Accelerating Feature Detectors For Real-Time Vision-Based Applications

THESIS

Submitted in partial fulfillment of the requirements of

BITS F423T / BITS F421T (Thesis)

by

Bhavan Jasani
(2011B5A3305G)

Under the supervision of

Dr. Lam Siew Kei
Assistant Professor
School Of Computer Science And Engineering
Nanyang Technological University

BITS, PILANI –K K BIRLA GOA CAMPUS

May 2016

CERTIFICATE

This is to certify that the Thesis entitled, *Accelerating Feature Detectors For Real-Time Vision-Based Applications*

is submitted by *Bhavan Jasani* ID No . *2011B5A3305G*

in partial fulfillment of the requirements of **BITS F423T/BITS F421T.** Thesis embodies the work done by him/ her under my supervision.

[illegible]

Signature of the supervisor

Date 6|5|16

Name LAM SIEW FEI
Designation ASSISTANT PROFESSOR.

Abstract

Thesis Title	: Accelerating Feature Detectors For Real-Time Vision-Based Applications
Supervisor	: Asst Prof Lam Siew Kei
Semester	: Second
Year	: 2015-16
Name of student	: Bhavan Jasani
ID No.	: 2011B5A3305G

Abstract:

Feature detection is a fundamental step in many real time applications such as visual SLAM, video tracking and robotic navigation. Harris Corner detector has been demonstrated to be a good feature detector algorithm, it is one of the most accurate corner detector algorithm and works well for noisy images. But it is a computationally complex algorithm involving multiple steps which generate intermediate image data and all these steps are carried out for each and every pixel in the image, and also the size of intermediate data increases for subsequent steps. The computational complexity can be a bottle neck for real time implementation on low power embedded vision systems.

This thesis work is on developing hardware - efficient implementation of Harris Corner Detector on FPGA's which can run in real time with significantly lower logic and memory resources. A complete embedded vision system consisting of camera as the input, FPGA running Harris Corner detector and a VGA display for displaying the detected corners in real time on the input video stream is implemented and demonstrated.

Acknowledgement

Foremost I would like to express gratitude to my thesis supervisor Assistant Professor Lam Siew Kei, for providing me this opportunity to work under him and for his indispensable advice and support on various technical aspects of the project. I would like to express my gratitude to Mr. Gopalakrishna Hegde, Research Associate, Hardware and Embedded Systems lab for his indispensable technical advice throughout the project work.

I would also like to thank my on-campus supervisor Mr. Pravin Mane and to Academic Research Division and BITS-Pilani University in general for keeping the concept of off-campus thesis as a part of academic curriculum.

Table Of Contents

List of Abbreviations

CV = Computer Vision

FPGA = Field Programmable Gate Array

HCD = Harris Corner Detector

NMS = Non-Maximal Suppression

SDRAM = Synchronous Dynamic Random-Access Memory

VGA = Video Graphics Array

List of Figures

Introduction

What are feature detectors? [1,2,3]

Feature detection is a starting task in many computer vision applications like optical flow measurement, camera motion estimation, video tracking, panorama stitching, visual SLAM (simultaneous localization and mapping). Features are the interesting parts in the image which uniquely define themselves as compared to their neighboring region.

For example in case of motion estimation, features are detected between two consecutive frames and then the corresponding features between the adjacent frames are matched (as shown in Figure 1) and then the by finding the shift of features in the two frames one can estimate the motion of the objects in the video or that of the camera which can further be used for stabilizing the video from a shaking motion of camera.

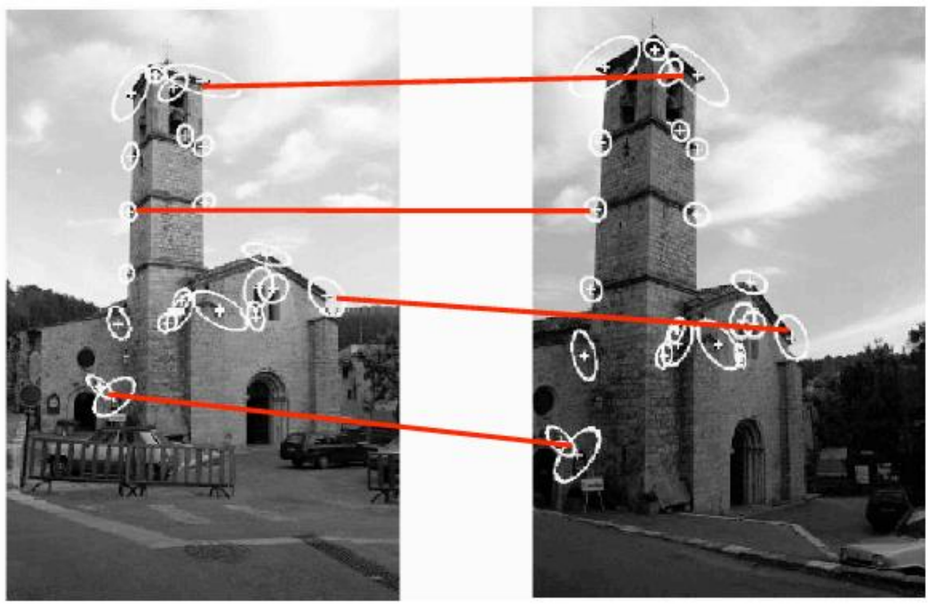

Figure 1: Feature detection and matching example [1]

What are good features? [2]

