrm{M})^2,$$

where k is a value about 0.04-0.06. If R is bigger than a threshold, the center pixel in the window is a corner feature candidate. By scanning the windows at different positions of the entire image, all feature

candidates can be found. In practice, a local window (say 25 × 25 pixels) may contain too many corner features which cause confusion to some vision algorithms such as *SfM*. To solve this, the system should only allow one corner feature for a local window to be reported, and that can be achieved by a step called "non-maximum suppression" in the original Harris detector algorithm. Moreover, feature candidates are sorted by the Harris score (value of R), the N highest score candidates are the best N corner features. In addition, the amount of features found in every image in a sequence is normally kept to be the same for maintaining better continuity. However, this method is very complex for an FPGA and also causes time delay. Therefore, in this work, an efficient method is proposed and will be discussed in section 3. After this step, the remaining feature candidates are all good sparse feature points. Of course, the number of feature points obtained may vary according to the image content and requirements.

2.2 Related work

There are several existing implementations of the Harris corner feature detector on hardware. However, most of them are either using too much hardware resource or not controlling the amount of feature points reported. In [2], it develops an affine-invariant feature detector that uses too many look-up-tables that can only fit into expensive high-end FPGA devices. Such devices may also consume too much power. Systems in [4, 6, 7, 10, 11] use a fixed threshold to obtain the Harris score only, thus, there is no mechanism to control the total amount of features in an image. In [5], it calculates the eigenvalues (the hardware is too complex) directly instead of using the approximation formula, and a fixed threshold is used. In [8], it uses a sorting module to control the amount of feature points. However, the sorting algorithm employed is iterative therefore it would be difficult to implement it on a pipelined-system (because one must unroll it into a sequence of hardware representation which is costly). The paper reports using 18,539 4-input look-up-tables plus 33 DSPs (Digital Processing Processors) for the entire system which includes the sorting module. It is quite large for a hardware corner detector. In [9], it uses a sorting module plus an adaptive threshold module to control the number of features found. The paper argues that simply adding or multiplying a constant to the threshold may let the amount of features oscillate around the target. They propose a complex mechanism called "inverse cumulative histogram" to overcome the problem. The hardware usage of the entire system is 18,431 4-input look-up-tables plus 81 DSPs. We argue that it is unnecessarily to use such a large device. We suggest it can be solved by simply multiplying a constant to the threshold.

The Harris algorithm also requires to constraint only one corner feature exists in a local area (say 25 × 25 pixels). In [11], it shows that if such a constraint does not exist, all pixels around the actual corner will be considered as corner features and is problematic in many applications. Systems in [3, 4, 5, 6, 10] use a method called "non-maximum suppression" to solve this problem. Only the one with the maximum Harris score in a local window will become a corner feature. However, this mechanism requires a Harris score window that consumes extra hardware and also causes a delay of a couple of scan lines.

3. Implementation

3.1 Overview of the system

The design is implemented on the Avnet Zedboard [12], using an OmniVision OV7670 image sensor [13] and a VGA monitor to show the result. The FPGA part is the Xilinx xc7z020-clg484-1, which is an SoC consisting of FPGA slices and a due-core ARM Cortex-A9 processor (not used in our design).The system block diagram and the hardware setup are shown in Figure 1 and 2. The camera registers are initialized through the SCCB interface. The camera will convert its Bayer pattern data to the 16-bit RGB565 format, and output in two clock cycles though the 8 bit data bus, together with line and frame synchronous signals. The image resolution is 640 × 480 and frame rate is 60 fps. The camera data capture-module will convert the separated data back to the16-bit RGB565 format. The Harris corner feature detector will output the corner feature result and the image for display. Due to the timing mismatch between the camera and VGA signal, a frame buffer must be inserted to separate the two clocked domains (can be removed to save cost if no display is required). Also, one may use external DRAM instead of wasting block RAM resources. Figure 3 shows the block diagram of the Harris corner feature detector in detail. The system is written in HDL (VHDL and Verilog) and is synthesized using the Xilinx Vivado IDE. The camera interface and VGA controller is based on an open-source project [15].

