Technical University of Crete

Electronic and Computer Engineering Department

Microprocessor & Hardware Laboratory (MHL)

IMPLEMENTATION OF THE OPENSURF ALGORITHM IN FIELD PROGRAMMABLE GATE ARRAYS

Diploma's Thesis

Dimitris Bouris

Committee: Assist. Professor I. Papaefstathiou (Thesis Advisor)
Professor A. Dollas
Professor M.Zervakis

Chania, Crete 2010

Abstract

Feature detectors are schemes that locate and describe points or regions of ‘interest’ in an image. Today there are numerous machine vision applications needing efficient feature detectors that can work on Real-time; moreover, since this detection is one of the most time consuming tasks in several vision devices, the speed of the feature detection schemes severally affects the effectiveness of the complete systems. As a result, feature detectors are increasingly being implemented in state-of-the-art FPGAs. This thesis describes an FPGA-based implementation of the SURF (Speeded-Up Robust Features) detector introduced by Bay, Ess, Tuytelaars and Van Gool; this algorithm is considered to be the most efficient feature detector algorithm available.

This implementation can support processing of standard video (640 x 480 pixels) at up to 56 frames per second while it outperforms a state-of-the-art dual-core Intel CPU by at least 8 times. Moreover, the proposed system, which is clocked at 200MHz and consumes less than 20W, supports constantly a frame rate only 20% lower than the peak rate of a high-end GPU executing the same basic algorithm; the specified GPU consists of 128 floating point CPUs, clocked at 1.35GHz and consumes more than 200W.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Human vision is a complex combination of physical, psychological and neurological processes that allows us to interact with our environment. We use vision effortlessly to detect, identify and track objects, to navigate and to create a conceptual map of our surroundings.

The goal of computer vision is to design computer systems that are capable of performing these tasks both accurately and efficiently using a minimal amount of time and resources. Current computer vision systems are far from matching the flexibility of biological visual systems. However they have been successfully applied in areas such as manufacturing, medicine and robotics to perform object recognition, scene reconstruction and motion tracking among many others.

Part of research in computer vision focuses on developing algorithms that can locate and describe points or regions of 'interests' (features) in an image. The idea is that if a structure in an image (such as part of an object or a texture) can be described by a limited set of features, and this set is scale and rotation invariant, then the features can be used to identify and match the same structure in different images.

Once a set of features is detected in an image, the characteristics of the image around the features can be encoded description vectors. These description vectors, for example, can be computed independently for two images. Description vectors can also be used to index a database of object descriptors to perform object recognition. Features and descriptors are usually computed in the preliminary stages of systems for object recognition, object tracking, motion analysis and video indexing.

What constitutes a feature depends to a large extent on what is the intended use for the information that is extracted from the images. For example, different application may have different requirements for the robustness of the features to changes in viewing conditions. In general, feature detectors try to achieve invariance to changes in illumination, scale, location, 2D rotation and even affine or perspective transformations. In addition, features should be distinctive so the features corresponding to different structures can be easily distinguished.

As the complexity of vision task increases, the processing time and computer resources required to perform it, are increased respectively. In particular, embedded vision systems, like the ones used in autonomous robot navigation need to process large amounts of data in real-time. For this reason, many vision algorithms have been ported to integrated circuits such as Application-Specific Integrated Circuits (ASICs)

and Field-Programmable Gate Arrays (FPGAs) [21]. Integrated circuits often referred to simply as hardware, can increase the speed of vision algorithms by one to three orders of magnitude when compared to the speed achieved in general-purpose processors.

Hardware implementation of vision algorithms can achieve this increase in speed because of the inherent parallelism of many of the operations involved in processing an image. For example, convolution of an image with a two-dimensional filter is implemented in a general-purpose processor as a series of multiplication and additions where only one coefficient of the filter is processes at a time. Thus, several cycles are required to produce a result for a single image pixel. On the other hand, in a hardware implementation all coefficients in the filter could potentially be processed at the same time, as long as there are enough resources to implement the necessary multipliers and adders. Moreover, since many of the processes in a vision system are relatively independent of each other, the system can be pipelined so that several processes are executed at the same time, which greatly increases the throughput of data.

The amount of resources available to implement operations in hardware is often limited by factors such as cost, weight and power consumption. Thus, it is important to design an implementation of the algorithm that is carefully crafted to balance the need of accuracy and numerical precision with an efficient use of the hardware resources. For this purpose, many hardware designs use a fixed-point representation instead of the floating point representation used in general-purpose processors.

The extensive use of feature detectors as preliminary stages of vision applications, coupled with the capacity and speed of current FPGAs, have led to the development of smart camera systems with integrated feature detection stages. These systems can preprocess incoming video from a camera and provide real-time information to subsequent processing stages.

This thesis describes an FPGA-based implementation of a scale and rotation invariant detector and one part of feature descriptor introduced by Herbert Bay, Andreas Ess, Tinne Tuytelaars, Luc Van Gool [1], referred to in the literature as SURF(Speeded-Up Robust Features). This thesis is organized as follows: Chapter 2 introduces the feature detector, field-programmable gate arrays and gives a brief review of the design methodology. Chapter 3 provides a detailed description of the design of the detector, including a discussion on the fixed-point arithmetic. Chapter 4 presents the results obtained from the completed system and Chapter 5 presents the conclusions of the project and discusses possible directions for future work.

Chapter 2

Background

This chapter introduces some of the concepts applied throughout this thesis. Section 2.1 provides a brief overview of the methods developed for feature detection encountered in the literature, with focus on those based on image intensity that achieve invariance to scale, viewing conditions and other transformations. Section 2.2 describes SURF feature detector and descriptor. Section 2.3 gives an overview of the algorithm in steps. Section 2.4 introduces Field-Programmable Gate Arrays (FPGAs). Section 2.5 discusses some of the existing FPGA-based, GPU-based vision systems. Section 2.6 presents the design methodology.

2.1 Feature detectors

The literature on feature detectors is extensive. The detection algorithms can be based on image characteristics such as contours, intensity and phase (refers to the phase response of a complex-value filter applied to an image), as well as on parametric models (efficient models associated to features such as edges, corners, vertices by searching the parameters of the model that best approximate the observed grey level image intensities). A complete description of the existing methods and the evaluation criterion of different interest point detectors are explained in [43]. Despite their differences, all methods attempt to achieve invariance to changes in viewpoint to facilitate the description and matching of objects across images.

Most intensity-based feature detectors can be traced back to the work of Moravec [3]. His method observes the average changes in image intensity over a local window, result from shifting the window over the image. The features are the points at which the image intensity changes in more than one direction. Lowe [4, 5] proposed SIFT (Scale Invariant Feature Transform) which is the most recent feature detector.

In general, the size of objects can vary significantly between images and thus algorithms that search for features at a single scale miss important information present at other scales. Many of the methods that achieve invariance to scale changes analyze images at multiple scales using image pyramids [6]. Image pyramids are hierarchical structures that represent an image at different resolutions. The different ‘levels’ of the pyramid are formed by convolving the original image with filters at different scales.

Since Gaussian filters attenuate high-frequency components, each level can be sub-sampled to reduce the amount of effort required to compute the next level. The number of levels and the type of filters used depend on the application. Often different levels of the pyramid are combined to form different representations, as in the case of difference-of-Gaussians pyramid, where each level is formed by subtracting adjacent levels of Gaussian pyramid.

There are numerous approaches that explore ways to detect features that are invariant to scale changes. Crowley and Parker [7] presented a multi-scale representation of shapes in an image, constructed by detecting peaks and ridges in a pyramid of difference-of-Gaussians. Their algorithm forms a multi-scale tree that describes shape by linking adjacent peaks and ridges across different levels of the pyramid. Linderberg has done extensive work on the optimal scales for feature selection [8, 9, 10, 11]. He proposed a method to detect blob-like structures over scale space [10] and later studied how feature detection with automatic scale selection can be formulated for various kinds of features based on scale spaces formed by normalized differential operators [11].