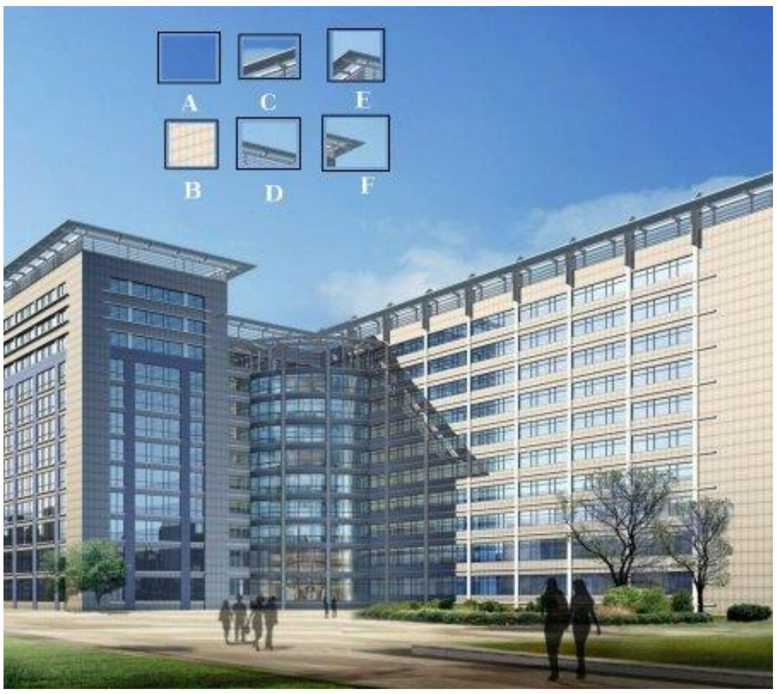

Figure 2: Patch matching example image [2]

Consider the Figure 2 shown above, there are six patches shown at the top which are taken from the image and the aim is to locate these regions exactly in the image.

Consider image patches A and B, they are respectively of sky and building wall, these are flat regions, they have no intensity variation and so it's difficult to find their exact location in the image because many nearby regions in the image look like them.

The image patches C and D are the edges in the building, locating them is much simpler than flat regions, and one can easily find their approximate location, but it's difficult to find the exact location because you can move the patch along the edge of the building and they will match in all those regions. Only normal to the edge the pixels are different. So edges are better features in comparison to flat regions but not good enough for exact matching.

The image patches E and F are the corners of the building, one can easily find their exact location in the image because near corners no matter which direction you move the patches will look different. So corners are better features in the image as compared to edges and flat regions.

A corner is like an intersection of two non-parallel edges. An image patch which contains corner will have lot of pixel intensity variations and so generally it can be uniquely identified in an image and so a corner serves as a good image feature for feature matching between two images of the same scene.

Since feature detection is the starting point of many computer vision algorithms, it can be a deciding factor for the performance of the consecutive steps of the algorithm and so it's important to have a good feature detector.

An important criterion for finding the performance of a feature detector algorithm is the repeatability criteria – which tells the numbers of corresponding features which will be detected in two or more different images of the same scene but involving slight changes in scale, rotation or view point. [4]

There are various different feature detectors with varying complexity and performance. Harris Corner Detector is an important feature detector, it is one of the most accurate corner detection algorithm and it works well for noisy images.

Harris Corner Detector

An early attempt for detecting corners was made by Chris Harris and Mike Stephens mentioned in their 1988 paper titled "**A Combined Corner and Edge Detector**"[5] which is now called Harris Corner Detector.

The basic idea of Harris Corner Detector (HCD) as shown in the Figure 3 is to find small patches of image which generate large variations in pixel intensity when moved around in all directions because these regions are likely to contain corners.

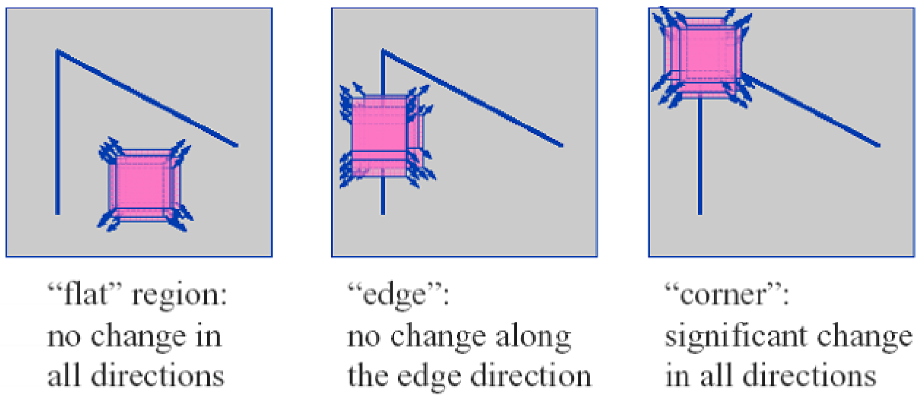

Figure 3: Comparision of corner, edge and flat region [1]

HCD algorithm takes an image pixel along with its neighborhood to finds whether the image pixel is an edge, corner or flat region based on the value of Harris measure which is the output of the algorithm. It relies on the image gradients for detecting corners because a corner will have high variance in its gradient. Harris corner detector is a commonly used feature detector because it is invariant to rotation and illumination in images and to a certain extent to change in scale.

