EE2003 Computer Organization Project

Harris Corner Detection on FPGA

Prasanna Bartakke EE19B106, Akshat Joshi EE19B136, and Sarthak Vora EE19B140

1 Introduction

Harris Corner Detector [1] is a corner detector operator that is commonly used in computer vision algorithms to extract corners and infer features of an image. The detector algorithm measures the pixel intensity in all directions by taking the differential of the corner score into account. However, the computationally intensive software implementation cannot be executed efficiently on a low-cost CPU. Our implementation optimises memory usage and aims to parallelise the algorithm[2].

Corners are important features in an image which are invariant to translation, rotation and illumination. Unlike edges, we observe a significant pixel gradient change in all directions in case of corners. Due to this property, shifting a sliding window in any direction leads to a large change in feature computation. This forms the basis of our implementation on hardware and the algorithm finally detects corner points.

2 Implementation

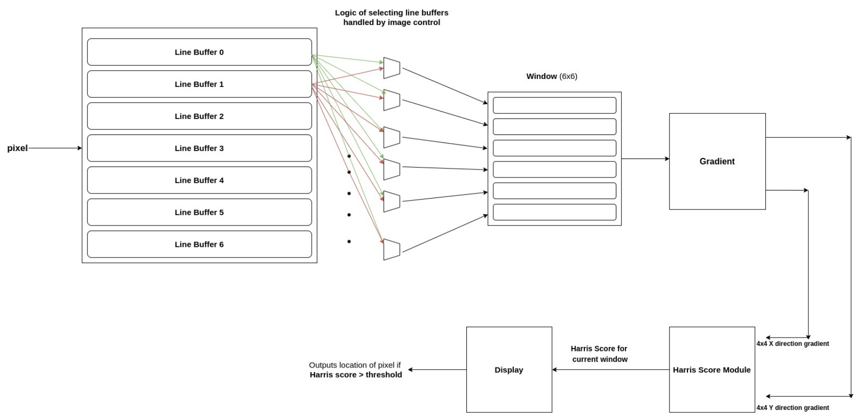


Figure 1: Block diagram of Harris Corner feature detector.

2.1 Line Buffer

The size of the input image is 512 x 512. At each clock cycle, 1 pixel is sent as input to the harris corner detection module. The line buffers allow efficient window generation. Without Line Buffers, the same image row would have to be fetched entirely again and again to generate a new window. This would cause a slow down and waste of precious clock cycles. The line buffers allow to cache the entire image row, preventing the need to fetch the same row from memory again and again.

In the architecture, a 6 x 6 window has to be generated at every clock cycle. Therefore, 6 line buffers of size 512 bytes(for 512 pixels) each can be used, and 6 consecutive entries from the 512 entries can be selected in the line buffer thus allowing to generate a 6 x 6 window from the line buffers. However, using 6 line buffers is also inefficient as every time we move to the next row, we will have to wait for 512 clock cycles to fill the entire line buffer, before a correct window can be generated. Hence, in the architecture 7 line buffers have been used so that by the time the first 6 line buffers are being read out from for generating windows, the next row can be written into parallelly into the 7th line buffer.

Hence, 7 line buffers of 512 bytes each have been used in the architecture having a total size of 3.5 KiB, which can be easily realised using the FPGA block RAMs.

2.2 Gradient

To generate the gradient, the Sobel Filter is used. The sobel filter gives an approximation of the derivative of the image. It consists of two 3x3 linear operators one each for the x and y directions respectively given the gradients in the x and the y direction. For window W the following equations give the gradients G_x and G_y

$$G_x = \begin{bmatrix} +1 & 0 & -1 \\ +2 & 0 & -2 \\ +1 & 0 & -1 \end{bmatrix} \circledast W, \quad G_y = \begin{bmatrix} +1 & +2 & +1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} \circledast W \qquad (1)$$

G_x and G_y will be 4 x 4 when the 3 x 3 sobel filter is convolved on the 6 x 6 window that is generated by the line buffers. The 16 entries of G_x and G_y are computed in parallel using simple combinational logic. The G_x and G_y values are finally written to pipeline registers at the clock edge, from where they can be used by the next stage for harris score calculation.