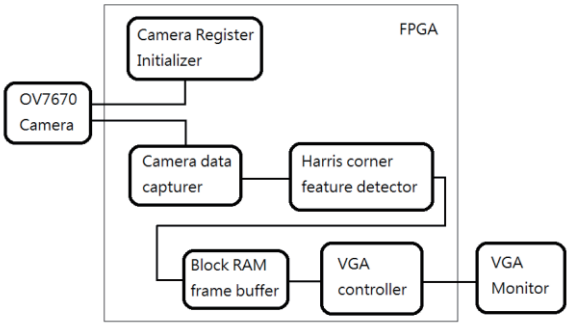

Figure 1: System block diagram

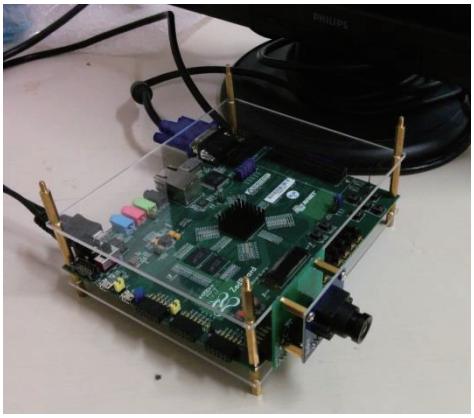

Figure 2: Photo of the hardware setup

3.2.1 Calculating corner feature score

The RGB565 pixel data is converted to 8-bit grey scale using the formula: $0.5 * G + 0.25 * R + 0.25 * B$. (R=Red, G=Green, B=Blue color signal) The multiplication operation is implemented by bit shifting. Then, the pixel goes through a series of D flip-flops and line buffers to create a 6×6 window, where the line buffer is built using block RAM resources. An array of 50 subtractions calculates the image gradient in X and Y directions. The window size is 5×5 pixels. An array of multipliers and adders calculate the elements of the 2×2 matrix M (described in section 2) in parallel. The element M_{12} and M_{21} are actually the same, so only 3 elements are needed for the calculation. The Harris score is obtained by the approximation formula and compared to a threshold. If it is bigger than the threshold, it is a corner feature candidate. In the approximation formula, multiplying 0.04 is estimated by (5/128) because multiplying a 3-bit number by 5 and shifting 7-bit (achieve dividing 128) is easier than dividing it by 25.

3.2.2 Eliminating feature points in local area

If the Harris score is bigger than a threshold, the feature candidate will be considered as a corner feature first. This candidate will also propagate in a feature window (the 1bit×35 window in Figure 3). The feature candidates come later will not be considered as a corner feature because a feature candidate is already found. As mentioned, there should only be one corner feature in a local area. Our proposed method requires a binary window only, where the classical non-maximum suppression method requires a window of Harris scores. Therefore, the hardware resources required of our approach is largely reduced. Also, the classical non-maximum suppression method determines the corner feature after the Harris score propagates through the window which may result in some time delay. In our proposed method, the window is a branch of the pipeline. No extra delay is created. Details of the system are explained below.

3.2.3 Finding best N corner feature candidates

Our idea is that we provide a corner feature counter which will be cleared at the end of every video frames. If the corner features found in the last frame is out of the expected range, the threshold will be multiplied or divided by 2. If the number of corner features is within the range, the threshold will remain the same.