Mathematics of Harris Corner Detector [1,2,6,7]

Consider a grayscale image I, the change in intensity of a small patch in image for the shift (u,v) is given by:

$$E(u,v) = \sum_{x,y} \underbrace{w(x,y)}_{\text{window function}} \, [\underbrace{I(x+u,y+v)}_{\text{shifted intensity}} - \underbrace{I(x,y)}_{\text{intensity}}]^2$$

Where,

- E(u,v) is the difference in pixel intensities between original window and the window displaced by (u,v)
- w(x,y) is the window function, it acts like a mask to ensures only a small patch (of the size of the window) in the image is selected at a time, the commonly used window functions are show below:

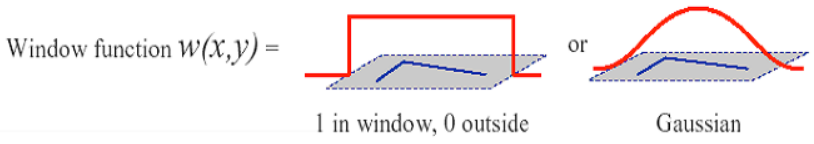

Figure 4: Window function examples [1]

- u is the window's displacement in the x direction
- v is the window's displacement in the y direction
- I(x, y) is the pixel intensity in the image at location (x,y)
- I(x+u, y+v) is the pixel intensity at the window displaced by (u,v)

For nearly constant patches or flat region the term inside the square brackets will be nearly zero and for very distinctive patches like those which contain corners the value will be large. So to find corners we need to look for windows or patches that produce large E(u,v) value, which essentially means to maximize the terms inside the square brackets:

$$\text{Maximize} \quad \sum_{x,y} [I(x+u,y+v) - I(x,y)]^2$$

Using first order Taylor series approximation we get:

$$E(u,v) \approx \sum_{x,y} [I(x,y) + uI_x + vI_y - I(x,y)]^2$$

This can be simplified as:

$$E(u,v) \approx \sum_{x,y} u^2 I_x^2 + 2uvI_x I_y + v^2 I_y^2$$

This equation can be written in matrix form as:

$$E(u,v) \approx [u \quad v] \left(\sum_{x,y} w(x,y) \begin{bmatrix} I_x^2 & I_x I_y \\ I_x I_y & I_y^2 \end{bmatrix} \right) \begin{bmatrix} u \\ v \end{bmatrix}$$

$$E(u,v) \approx [u \quad v] \, M \begin{bmatrix} u \\ v \end{bmatrix}$$

Where the matrix M is defined as:

$$M = \sum_{x,y} w(x,y) \begin{bmatrix} I_x^2 & I_x I_y \\ I_x I_y & I_y^2 \end{bmatrix}$$

Matrix M above consists of the product of image derivatives in x and y directions which we got because of Taylor's series expansion. The matrix elements are weighted by the weights defined in window function and summed over the region defined by the window function. For Harris Corner Detector a Gaussian window is used, this smoothens out the noise in image.

The eigenvalues of the matrix M can determine the suitability of a window. A score, R which is known as Harris measure is calculated for each window, this score determines whether the window contains a corner, edge or a flat region:

Measure of corner response:

$$R = det\, M - k\,(trace\, M)^2$$

Where,

- det M = $\lambda_1\lambda_2$
- trace M = $\lambda_1 + \lambda_2$
- λ_1 andλ_2 are the eigenvalue of matrix M
- k = empirical constant in the range 0.04 – 0.06

Properties of Harris measure (R) as shown in Figure 5 are:

- It depends only on the eigenvalues of matrix M
- It is positive and large in magnitude for a corner, a threshold is defined for classifying a pixel as a corner
- It is negative and of large magnitude for an edge
- Magnitude of R is small for a flat region

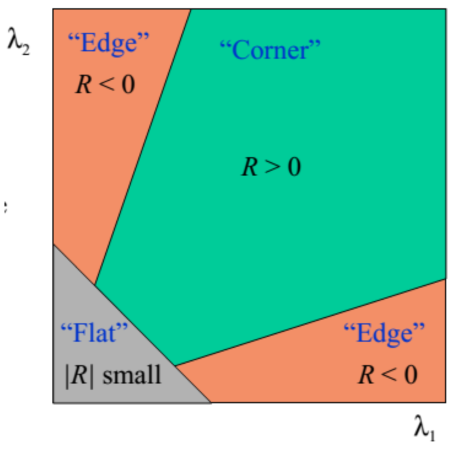

Figure 5: Classification of pixel based on Harris Measure [7]

HCD algorithm summary [1]

1) Compute x and y derivatives of the image at every pixel using Sobel operators

$$I_x = G_\sigma^x * I \quad I_y = G_\sigma^y * I$$

2) Compute product of derivatives at every image pixel

$$I_{x2} = I_x.I_x \quad I_{y2} = I_y.I_y \quad I_{xy} = I_x.I_y$$

3) Apply Gaussian smoothing to the products of derivatives at every pixel

$$S_{x2} = G_{\sigma'} * I_{x2} \quad S_{y2} = G_{\sigma'} * I_{y2} \quad S_{xy} = G_{\sigma'} * I_{xy}$$

4) Define the Harris matrix at every image pixel (x,y)

$$H(x,y) = \begin{bmatrix} S_{x2}(x,y) & S_{xy}(x,y) \\ S_{xy}(x,y) & S_{y2}(x,y) \end{bmatrix}$$

5) Compute the Harris measure (R) at every image pixel using the below formula

$$R = Det(H) - k(Trace(H))^2$$

Where, k is an empirical constant in the range (0.04 - 0.06)

6) Apply threshold on the values of R, points greater than the threshold are potential corners

7) Apply non maximal suppression around detected potential corners to remove adjacent falsely detected corners