Harris and Stephens [12] improved Moravec's idea to make the detection more robust to noise and nearby changes. Their method used a second moment matrix to describe the image gradient in Gaussian window centered at a pixel position. This detector is robust to rotation and translation, however its performance degrades as the change in scale becomes more significant because image derivatives are sensitive to changes in the size of local structure. Mikolajczyk and Schmid [13] extended Harri's approach to achieve invariance to scale. Their methods use a scale-normalized second moment matrix and select points at which the matrix presents two large eigenvalues. The characteristic scale for each point is the scale at which the Laplacian achieves an extremum. Shokoufandeh, Marsic and Dickinson [14] developed a scale-invariant method that captures the salient regions of an object using a wavelet transform at different scales.

SIFT by Lowe [5] use four major stages of computation to generate the set of image features. Scale-space extrema detection stage searches over all scales and image locations. It is implemented efficiently by using a difference of Gaussian function to identify potential interest points that are invariant to scale and rotation. Key-point localization stage fits each candidate location by a 3D quadratic function to determine the location and scale and then enforcing the condition that the Hessian matrix of the image intensity presents two large eigenvalues. Orientation assignment stage assigns an orientation at each feature based on the dominant directions of the local gradients. Key-point descriptor stage measure local image gradients at the selected scale in the region around each key-point and these gradients transformed into a representation that allows for significant levels of local shape distortion and change in illumination.

2.2 SURF detector

The system described in this thesis implements the Surf interest point detection and orientation assignment at each detected interest point [1]. The Surf detector consists of two basic distinct parts. Fist part called interest point detection and second known as interest point description. Both parts are based on the use of integral images as made popular by Viola and Jones [15]. This section presents the concept of integral images, the interest point detection and description from a theoretical standpoint.

2.2.1 Integral Images

Much of the performance increase in SURF can be attributed to the use of an intermediate image representation known as 'Integral Image' [15]. The integral image is computed rapidly from an input image and is used to speed up the calculation of any upright rectangular area. Given an input image I (grayscale representation) and a point (x,y) the integral image I_Σ is calculated by the sum of the values between the point and the origin. Formally this can be defined by the formula:

$$I_\Sigma(x,y) = \sum_{i=0}^{x}\left(\sum_{j=0}^{y} I(x,y)\right) \quad (2.1)$$

Using the integral image, the task of calculating the area of an upright rectangular region is reduced to four operations. If we consider a rectangle bounded by vertices A, B, C, D as in Figure 1, the sum of pixel intensities is calculated by:

$$\sum = A + D - (C + B) \quad (2.2)$$

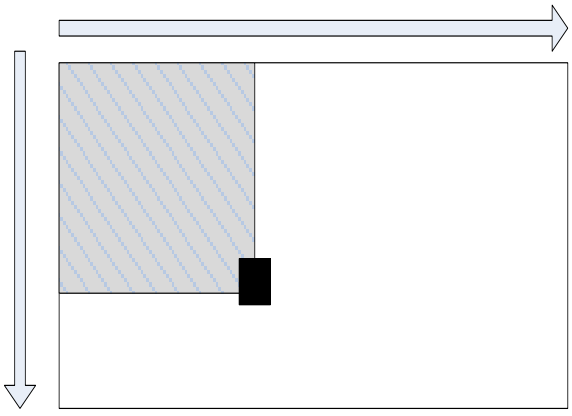


Figure 2.1: Integral Image Computation

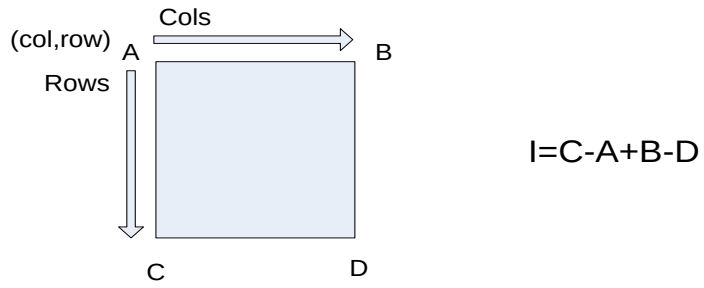


Figure 2.2: Area computation using integral images

Since computation time is invariant to change in size this approach is particularly useful when large areas are required. SURF makes good use of this property to perform fast convolutions of varying size box filter at constant time. Once the integral image has been computed, it takes three additions and four memory accesses to calculate the sum of the intensities over any upright, rectangular area.

2.2.2 Interest Point Detection

The SURF detector is based on the determinant of a Hessian-matrix approximation because of its good performance in accuracy. In order to motivate the use of the Hessian, we consider a continuous function of two variables such that the value of the function at (x,y) is given by f(x,y). The Hessian matrix is the matrix of partial derivatives of the function f and the determinant of the matrix are presented by the next equations.

$$H(f(x,y)) = \begin{bmatrix} \frac{\theta^2 f}{\theta x^2} & \frac{\theta^2 f}{\theta \chi \theta y} \\ \frac{\theta^2 f}{\theta \chi \theta y} & \frac{\theta^2 f}{\theta y^2} \end{bmatrix} \quad (2.3)$$

$$\det(H) = \frac{\theta^2 f}{\theta x^2}\frac{\theta^2 f}{\theta y^2} - \left(\frac{\theta^2 f}{\theta \chi \theta y}\right)^2 \quad (2.4)$$

The value of the determinant is used to classify maximum and minimum of the function f. Since the determinant is the product of eigenvalues of the Hessian we can classify the points based on the sign of the result. If the determinant is negative then the eigenvalues have different signs and hence the point is not a local extremum (minimum or maximum). If it is positive then either both eigenvalues are positive or both are negative and in either case the point is classified as an extremum (Leading Minot Test theorem) [16].

Translating this theory to work with images rather than a continuous function is a simple task. The function f(x,y) is replaced by the image pixel intensity function I(x,y). The calculation of the second order partial derivatives of the image becomes by convolution with an appropriate kernel. In the case of SURF the second order scale normalized Gaussian is the chosen filter as it allows analysis over scales. We can construct kernels for the Gaussian derivatives in x,y and combined xy direction such that we calculate the four entries of the Hessian matrix. Use of the Gaussian allows varying the amount of smoothing during the convolution stage so that the determinant is calculated at different scales. Furthermore, since the Gaussian is a circularly symmetric function, convolution with the kernel allows for rotation invariance. Gaussians are optimal for scale-space analysis [17, 18]. Hessian matrix as function of both space $\mathbf{x}$ = (x,y) and scale σ.

$$H(\mathbf{x},\sigma) = \begin{bmatrix} L_{xx}(\boldsymbol{x},\sigma) & L_{xy}(\boldsymbol{x},\sigma) \\ L_{xy}(\boldsymbol{x},\sigma) & L_{yy}(\boldsymbol{x},\sigma) \end{bmatrix} \quad (2.5)$$

Here $L_{xx}(\mathbf{x},\sigma)$ refers to the convolution of the second order Gaussian derivative $\frac{\theta^2 g(\sigma)}{\theta \chi^2}$ with the image at point $\mathbf{x}$=(x,y) and similarly for L_{yy} and L_{xy}. These derivatives are known as Laplacian of Gaussians. Thus, determinant of Hessian is calculated for each pixel and this value is used to find interest points.

Bay [1] proposed an approximation to the Laplacian of Gaussians by using box filter representations of the respective kernels. The use of these filters in combination with the use of integral images described in previous section increases the performance. To quantify the difference we can consider the number of array accesses and operations required in the convolution. For a 9x9 filter we would require 81 array accesses and operations for the original real valued filter and only 8 for the box filter representation. As the filter is increased in size, the computation cost increases significantly for the original Lapalacian while the cost for box filters is independent of size.