2.3 Harris Score

The Harris algorithm takes a 6x6 window of the image and computes the gradient as explained above. The computed gradients are then passed on to this module to compute the value of Harris Score. If G_x, G_y represent the gradients in a window W, we calculate the gradient matrix M as follows -

$$M = \begin{bmatrix} \sum_{i,j}^{W}(G_x(i,j))^2 & \sum_{i,j}^{W}(G_x(i,j) * G_y(i,j)) \\ \sum_{i,j}^{W}(G_y(i,j) * G_x(i,j)) & \sum_{i,j}^{W}(G_y(i,j))^2 \end{bmatrix} \tag{2}$$

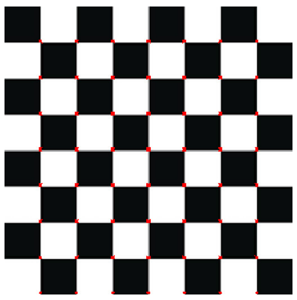
Finally, we compute the Harris score from the above matrix. The equation is shown below -

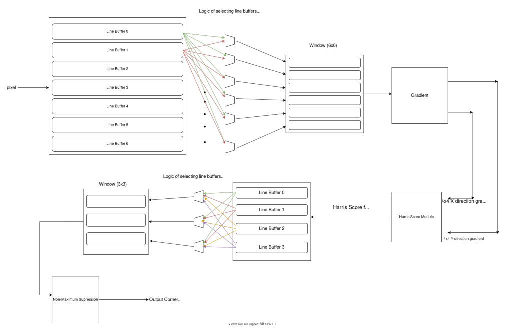
$$R = Determinant(M) - \frac{5}{128} * Trace(M)^2 \tag{3}$$

2.4 Display

Consider the windows which have harris score greater than the threshold. These windows have a corner at their central pixel. The display module prints out these corner pixels to a file, which are processed later by the Python script.

3 Results


(a) Output image, Red points indicate corners.



(b) Block diagram of Harris Corner feature detector with non max supression

Figure 2

For speed comparison, a software implementation of the harris algorithm is written in C and compiled for the PicoRV32. The iverilog simulator is used for both the PicoRV32 as well as the RTL code of the accelarator. It took 19 minutes on average to calculate harris scores for windows corresponding to first 6 rows of the image on the PicoRV simulation. Extrapolating to all 506 set of rows it would take about 7 days on the software implementation. In comparison, the hardware accelerator took only 1 minute on average to process the entire image. Note that the C code is so slow because of the speed limitations of the simulator itself, and also the fact that the PicoRV32 may take multiple clock cycles for one ALU operation to complete, while in the hardware case each window gets processed with multiple ALU operations and passed from one stage to the next in just a single clock cycle. The memory access latency in the hardware is hidden(except for few clock cycles of initial delay to fill the first 6 line buffers) as at every clock cycle not only one pixel is released but also windows generated in the previous clock cycles are being processed upon parallelly in the later stages of the pipeline.

4 Further Improvements

The previous block outputs all the points which have harris score greater than the threshold. There can be many windows with a harris score greater than the threshold in a small region, and their central pixel will be classified as a corner. To avoid too many corner points concentrated together, non-max suppression[3] can be used. In this method, the harris score of each window is compared with the harris score of neighbouring eight windows. If the harris score of the central window is greater than all the neighbouring eight windows, and it is also greater than the threshold, then this window contains the corner. The pixel at the centre of this window is the corner. 4 line buffers are used to generate windows of size 3 x 3. This method avoids storing the harris score of the entire image at once. The code for this part is included in the `non max supression` branch.

Conclusions

In this project, the parallel computing power of FPGA is used to improve the calculation speed of the corner matching algorithm. The resulting corners which are detected match with the expected output.

References

[1] C. Harris and M. Stephens. A combined corner and edge detector. In *Proceedings of the Alvey Vision Conference*, pages 23.1–23.6. Alvety Vision Club, 1988. doi:10.5244/C.2.23.

[2] Tak Lon Chao and Kin Hong Wong. An efficient fpga implementation of the harris corner feature detector. In *2015 14th IAPR International Conference on Machine Vision Applications (MVA)*, pages 89–93, 2015.

[3] Chengda Xu and Yunshan Bai. Implementation of harris corner matching based on fpga. In *Proceedings of the 2017 6th International Conference on Energy and Environmental Protection (ICEEP 2017)*, pages 807–811. Atlantis Press, 2017/06.

Division of work

Prasanna Bartakke EE19B106: Worked on Line Buffer(line_buffer.v), Image Control(image_control.v) and Non-max Supression Verilog Codes, Python Scripts

Akshat Joshi EE19B136: Worked on Sobel Filter(gradient.v), Test bench for Gradient and display (gradient_tb.v (not in main branch), display.v), C code implementation, Python scripts, debugging Image Control(image_control.v), Accelarator Test Bench(harris_tb.v)

Sarthak Vora EE19B140: Worked on Harris Score module (harris_score.v), analysed the stage 2 pipeline of [2] (didn't implement), debugging non-max supression codes (check_corners.v, display_output.py)