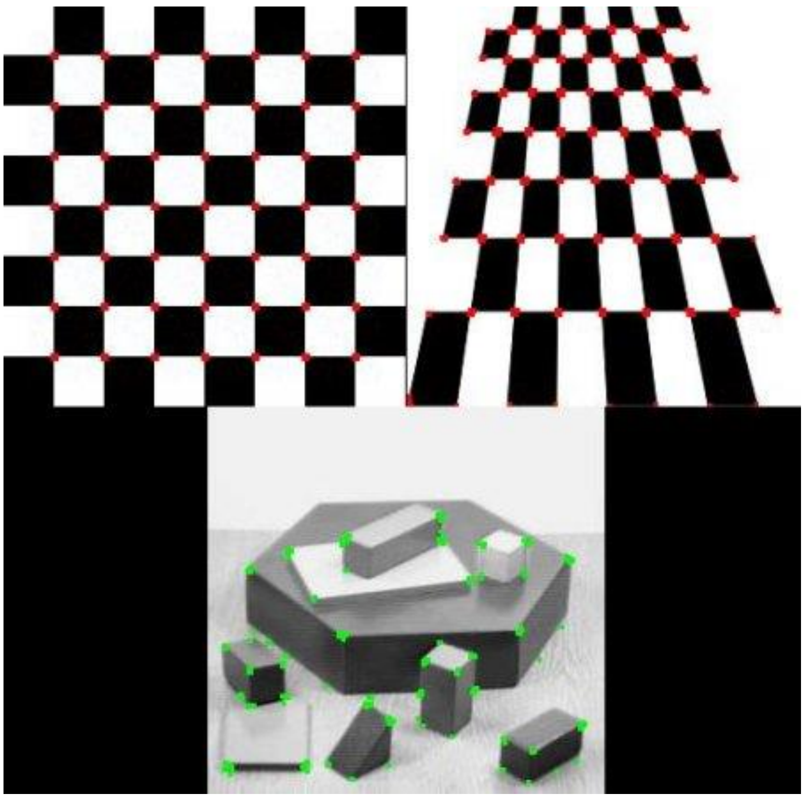

Figure 6: Example of corners detected using HCD [1]

Hardware implementation of HCD

System overview

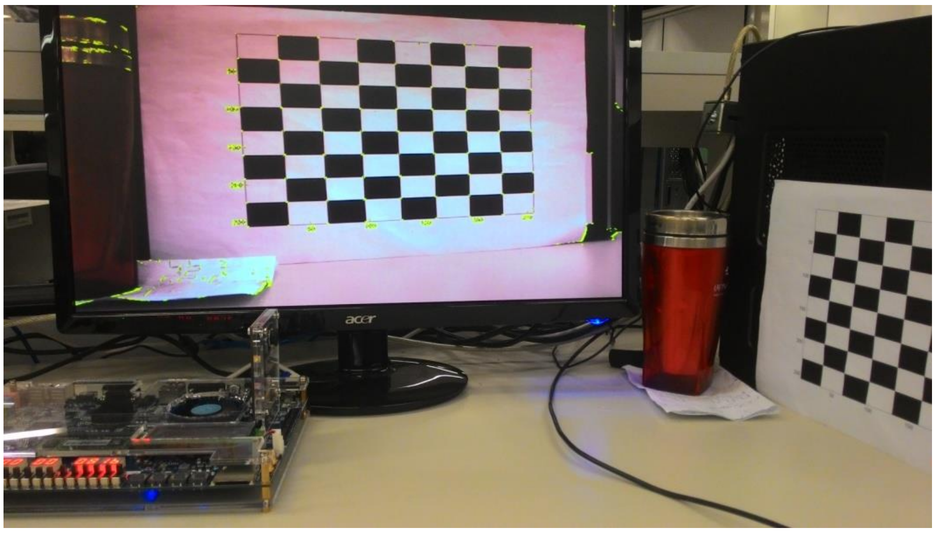

Figure 7: Overall system consisting of camera, FPGA development kit and display

Figure 7 shows the overall system design consisting of the camera as the input device, VGA display as the output device and FPGA as the interface between these and running Harris Corner Detector.

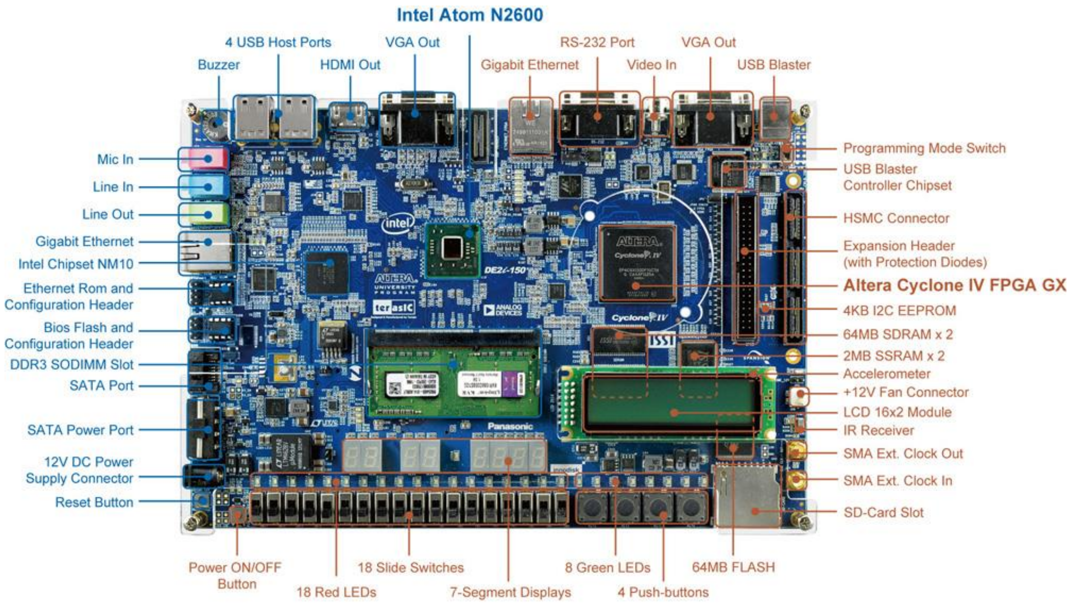

Figure 8: Terasic DE2i-150 FPGA development board [8]