Figure 2.3 gives a brief explanation of box filters and how these are implemented in an image. For the D_{xy} filter the white regions with a value of -1 and the black with a value of 1 and the remaining areas are not weighted at all. The D_{xx} and D_{yy} filters are weighted similarly but with the white regions have a value of -1 and the black with a value of 2. Simple weighting allows for rapid calculation of areas but in using these weights we need to address the difference in response values between the original and approximated kernels. Bay [1] proposes the following formula as an accurate approximation for the Hessian determinant using the approximated Gaussians:

$$\det(H_{approx}) = D_{xx}D_{yy} - (0.9D_{xy})^2 \quad (2.6)$$

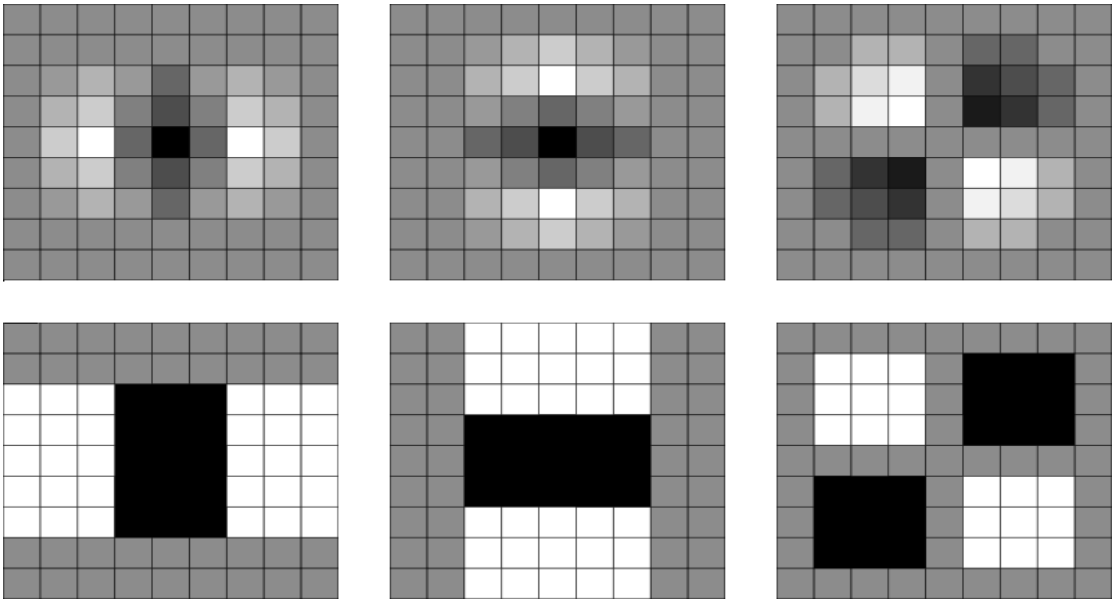


Figure 2.3: Laplacian of Gaussian Approximation. Top Row: Discretised and cropped second order derivatives in the x, y and xy-directions. Bottom Row: Weighted Box filter approximations in the x, y and xy-directions(D_{xx}, D_{yy}, D_{xy})

The necessity of finding objects under different scales leads to the introduction of scale-space concept [19]. A scale-space is a continuous function which can be used to find maximum and minimum across all possible scales [20]. In computer vision the scale-space is typically implemented as an image pyramid where the input image is iteratively convolved with Gaussian kernel and repeatedly sub-sampled. In SURF scale-space is created by applying kernels of increasing size to the original image. This allows for multiple layers of the scale-space pyramid to be processed simultaneously and negates the need to subsample the image hence providing performance increase. The difference between approach with image pyramid and SURF approach is shown in Figure 2.4.

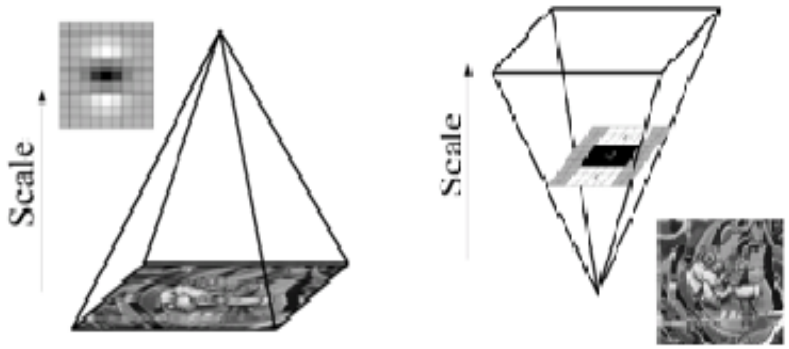


Figure 2.4: Image pyramids vs SURF approach

The scale-space is divided into a number of octaves, where an octave refers to a series of response maps of covering a doubling of scale. Each octave is divided into four intervals and filter size is given equation (2.7). The lowest level of the scale-space is obtained from the output of the 9x9 filters shown in Figure 2.5. These filters correspond to a real valued Gaussian with σ=1.2. Subsequent layers are obtained by up-scaling the filters while maintaining the same filter layout ratio. As the filter size increases, the value of associated Gaussian scale also increased and is calculated by next formula:

$$\text{Filter Size} = 3(2^{\text{octave}}\text{xinterval} + 1) \qquad (2.7)$$

$$\sigma_{\text{approx}} = \text{Current Filter Size} \cdot \frac{\text{Base Filter Scale}}{\text{Base Filter Size}} = \text{Current Filter Size}\ \frac{1.2}{9} \qquad (2.8)$$

When constructing larger filters, there are a number of factors which must be taken into consideration. The increase in size is restricted by the length of the positive and negative lobes of second order Gaussian derivatives. In the approximated filters the lobe size is set at one third the side length of the filter and refers to the shorter side length of the weighted black and white regions. Since we require the presence of a central pixel, the dimensions must be increased equally around this location and hence the lobe size can increase by a minimum of 2. Since there are three lobes in each filter which must be the same size, the smallest step size between consecutive filters is 6. Figure 2.5 presents the structure and the size of subsequent filters.

The construction of the scale space starts with 9x9 filter, which calculate the blob response of the image for the smallest scale. Then filters with sizes 15x15, 21x21 and 27x27 are applied, by which more than a scale of 2 has been achieved. But this is needed, as a 3D non-maximum suppression is applied both spatially and over neighboring scales. Hence, the first and last Hessian response map in the stack cannot contain such maxima themselves, as they are used for reasons of comparison only. Similar considerations hold for the others octaves. For each new octave, the filter size increase is doubled (going from 6 to 12 to 24). At the same time the sampling intervals between processing pixels for the extraction of the interest points are doubled as well as for every new octave. This reduces the computation time and the loss in accuracy is comparable to the image sub-sampling of the traditional approaches (image pyramids). The filter sizes for the second octave are 15, 27, 39, 51 and for the third octave 27, 51, 75, 99.

Once the responses are calculated for each layer of the scale-space, they are scale-normalized. For the approximated filters, scale-normalization is achieved by dividing the responses by the filter area. This leads to perfect scale-invariance.

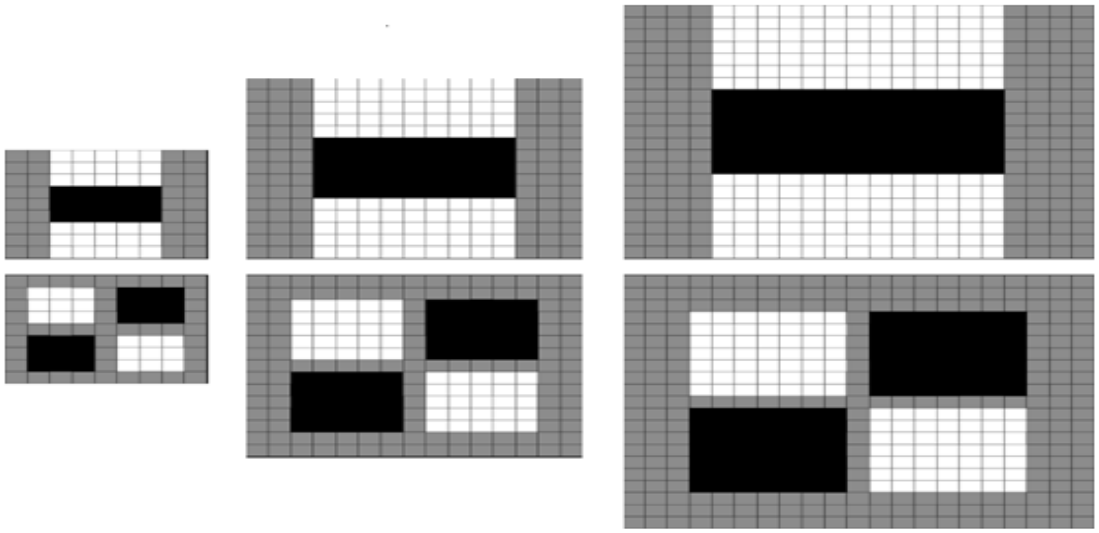


Figure 2.5: Filter Structure. Subsequent filter's sizes differ by a minimum of 6 to preserve filter structure

Final part of detection consists of interest point localization. This task is divided into three steps. Firstly, the determinants are compared to a predetermined threshold and localization is not performed for values under this threshold. Increasing the threshold lowers the number of detected interest points, leaving only the strongest while decreasing this threshold allows more interest points to be detected.