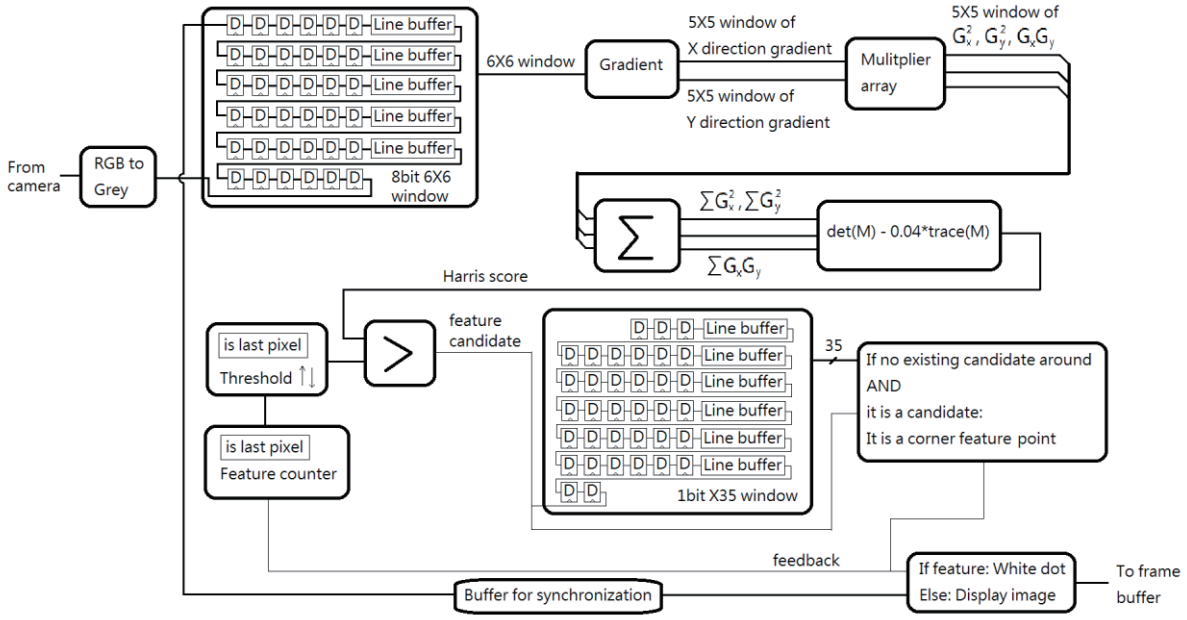

Figure 3: Block diagram of the Harris corner feature detector

The reason we choose multiplication instead of addition and subtraction is that the amount of features obtained is not linearly proportion to the threshold, but is proportional to a function closer to the second order. We also set an upper and lower bound for the threshold. If the image is too dark, it is meaningless to further lower down the threshold to capture bad corner/edge features. If there is no upper bound and the image background is too complex, the threshold will become very high and the corner detector may not output features for a while for some monotonic background, hence the system is not responsive enough. Our approach of using the adaptive threshold retains the functionality of feature sorting, while largely reduces the hardware usage. Figure 4 shows the locations of the corners found by our method through the Zedboard VGA display facility. The original grey scale image is shown and filled with white dots at locations of corner feature points detected.

4. Experimental results

Figure 4: Harris Corner feature detected by our FPGA hardware method

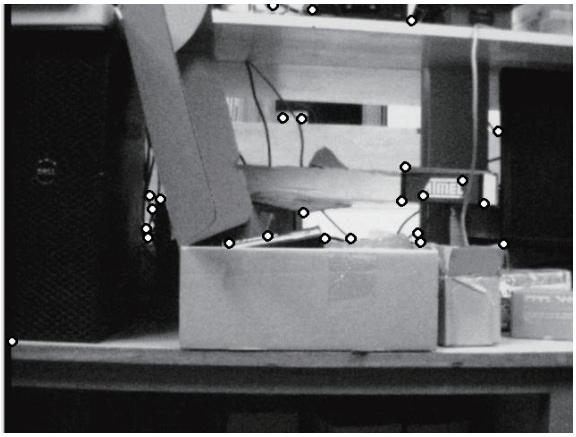

Figure 5: Harris Corners detected by the OpenCV library running on a desktop computer

4.1 Feature detection result

The one pixel white dots in Figure 4 indicate corner features detected by hardware. The white or black circles were added on these pictures by hand to assist viewing. A video frame is captured using the same camera by a desktop computer. Figure 5 shows the Harris feature detection result using the OpenCV library. One may find that the detection result is not the same since the algorithm is modified for hardware implementation. However, the hardware detection result is equally good, except for those features found at the bottom scan line. They are caused by the mixing of the last few rows of the current frame and first few rows of the next frame, as the Harris window will not stop scanning on the image boundary. These bad features can easily be eliminated by software in applications using our hardware.

4.2 Hardware resources usage and speed analysis

Table 1 shows the resource usage when allowing our hardware Harris detector to use the on-board DSP (digital signal processor) resources available in the Zedbaord development system [14]. With the help of the DSP, our approach can operate at 144 frames per second. We also show tests that no DSP is used. Table 2 shows the resource usage when we force the system to stop using the DSP. The maximum frame rate drops to 98 frames per second. We see that with the help of the DSP, the use of LUT (look up table) and registers are reduced. The experiment shows that there are two choices for a designer. Either using a DSP with less LUTs and registers resulting in higher frame rate, or use no DSP but more LUTs and registers resulting in a little lower frame rate.

Table 1. Hardware resource usage (with DSP).

Resources	Full system	Harris detector
6-bit LUT	1400	977
Register	870	662
36kb BRAM	64	0
18kb BRAM	5	5
DSP48E1	110	110

Simulated operating frame rate = 144 frames per second.