The FPGA development kit used (shown in Figure 8) for the implementation is the Terasic DE2i-150 which consists of Intel Atom Dual Core Processor N2600 and Altera Cyclone IV EP4CGX150DF31 FPGA both of which are linked together via two high speed PCIe lanes. The development kit provides easy support for connecting and using camera, SDRAM and VGA display which are the essential components of this system. [8]

Figure 9: Terasic D5M digital camera [9]

The camera is used is the Terasic D5M shown in Figure 9 which is a 5 Mega Pixel digital camera compatible with various Terasic FPGA development kits. It provides user controllable resolution of up to 2592 x 1944 and up to 150 frames per second. [9]

The camera is connected to the GPIO interface of the Cyclone IV FPGA via the 40-pin expansion header available on the development board. The output of camera is in RGB Bayer Pattern format and so has to be converted to RGB format by FPGA for further processing. After converting the raw camera data into RGB we get 30 bits RGB pixel data (10,10,10) at every clock cycle of the camera. For the implementation purpose the camera was set to output 640 x 480 resolution video. [10]

Output from camera is first decoded by FPGA, then a complete image frame is stored on the SDRAM available on the development board. Data from SDRAM is then feed to the Harris Corner Detector which in turn detects the corners and the video from camera with detected corners marked in yellow are sent to computer monitor connected via the VGA connector which outputs in real time the video with detected corners.

Verilog modules for interfacing the Terasic D5M camera, onboard SDRAM and the VGA display to the FPGA development board were provided by Terasic [8,9] although some of them had to be slightly modified because they were meant for older Terasic FPGA development boards, resources available on [10,11] were helpful for that.

Implementation details [10,11]

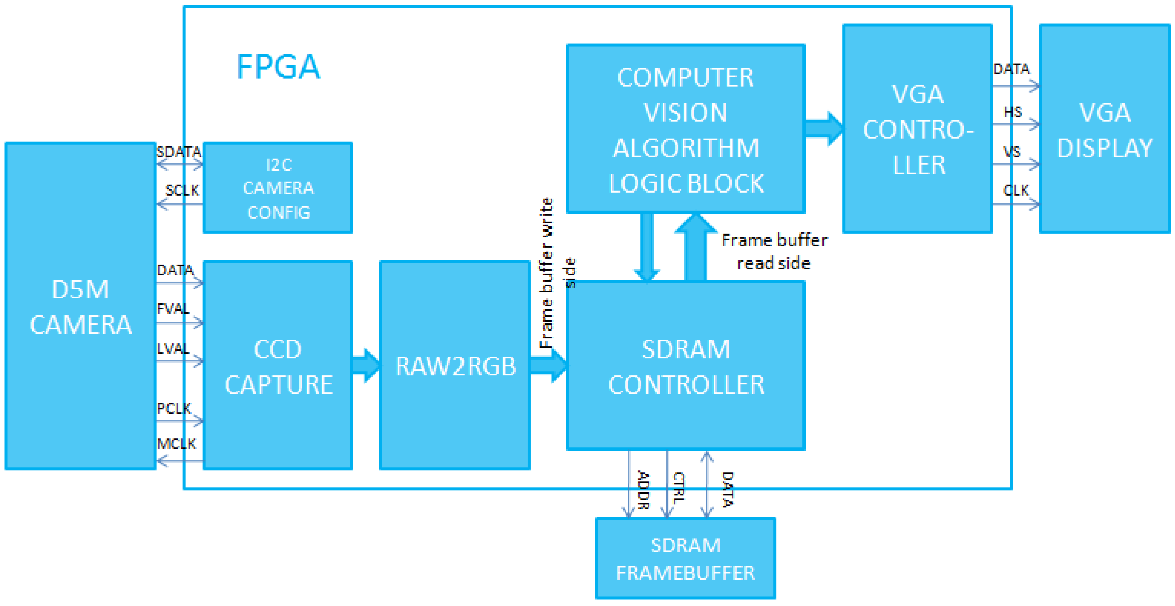

Figure 10: Overview of the implemented system

Figure 10 shows the block diagram of the overall system from camera to VGA display with an overview of FPGA blocks. Following blocks are implemented in FPGA:

1) CCD CAPTURE

   This block acts as an interface between the camera and the FPGA. The 12 bit parallel data coming from camera is captured by the CCD CAPTURE block and it calculates the pixel coordinates and the frame count.

2) I2C CAMERA CONFIG

A clock (MCLK as shown in figure 10) has to be feed to camera which is generated by the PLL present in FPGA. Camera in turn uses an internal adjustable PLL to generate a “camera clock” shown as PCLK in figure 10 whose input is the clock from FPGA. The camera outputs the 12 bit pixel data every rising edge of the “camera clock”. The scaling factor of the internal PLL and other properties of camera like image resolution and frames per second can be controlled by FPGA through this block using I2C interface with camera.

3) RAW2RGB

Since the data coming from camera is in the RGB Bayer Pattern format it has to be converted to RGB format for further processing. This is implemented by this block. It takes as input the 12 bit raw data coming from CCD CAPTURE block and converts it into 30 bit RGB (10-10-10) data.