After the previous process, a non-maximal suppression is performed to find a set of candidate interest points. To do this each pixel in the scale-space is compared to its 26 neighbors, comprised of the 8 points in the native scale and the 9 in each of the scales above and below (non-maximal suppression in a 3x3x3 neighborhood). Before these comparisons, we have found the maximum value in the 3x3 region between the two intermediate intervals of each octave. Figure 2.6 describes then non-maximal suppression step. Essentially, algorithm finds the maximum value and if it is less than threshold, the localization is not performed. Scale space interpolation is especially important in our case, as the difference between the first layers of every octave is relatively large.

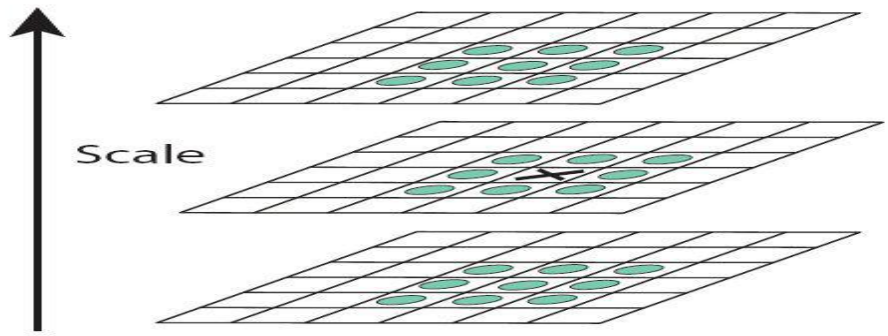


Figure 2.6: Non-Maximal Suppression

The final step of interest point localization involves interpolating the nearby data to find the location in both space and scale to sub-pixel accuracy. This is done by fitting a 3D quadratic as proposed by Brown [39]. In order to do this we express the determinant of the Hessian function H(x,y,σ), as a Taylor expansion up to quadratic terms centered at detected location. This is expressed as:

$$H(x) = H + \frac{\theta H^T}{\theta \chi} x + \frac{1}{2} x^T \frac{\theta^2 H}{\theta \chi^2} x \quad (2.9)$$

The interpolated location of the extremum (candidate interest point resulted by non-maximal suppression), x=(x,y,σ), is found by taking initially the derivative of function in equation 2.9 and setting it to zero such that:

$$X = - \frac{\theta^2 H^{-1}}{\theta \chi^2} \frac{\theta H}{\theta \chi} \quad (2.10)$$

The derivatives are approximated by finite differences of neighboring pixels of the candidate interest point's neighboring determinant values. The result of equation (2.10) is a vector with three values. These three values determine the exact position of the interest point (x,y,scale). In case all three values are less than 0.5 they are adjusted as following: x and y are multiplied with the sampling interval (described previously), the result of the multiplication is added to x and y respectively and the final results are rounded to the nearest integers. The new scale is computed based on equation (2.8). In this formula the requested filter size is computed by equation (2.7) while the interval value represents the actual scale of the reported point.

2.2.3 Interest Point Description

The SURF descriptor describes how the pixel's intensities are distributed within a scale dependent neighborhood of each interest point detected by the detection part. Extraction of the descriptor is divided into two distinct tasks. First each interest point is assigned an orientation before a scale dependent window is constructed in which a 64-dimensional vector is extracted. All calculations for the descriptor are based on measurements relative to the detected scale in order to achieve scale invariant results. In this work is implemented only the orientation assignment.

In order to achieve invariance to image rotation each detected interest point is assigned a reproducible orientation. Extraction of the descriptor components is performed relative to this direction so it is important that this direction has to be repeatable under varying conditions. To determine the orientation, the concept of Haar Wavelets is used.

Haar wavelets are filters which used in combination with integral images in order to increase robustness and decrease computation time. These simple filters represented by figure 2.7 can be used to find gradients in the x and y directions. The left filter computes the response in the x-direction and the right in the y-direction. Weights are 1 for black regions and -1 for white.

Figure 2.7: Haar Wavelets

Haar wavelet responses of size 4σ are calculated for a set of pixels within a radius of 6σ, where σ refers to the scale of the detected interest point (σ is a rounded value of detected decimal value of scale). The set of pixels is determined by sampling those from within the circle using a step of σ. The responses are normalized by value that follows Gaussian distribution with standard deviation of 2.5σ where σ is the detected scale, centered at the interest point. The weighted responses are represented as points in vector space with the x-response and y-response as values. For each x,y-response we compute the orientation of the formed vector. The set of pixels around interesting point as mentioned previously, is described by x,y-response and the respective orientation. The dominant orientation is selected by rotating a circle segment covering an angle of $\frac{\pi}{3}$ around the origin, every 11 degrees. At each circle position, the x and y-responses for which the orientation belongs to angle space of a specific circle segment, are summed and used to form a new vector. We compute the length of the summed x and y-responses and the orientation of the vector formed by these summed responses, for every circle segment. The longest vector among all circle segments, gives its orientation to the interest point. The process of orientation assignment is described by the Figure 2.8.

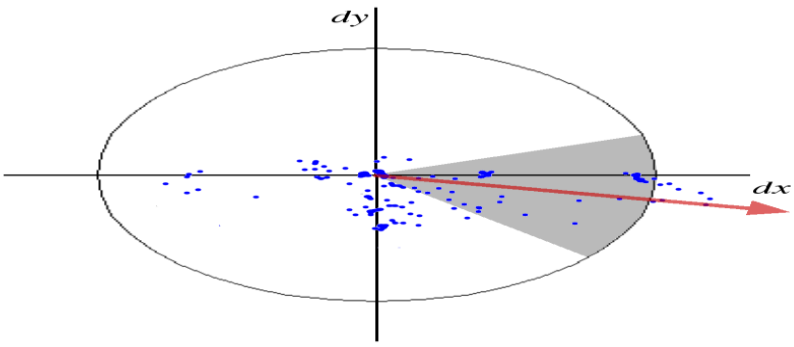


Figure 2.8: Orientation assignment with a sliding window of $\frac{\pi}{3}$.

2.3 Algorithm

This section describes the basic steps of detection and orientation task processes.

1. Calculate the integral image representation.

2. Calculate the determinant response map into scale-space level.

3. Scale normalization of the determinant response map

4. Perform non maximal suppression to localize interest points in scale-space.

5. Result of non-maximal suppression compared to a predetermined threshold.

6. Interest point localization into scale-space based on equation (2.10)

7. Calculate the orientation of each interest point.

2.4 Field-programmable gate arrays (FPGAs)

A field-programmable gate array (FPGA) is a semiconductor device that contains programmable logic components, programmable interconnects, memory blocks and I/O elements [22] Figure 2.10. Modern FPGAs contain also include dedicated arithmetic circuitry, such as embedded multipliers and adders, and microprocessors either as fixed embedded components (‘hard’ processors) or software module (‘soft’ processors).