Table 2. Hardware resource usage (**without** DSP)

Resources	Full system	Harris detector
6-bit LUT	9849	9485
Register	4335	4131
36kb BRAM	64	0
18kb BRAM	5	5

Simulated operating frame rate = 98 frames per second.

4.3 Comments on the performance

The corner detector performs well on high contract scenes where the background and target objects have distinct grey levels or colors. Applying our system on ordinary indoor environments usually generates good results. However, if the scene is too dark or too bright, our system may fail to capture good features.

4.4 Public domain accessibility

Researchers are welcome to test our code. Source code and demo videos of our project can be downloaded from:

http://www.cse.cuhk.edu.hk/~khwong/www2/public/mva15/mva15.html

5. Conclusion and discussion

We have successfully developed a complete Harris corner feature detector on FPGA using fewer logic gates as compared to previous approaches. As shown in the experiment results, the Harris detector uses 9485 6-bit LUTS and 4131 registers only without using any DSP resources. This low budget of hardware is the result of our improved method that simplifies the maximum suppression procedure in the classical Harris detector. Our hardware experiment showed that using a modern FPGA can enable our system to perform 60 frames per second Harris feature detection. Moreover, our simulation experiment showed that the detection rate can goes up to 144 frames per second. One application of this work is to apply it to real time 3-D reconstruction or pose estimation for mobile devices. Hence, the reduced hardware requirement for feature extraction can result in lowering manufacturing cost and power consumption for these systems.

References

[1] C. Harris, M. Stephens, “A combined corner and edge detector”, Proc. Alvey Vision Conference, 1988, pp. 147–151.

[2] C. Cabani and W. J. MacLean, “A proposed pipelined-architecture for FPGA based affine-invariant feature detectors”, IEEE CVPR 2006, New York, US , June 2006

[3] J. Nikolic, etal.“A Synchronized Visual-Inertial Sensor System with FPGA Pre-Processing for Accurate Real-Time SLAM”, In IEEE ICRA, 2014.

[4] A. Amaricai, C.-E. Gavriliu, O. Boncalo. “An FPGA Sliding Window-Based Architecture Harris Corner Detector”, In International Conference on Field Programmable Logic and Applications (FPL), 2014

[5] A. Benedetti and P. Perona, “Real-time 2-D feature detection on a reconfigurable computer,” in Proc. IEEE Conf. Computer Vision and Pattern Recognition, Santa Barbara, CA, 23-25 June 1998, pp. 586-593.

[6] M. Fatih Aydogdu, M. Fatih Demirci, and Cosku Kasnakoglu , “Pipelining Harris Corner Detection with a Tiny FPGA for a Mobile Robot,” in proc. of ROBIO,Shenzhen, pp. 2177-2184, 12-14 Dec. 2013.

[7] A. Morfopoulos, B. Metz. “Rapid Corner Detection Using FPGAs”, In NASA Tech Briefs, December 2010; 9-10

[8] M. Birem, F. Berry, “Hardware Architecture for Visual Feature Extraction”, Scabot'12 Workshop in IROS 2012

[9] M. Birem, F. Berry. “FPGA-based Real time Extraction of visual features,” IEEE Int. Sym. on Cir. Sys. (ISCAS), 2012

[10] H.J. SONG, Q. LI. A Practical Target Tracking System Design. In Proceedings of the 2009 International Workshop on Information Security and Application (IWISA), 2009

[11] Y. Li, “Fpga implementation for image processing algorithms,” Digital Signal Processing, no. December, 2006. (accessed on 15/12/2015 http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.163.3502&rep=rep1&type=pdf)

[12] Avnet Zedboard, Manufacturer part number: AES-Z7EV-7Z020-G (accessed on 15/12/2015, www.em.avnet.com/en-us/design/drc/Documents/Xilinx/PB-AES-Z7EV-7Z020-G-V4b.pdf)

[13] OmniVision, OV7670 datasheet (accessed on 15/12/2014, http://www.voti.nl/docs/OV7670.pdf)

[14] Xilinx, DS190 Zynq-7000 All Programmable SoC Overview (accessed on 15/12/2014, http://www.xilinx.com/support/documentation/data_sheets/ds190-Zynq-7000-Overview.pdf)

[15] Hamsterworks Wiki, Zedboard OV7670, [online] 2013, (accessed on 15/3/2015, http://hamsterworks.co.nz/mediawiki/index.php/Zedboard_OV7670)