4) SDRAM CONTROLLER

The camera clock operates at around 77.5 MHz for 640 x 480 image resolution whereas the VGA display by it defined standard, operates at 25 MHz for displaying 640 x 480 image and so it’s essentially to use frame buffer for storing the image frame. Although the data is written at much higher frequency as compared to the display & processing frequency no perceivable information is lost since in a video few consecutive frames are almost the same.

The data coming from camera is stored as a complete frame in SDRAM available on board before using it for further processing. In the available SDRAM one location can store 16 bits but since we have 30 bits of data for every pixel, two locations are used for writing data simultaneously, same is true for reading data from SDRAM. The interface of SDRAM with FPGA is implemented by the SDRAM CONTROLLER block which supports simultaneously dual port read and write operations.

5) COMPUTER VISION ALGORITHM LOGIC BLOCK

Figure 11 shows the custom logic for implementing the feature detector. It consists of two parallel paths. The first one is the RGB2GRAYSCALE + HARRIS CORNER DETECTOR and another one is the DELAY PIPELINE. All the subsequent blocks from SDRAM controller's read side run at VGA display's clock.

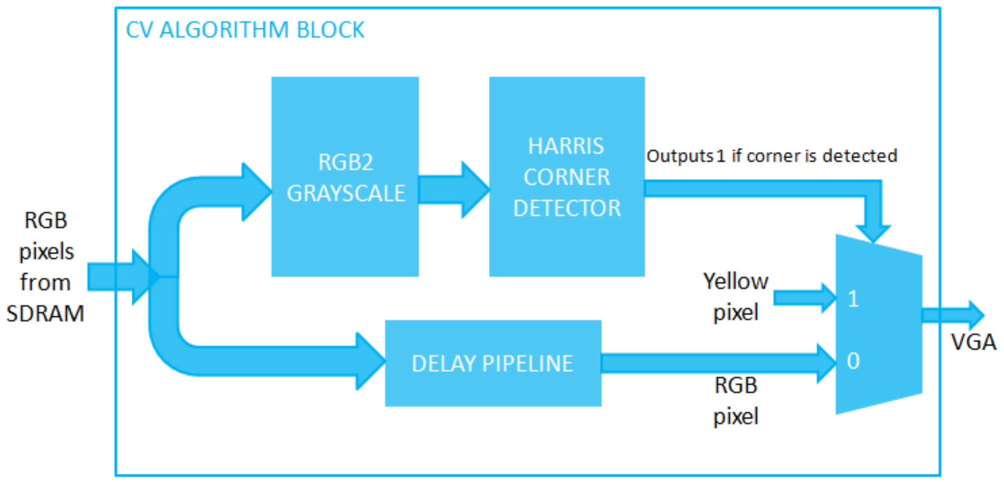

Figure 11: CV Algorithm logic block

5.1) RGB2GRAYSCALE

For detecting corners we don't require any information about individual RGB colors, grayscale information is sufficient so the Harris Corner Block takes grayscale pixel values as the input rather than RGB pixels, this avoids processing three separate color channels which would have required more logic and memory resources.

The block takes RGB pixel every clock cycle and converts it into grayscale using the below equation.

$I = 0.25 \times R + 0.50 \times G + 0.25 \times B$

More weight has been given to green pixel because human eyes are more sensitive to green color and so actually the camera's CCD sensor contains twice as many green filters as compared to red and blue filters.

5.2) HARRIS CORNER DETECTOR

This block implements a pipelined version of Harris Corner Detector, it takes grayscale pixels every clock cycle as input and generates a 1 bit signal which indicates whether the current pixel is corner or not. The details about this block are explained in the next section.

5.3) DELAY PIPELINE

This block simply takes RGB pixel every clock cycle and delays it by number of clock cycles equal to the pipeline stages in HCD block. Since the HCD block is pipelined it is essential that the data coming from HCD block which tells whether the input pixel is a corner or not and the corresponding RGB pixel read from SDRAM which has to be given to VGA controller are the synchronized.

6) VGA CONTROLLER

The VGA controller implemented inside FPGA takes the data which has to be displayed on VGA display and generates signals which are required for interfacing the VGA display with the FPGA via the VGA digital to analog converter IC present on the development board.

Details of Harris Corner Detector Block

Various different hardware architectures have been implemented like [12, 13, 14] for Harris Corner detectors, generally they can classified to be either based on frame buffers approach or the local neighborhood approach.

As mentioned previously the Harris Corner Detector consists of 5 sequential steps - gradient computation, product of gradients calculation, Gaussian smoothing, Harris measure calculation and NMS. In the frame buffer based approach as implemented in [12] the individual steps are performed over complete image and the resultant intermediate images are stored in frame buffer(s). Whereas in the local neighborhood based approach as implemented in [13] the individual steps are carried out over only a small neighborhood of pixels which are stored in line buffers. In this work I have implemented a pipelined architecture based on the local neighborhood approach, as this would require fewer FGPA resources.

The thesis work first started with the implementation of the hardware architecture of Harris Corner Detector in Matlab, this allowed easily to make changes in the architecture to improve it in terms of computation and memory requirements and to find optimal parameters like threshold value for corners. Also it's easier to work with images and videos in Matlab. The implementation has been tested on the image data set available on [4].

Then later on the architecture was implemented in Verilog and tested in Modelsim using images and finally it was implemented on FPGA to make a complete real time system with a camera and a display which was the final aim. Since the work carried out is yet to be published the exact details of the architecture are not mentioned in the description below.