The basic programmable unit of an FPGA is a look-up table (LUT). The latest families of FPGAs such virtex-5 contain six-input LUTs, which can implement any basic logic function of up to six input variables (AND, OR, NOT, XOR, etc.). Register elements around the LUTs hold the values of the signals until some specified condition is met (generally a clock edge). LUTs and registers are combined into logic blocks to control both functionality and timing. Figure 2.9 shows the structure of a generic logic block. It includes multiplexers to choose between inputs and outputs, and a carry chain that can be used to cascade several blocks for implementation of fast adders and multipliers. Complex circuits can be designed by combining the functionality of multiple logic blocks. Thus, the processing power of an FPGA is directly related to the number of logic block it contains.

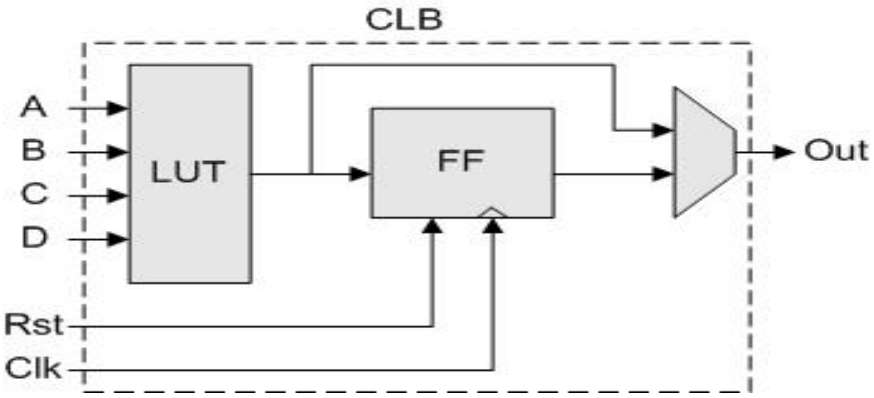


Figure 2.9: Logic block.

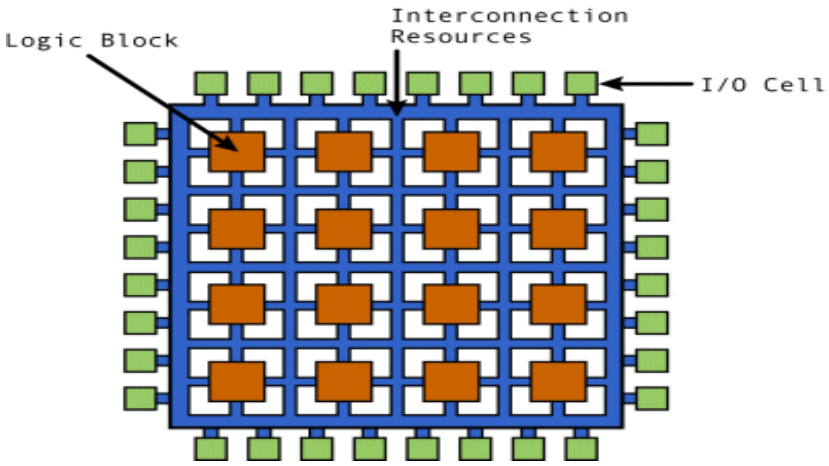


Figure 2.10: Structure of a generic FPGA.

Hardware description language such as VHDL and Verilog allow designers to specify the desired behavior of the circuit in terms of basic logic functions, or more operations such as counters and multipliers. In addition, FPGA vendors offer intellectual property (IP) cores that implement commonly used functions, and that are optimized for their target FPGA architectures. FPGA vendors offer also software tools that can analyze the description of the circuit, decide on the optimal routing and allocation of resources, and translate the information into a bit-stream that can be downloaded onto the chip to configure it.

2.5 Other vision systems on FPGAs and GPUs

FPGAs and GPUs are increasingly being used for feature extraction and other vision tasks to improve the performance of real-time systems. The most recent FPGA-based systems implement tasks such as stereo disparity estimation [23, 24], optic flow computation [25], template matching [26] and gesture recognition [27, 28], among others.

Among the FPGA-based feature detectors in the literature, many implement Tomasi and Kanadi's corner detector [29] as an initial stage for tracking [30, 31, 32, 33]. This detector is robust to rotation and translation, but not to changes in scale. Se [34] use the scale invariant SIFT features [4] for localization and terrain modeling in the ExoMars. Their system can compute SIFT locations for a 640x480 image in 60 ms using a Xilinx Virtex II FPGA. Bouganis [35] compute corner locations using complex steerable wavelets at four different scales and orientations. Sift and Surf are also implemented in GPUs [36, 37]. GPU-SIFT took on average 30 ms for the detection of 1000 features in 1024x768 resolution and GPU-SURF takes for 500 features in 640x480 resolution 11.7 ms. Recently, a CPU-based parallel algorithm was presented [41]. The testing system of this implementation was equipped with an Intel Core Cuo P8600 at 2.4 Ghz and it was able to extract and represent features (only orientation part of this task is performed by our implementation) from 640x480 image at a rate of 33 frames per second.

2.6 Design Methodology

The design of the feature detector started by dividing the algorithm into functional modules and implementing them in C(using OpenCv library) using floating point arithmetic. This provides valuable information on the types of operations involved in the algorithm, as well as the dynamic range and precision of the parameters and intermediate results. The source code we used for the evaluation has been downloaded directly from [42] and it is distributed under the GNU General Public License v4.

Once the floating point implementation was tested and verified, a fixed point version of the algorithm was derived from the floating point implementation by limiting the range and the precision of all intermediate results to an explicit number of integer and fractional bits. This fixed point implementation was used to study the effect of fixed point arithmetic on the algorithm, to determine which parameters and intermediate results affected the final results most significantly, and to select the optimal number of bits to represent each quantity. This information was used to design the hardware architecture and to code the description of the circuit in VHDL.

Once coded, each module in hardware design underwent several stages of verification. First, the VHDL descriptors were simulated in Modelsim using specially designed testbenches verify that all control signals and arithmetic results occurred at the correct cycles (cycle-true verification). Second, intermediate results from the Modelsim simulations were saved into text files and then they were compared to the corresponding results in the fixed-point implementation.

The development tools used in this project were ModelSim 6.3f, Xilinx ISE 10.1, OpenCV library and gcc compiler(4.3.4).

2.7 Summary

This chapter introduces some of the concepts applied throughout this thesis, presents a brief literature review on feature detection algorithms and FPGA-based, GPU-based vision systems and discusses the design methodology in the implementation of the hardware system.

Features are points or regions of interest in an image. There are numerous approaches to feature detection, however they all attempt to achieve invariance to viewing conditions to facilitate the description and matching of objects across images. Modern feature detection algorithms are robust to rotation and translation, as well as, to changes in scale transformations.

The system described in this thesis is an FPGA-based implementation of the interest point detection and orientation assignment part of SURF detector introduced by Herbert Bay, Andreas Ess, Tinne Tuytelaars, Luc Van Gool [1]. This system is implemented in Virtex5 vsx240T FPGA device. The use of FPGAs is expected to increase the speed of the feature detector considerably, at the expense of loss of numerical precision. Chapter 3 provides a detailed description of the design of the hardware implementation. Chapter 4 lists the hardware resources for each processing module, computational time for software, hardware and compares the accuracy of the results. Chapter 5 presents some ideas to improve performance in terms of speed and hardware resources.

Chapter 3

Hardware Design

This chapter describes the implementation of the interest point detector on Virtex5 vsx240T board and presents the reasoning behind the most important design decisions. Section 3.1 describes a high-level overview of the system. Section 3.2, 3.3, 3.4, 3.5, 3.6 and 3.7 provide a detailed description of the core modules of the feature detector. Section 3.8 discusses the methods and difficulties in translating an algorithm designed for floating-point computation into a fixed-point design suitable for implementation on FPGAs. Finally, the design was successfully placed & routed in the device Virtex5 vfx200T. The device choice was determined by the memory-intensive nature of our application.