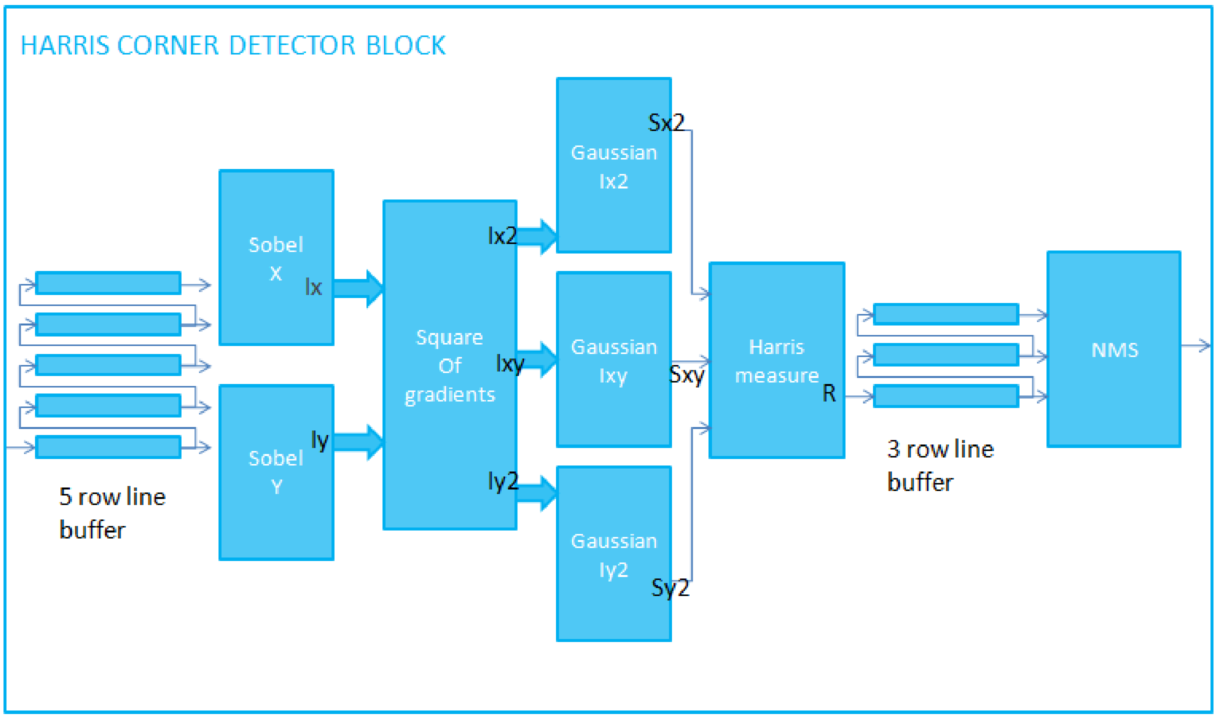

Figure 12: Harris Corner Detector block, the input to this block are the sequential grayscale image pixels and the output is a 1bit value which indicates whether the input pixel is a corner or not.

Figure 12 shows the detailed block diagram of the pipelined Harris Corner Detector which is the main part of CV algorithm logic block inside FPGA of the system. It consists of the following blocks

1) 5 row line buffer: Sobel operators require neighboring pixels to find image gradient and so this block stores 5 consecutive grayscale image rows for which Sobel operator has to be applied. It takes as input a single grayscale 8 bit pixel every VGA clock cycle and outputs the last pixels of every row to the next block. In the implementation I am finding gradients for 3 pixels in a column simultaneously as shown in Figure 13 which requires 5 x 3 pixel neighborhood which in turn requires 5 rows of image data at a time.

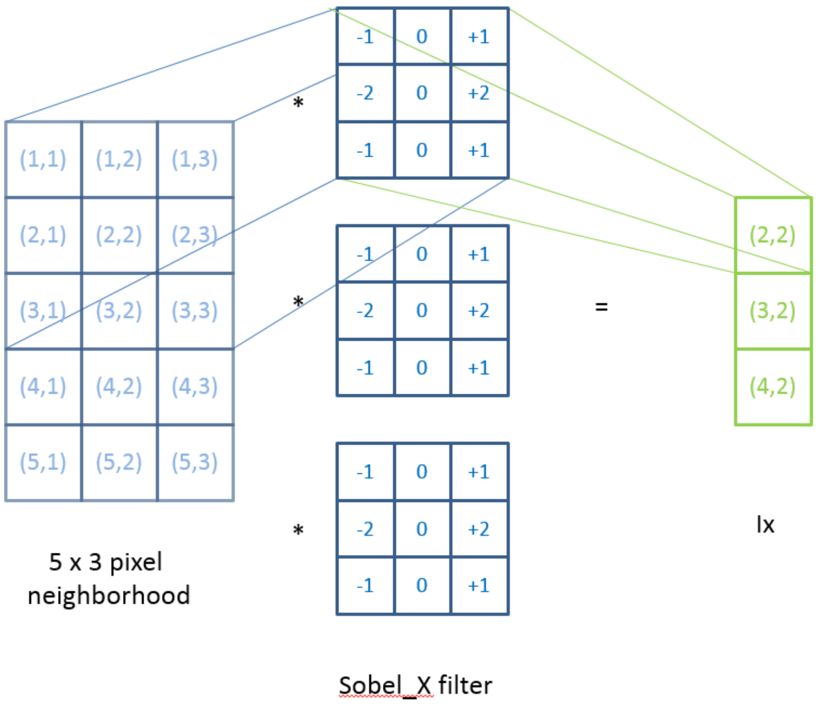

Figure 13: Convolution of Sobel X filter with 5 x 3 pixel neighborhood

2) Sobel operators: Sobel operators shown in Figure 14 are used for finding gradients in x and y directions. For a particular image pixel a 3 x 3 neighborhood of image pixels are taken from line buffer and are convolved with these filters to find gradients. Sobel operators are discrete version of derivative. The Sobel X filter takes difference of the 3 pixels in top row to that of 3 pixels in the bottom row to find the gradient for middle pixel of a 3 x 3 pixel region as shown in Figure 13. It acts as a horizontal edge detector. Similar is the case for Sobel Y filter.