3.1 System Overview

Figure 3.1 shows the high-level architecture of the system. Input image is converted to a grayscale representation and all processing stages of detector use these pixel values. The system can process images of a standard video (640x480 pixels).

The core of the detector consists of the main three stages described in chapter 2. The first stage computes the integral representation of the input image. The second stage finds the interest points. This stage is divided into three basic parts. First part computes the determinant response map and accesses the integral image module. It is important to mention that sequential memory accesses guarantee the maximum speed in terms of hardware design. The second part performs a non-maximal suppression and the third part is the interest point localization while determinant calculation continues its process. The latter stage computes the orientation of the interest points extracted from previous stages (stored into BRAMs).

One of the considerations of the design of the detector was to allow for scalability. For example, whenever possible the bit-widths of the parameters and intermediate signals have been defined as generic parameters, to allow for easier modification in case different numerical precision is required. DSP multipliers were extensively used for all required multiplications using LogicCore generator (optimal pipeline stages were also used to increase speed) and all look-up tables were implemented using slice LUTs.

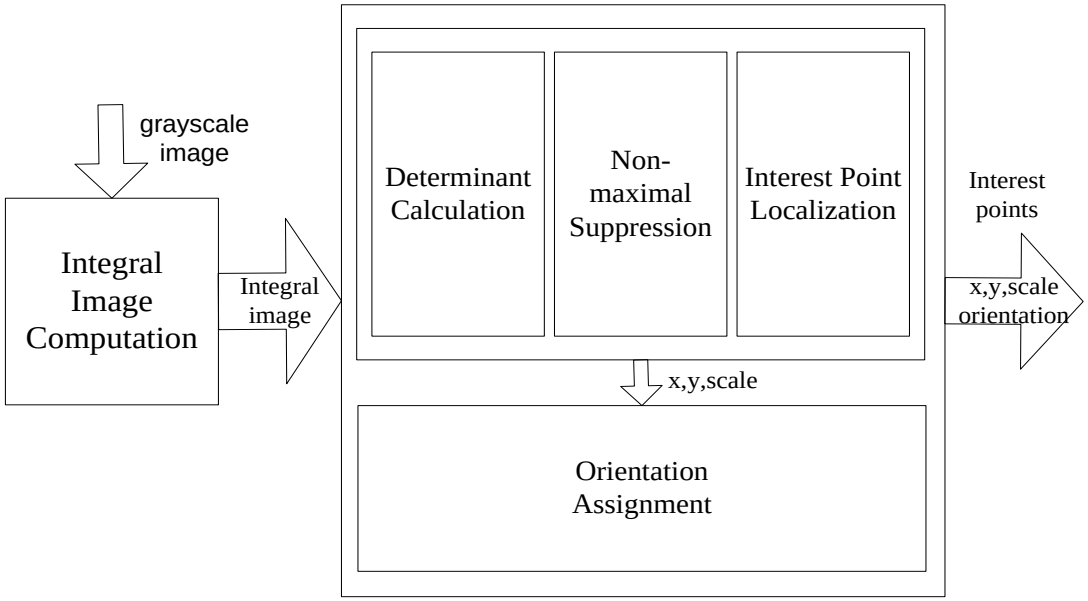


Figure 3.1: High-level Architecture of feature detector

3.2 Grayscale Image Module

Grayscale image contains one value per pixel described by eight bits. The large amount of on-chip memory in Virtex5 vsx240T board allows grayscale image to be loaded (prefetching) while detection and orientation parts continue their process using integral image representation.

A partition in two parts of the grayscale image leads to better memory utilization. Total number of pixels in 640x480 image dimension is 307200. Thus, first part of grayscale image contains 2^18 (262144) pixel values of eight bits and converted to rows means 640x400 (256000) and second part contains 2^16 (65536) pixel values where can stored the last 80 rows of image (640x80 = 51200). The number of bits without partitioning is 524288*8 because 524288 positions are required to store 307200 pixel values in power of two addressing. A little waste of memory is unavoidable but a gain of 1.5 Mbit is achieved compared to an unpartitioned memory scheme. Figure 3.2 represents the partitioned grayscale image and its ports. The signal sel is referred to the selection of first or second part for reading or writing based on the values of respective addresses.

The integral image representation is computed and stored to integral image module (section 3.3). The processing tasks (detection and orientation) communicate only with the integral image module. Thus, a new frame can be loaded to grayscale image module after integral image computation. Integral image is not the critical part of the whole procedure as it requires only 1.5ms to complete.

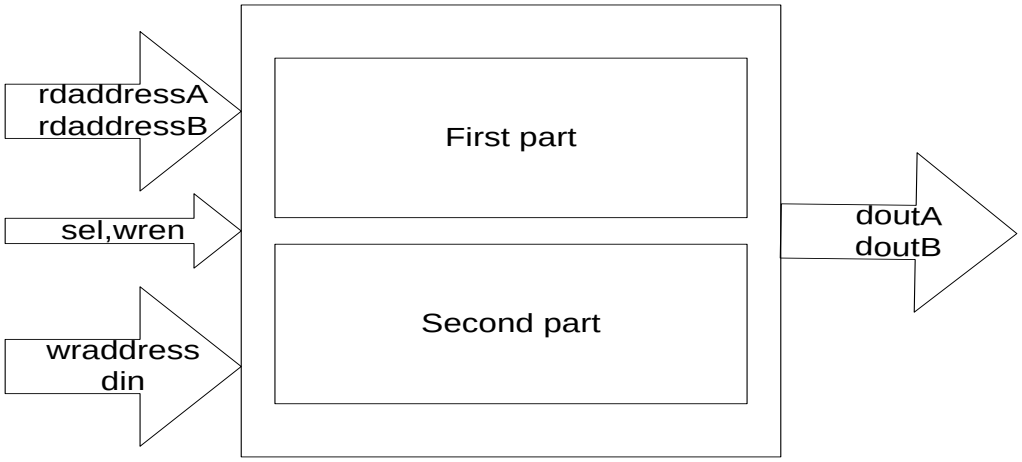


Figure 3.2: Grayscale Image.

3.3 Integral Image Module

The integral image module computes the integral representation of grayscale supplied input image and described in Chapter 2. In addition, this module handles integral images. Pixel values are normalized by factor of 255 in software. However, in our fixed-point implementation we normalize by a factor of 256 in order to replace the division with a simple bit-shifting while not loosing significant accuracy.

Integral image is divided into five parts. Each part holds 640x100 pixels except for the last which holds 640x80 elements. Each integrated pixel is a 32-bit value with the last eight bits as a fractional part. This specific partition of integral image into five parts achieves both better utilization of on-chip memory and allows more cores of detection to process the same image. Each memory part has 2^16 (65536) positions and stores 640x100 (64000) integrated pixels except for the last which uses the 640x80 positions of the fifth part. Thus, this scheme is able to load a complete frame (640x480 pixels) on the available BRAMs of vsx240T device and also it is possible the parallel processing of image data. Figure 3.3 depicts integral image module and how communicates with detection and orientation tasks.

This module reads the values of the integral image. The inputs are the two points in image (row_one,col_one) , (row_one,col_two) , (row_two,col_one) , (row_two,col_two) for each detection core and the outputs the four respective integral pixel intensities of the image. Row_one, row_two must be converted to the respective number of elements. For example, if row has as value 2, it means that 2*640 elements, beginning row from value 0 and for a 640x480 image. This conversion is done using a look-up table of 100 positions.

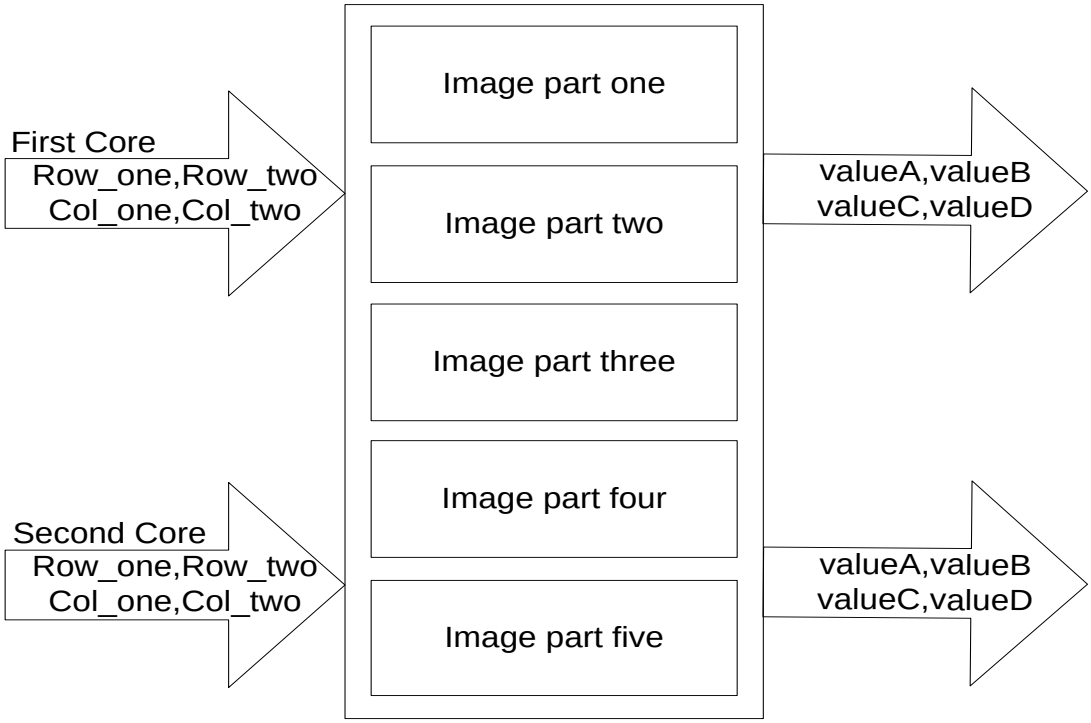


Figure 3.3: Integral Image Module.

A critical point for interest point detection in order to be parallelized is that box integral function (computes any rectangular area intensity) reads elements from rows with different odd number. Thus, a partition in odd and even rows for each of the previous memory parts leads to x2speedup as in each cycle we now process two rows considering only the interest point detection (not the orientation part of the algorithm). This module can process more rows if we can satisfy the condition of non-overlapping rows between the five parts of the integral image.

3.4 Determinant Module

The determinant module computes the determinant at each pixel location for all scales based on different filter sizes. Figure 3.4 represents Determinant Module's architecture.

In chapter 2 was analyzed the notion of scale-space and a specific scale is represented by a specific number of octave and interval. Each octave and interval is used to retrieve a set of coefficients from look-up tables. These coefficients form the dimensions of box filters and one constant used to perform division by the method of multiplication. A detailed explanation of these look-up tables is provided in Section 3.4.1.

The first image processing in this stage finds the parameters for specific octave and interval. Second stage computes the eight rectangular area intensities that

required computing the elements of the Hessian Matrix (D_{xx}, D_{yy}, D_{xy}) applying filters shown in Figure 2.3. These results used by the final stage to calculate the determinant of the matrix based on equation (2.6).

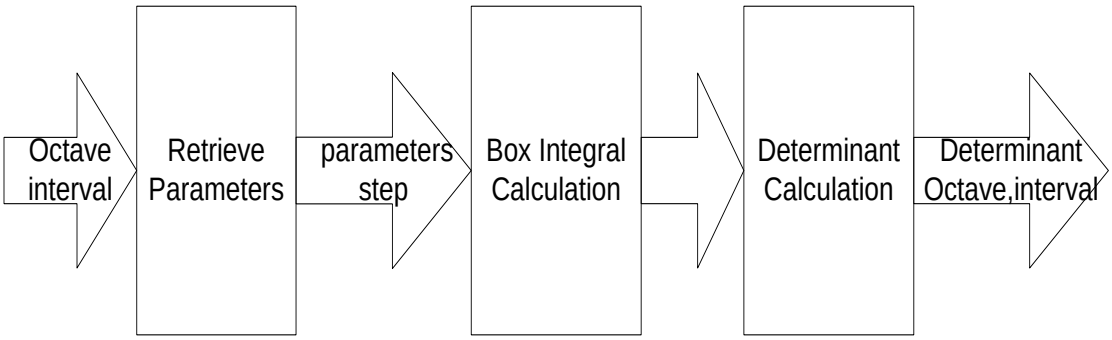


Figure 3.4: Determinant Module

3.4.1 Retrieve Parameters Module

Each octave and interval represents a discrete value of scale. The concept of scale-space explained in detail in Chapter 2. For a specific scale are applied box filters with the respective dimensions. These dimensions are located in look-up tables. Figure 3.5 represents the filter lengths for the different octaves and intervals. It becomes clear in this figure that second and last scale in each octave is repeated in next octave with different sampling step (sampling interval) in space. Finally, the number of detected points decays quickly for new octave and three octaves are enough to describe the important information of the image.

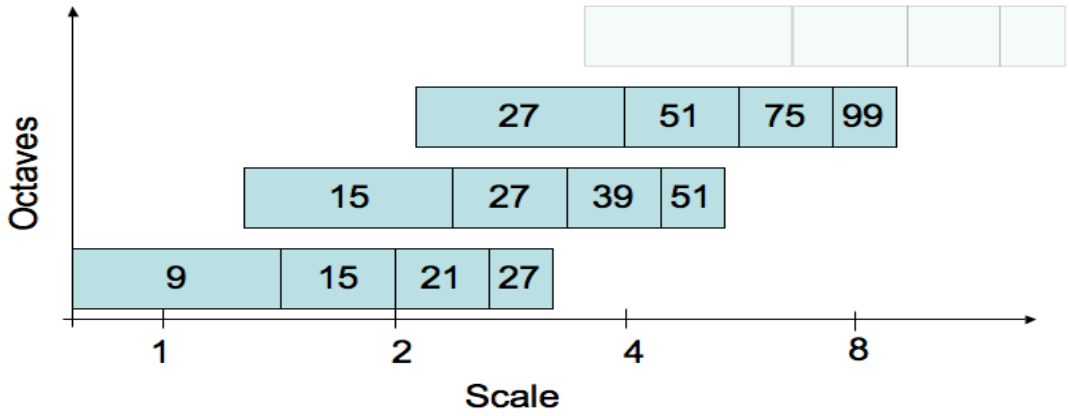


Figure 3.5: Filter lengths for different octaves and scales

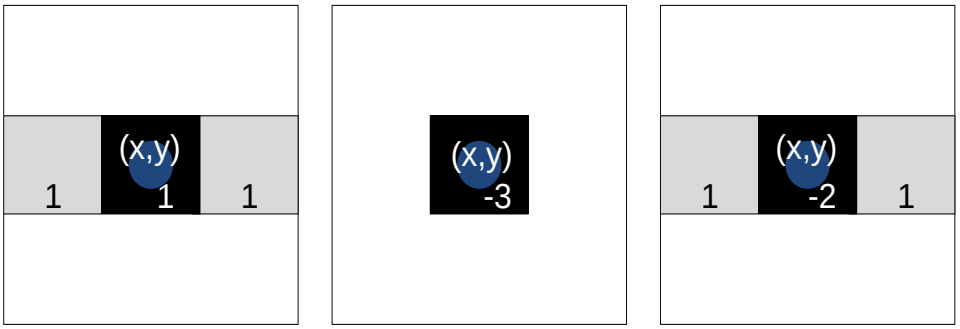
Box filters are centered to current processing pixel and different values are required to compute the responses. A normalization factor is used for each response, it is constant and pre-computed for each scale.

3.4.2 Box Integral Calculation Module

Basic unit of this module is the Boxintegral function. This function communicates with Integral Image module, performs four memory accesses, three additions/subtractions and calculates the intensity of a rectangular area. Totally, eight rectangular area calculations are required, two for Dxx, two for Dyy and four for Dxy.