$$\begin{bmatrix} -1 & 0 & +1 \\ -2 & 0 & +2 \\ -1 & 0 & +1 \end{bmatrix}$$

Sobel X filter

$$\begin{bmatrix} +1 & +2 & +1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}$$

Sobel Y filter

Figure 14: Sobel filters

3) Square of gradients: It takes as input the image derivatives Ix and Iy computed from the previous blocks and computes

   3.1) Ix2 which is pixel wise square of x-gradient

   3.2) Iy2 which is pixel wise square of y-gradient

   3.3) Ixy which is the pixel wise product of x and y gradients

4) Gaussian Smoothing: Three Gaussian smoothing blocks operate in parallel to apply 3 x 3 Gaussian filter shown below in Figure 15 to the square of gradients from previous block, to generate the elements of matrix M for every image pixel.

$$\frac{1}{16}\begin{bmatrix} 1 & 4 & 1 \\ 4 & 8 & 4 \\ 1 & 4 & 1 \end{bmatrix}$$

3 x 3 Gaussian kernel

$$M = \begin{bmatrix} Sx2 & Sxy \\ Sxy & Sy2 \end{bmatrix}$$

Figure 15: 3 x 3 Gaussian kernel

5) Harris measure: It takes input as the Gaussian smoothened product of image derivatives and computes it's Harris measure and compares the Harris measure against a threshold to detect whether the pixel is a corner or not.

$$R = (Sx2 * Sy2 - Sxy * Sxy) - k * (Sx2 * Sx2 + Sy2 * Sy2)^2$$

6) 3 row line buffer: Next block after this which is the non-maximal suppression (NMS) block requires a 3 x 3 neighboring region whereas the block Harris measure block outputs only one pixel of data every clock cycle and so a 3 row line buffer is used which stores the Harris measures of 3 consecutive rows of pixels.

7) Non maximal suppression (NMS) :This blocks takes the 3 x 3 region of Harris measure values if the central pixel is detected to be a corner in the previous block then it compares it with the rest 8 values, and if the central element has highest Harris measure then it considers the central pixel as a corner either wise not. Essential it removes the nearby falsely detected corners in proximity to the actual corner.

Conclusion

Feature detectors are an essential starting step in many computer vision algorithms and so implementation of good feature detectors is essential. Harris Corner Detector is an accurate feature detector but it involves multiple intermediate steps which can become a bottle neck for real time implementation on a low power embedded system.

The worked carried out in this thesis demonstrates the acceleration of Harris Corner Detector on a FPGA, the implementation use local neighborhood for corner detection thereby avoiding storage of intermediate image data in memory. HCD architecture has been optimized to use lesser logic and memory resources thereby enabling a low power and real time embedded vision system. The HCD architecture developed has been implemented on a FPGA which takes input from connected camera and display the corner detected video in real time on a VGA display. The bare implementation uses 3,928 / 149,760 which is 3% of the available logic elements, 248,514 / 6,635,520 which is 4% of the available memory bits and uses 46/760 which is 6% of available 9-bit embedded multipliers on the FPGA.

References

[1] Lecture 06: Harris Corner Detector slides, Robert Collins, CSE486, Penn State

[2] OpenCV 3.0.0-dev documentation -> OpenCV- Python Tutorials -> Feature Detection and Description - http://opencv-python tutroals.readthedocs.io/en/latest/py_tutorials/py_feature2d/py_table_of_contents_feature2d/py_tabl_of_contents_feature2d.html

[3] Wikipedia pages on – 1) feature detection, 2) interest point detection, 3) corner detection

[4] Affine covariant features webpage, Robotics @ University of Oxford – http://www.robots.ox.ac.uk/~vgg/research/affine/

[5] C. Harris, M. Stevens, "A combined corner and edge detector", Proceedings of the 4th Alvey Vision Conference, 1988

[6] Tutorials on AI shack - http://aishack.in/tutorials/

[7] “Matching with Invariant Features”, Darya Frolova, Denis Simakov, The Weizmann Institute of Science

[8] Terasic DE2i-150 FGPA development kit user manual

[9] Terasic D5M digital camera hardware specification guide

[10] Purdue University DE2i-150 support homepage - https://sites.google.com/site/de2i150atpurdue/real-time-image-processing

[11] Real-time Cartoonifier, Cornel University ECE 5760, project page - https://people.ece.cornell.edu/land/courses/ece5760/FinalProjects/f2010/kaf42_jay29_teg25/teg25_jay29_kaf42/

[12] M. F. Aydogdu, M. F. Demirci, C. Kasnakoglu "Pipelining Harris corner detection with a tiny FPGA for a mobile robot", Proc. IEEE International Conference on Robotics and Biomimetics, ROBIO, 2013

[13] A. Amaricai, C.-E. Gavriliu and O. Boncalo, "An FPGA sliding window-based architecture Harris Corner Detector", Field Programmable Logic and Applications (FPL), 2014 24th International Conference, pp. 1-4

[14] P. Y. Hsiao, C.L. Lu, L.C. Fu "Multilayered Image Processing for Multiscale Harris Corner Detection in Digital Realization", IEEE Trans. On Industrial Electronics, Vol. 57, Issue 5, 2010