A logic unit based on filter parameters and current pixel position computes the inputs of eight calls of boxintegral function. The whole process is represented by Figure 3.6. These eight values of rectangular areas pixel intensities are stored in buffers in order to be forwarded to the next processing module. The respective processing pixel (row, column, interval) and the normalizing factor for the current interval are also stored in buffers. Determinant calculation module begins when the eight rectangular areas have been calculated. The necessity of using multipliers with pipeline stages leads to make a fully pipelined determinant calculation module and buffers are used for storing intermediate results.

a)



b)

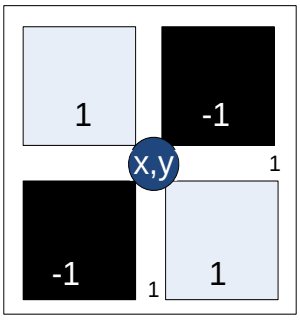


Figure 3.6: Description of Dxx,Dyy,Dxy computation process

a)Dxx, similarly for Dyy, b) Dxy

3.4.3 Determinant Calculation Module

This module performs three basic operations. Initially, it computes Dxx, Dyy, Dxy. Dxy requires only additions/subtractions. Dxx, Dyy are using a multiplier by 3 with pipeline stages. Dxy is normalized by using the normalizing factor and after Dxx and Dyy are computed, then they are also normalized. Normalization is a division by filter area (constant) which is implemented with the method of multiplication with the inverted value of normalizing factor (filter area). It results improvement both in speed as division circuit adds latency and in hardware resources utilization.

Finally the equation 2.6 computes the determinant of approximated Hessian matrix. All operations are performed using multipliers with optimal number of pipeline stages. Making this unit pipelined allows the calculation of determinants every eight cycles as the numbers of rectangular areas with an initial delay of 21 cycles.

3.5 Non-maximal Suppression Module

The non-maximal suppression module finds the interest points all over the scales. There are three main cores that find and localize exactly the interest points both in scale and in space. Figure 3.7 presents the overview of this module and in the next sections will be described the internal structure of these sub-cores. The most significant parts are the store determinant module which differs from software implementation and matrix inversion (sub-core of interest point localization module and implemented with single precision floating point arithmetic instead of the double precision of the software implementation).

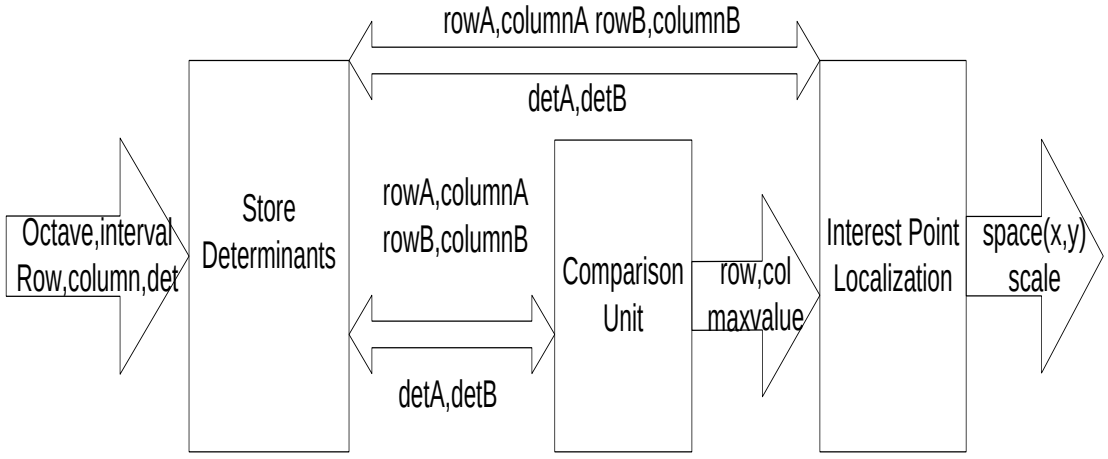


Figure 3.7: Non-maximal suppression overview

3.5.1 Store Determinant Module

In section 3.4 was mentioned that determinants are calculated for each pixel location (space), octave and interval which represent a discrete value of scale. The necessity of this module is that determinant response map in software implementation is stored into a matrix with total size equal to image size multiplied with the number of octaves*intervals.

A modification in sequence of the determinant computation allows for a better utilization of on chip memory while enables the non-maximal suppression to be executed in parallel with the determinant calculation.

Based on the values that non-maximal suppression module needs to perform its function, determinants are computed for one octave, for each pixel location and for four intervals (octaves are splitted into intervals). This module needs the determinants of five rows for all intervals to begin the processing stage. Three rows are needed to find the maximum determinant in 3x3 regions of two intermediate intervals of each octave. The two neighboring rows of these three rows are needed to find if the value from previous comparison is indeed the maximum value among neighboring pixels determinants of interval that maximum value was detected and neighboring intervals. Dividing the determinant response map every three rows we have to store also the previous and the next row of these three rows in order to perform the whole process of this module.

The main reason for the previous modification was the limited on chip memory in comparison with the size of determinant response map. The fact that this module functions independently from the determinant module leads to the usage of two extra ‘store’ structures of five rows size as previously described. The determinant module continues its processing stage while non-maximal suppression module process one of these ‘store structures’. Figure 3.8 presents the general architecture of ‘store’ structure and this is repeated for all intervals (totally 4) of each octave.

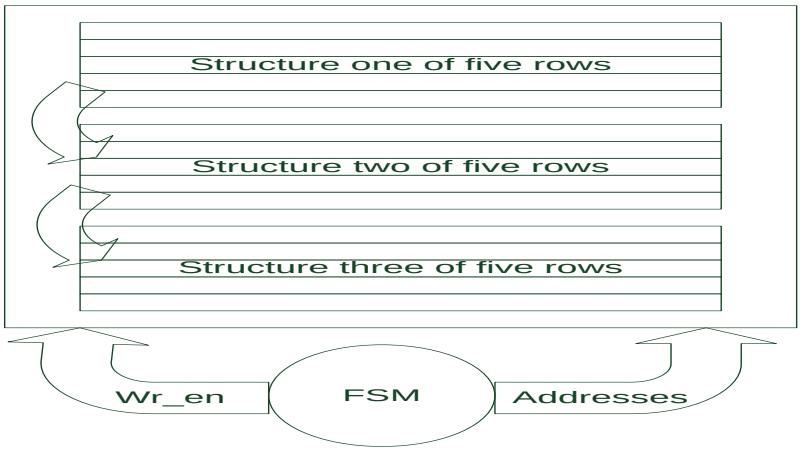


Figure 3.8: Store determinant basic structure

Inputs are write and read addresses and respective enables for each five_row module.

A **F**inite **S**tate **M**achine is used to select which component of five rows is available to write the determinants. Also, it computes which of these three components is ready so as the non-maximal suppression will start the processing. It writes for each 'five row' structure the value of the row which is stored in second row and it is the starting row of the next processing modules. This value is used to compute the pixel position in image resolution, based on the relative position in five row 'store' structure. In addition, write address computation is relative to current state of fsm. Table 3.1 depicts the values of signals for all states.

State	address_one	address_two	address_three	one_enable	two_enable	three_enable	ready_one	ready_two	ready_three
A	640	x	x	1	0	0	0	0	0
B	2*640	x	x	1	0	0	0	0	0
C	3*640	0	x	1	1	0	0	0	0
D	4*640	640	x	1	1	0	0	0	0
E	x	2*640	x	0	1	0	1	0	0
F	x	3*640	0	0	1	1	1	0	0
G	x	4*640	640	0	1	1	1	0	0
H	x	x	2*640	0	0	1	0	1	0
I	0	x	3*640	1	0	1	0	1	0
K	640	x	4*640	1	0	1	0	1	0
L	2*640	x	x	1	0	0	0	0	1
M	3*640	0	x	1	1	0	0	0	1
N	4*640	640	x	1	1	0	0	0	1

Table 3.1: FSM signals

Enable signals write the determinant to one or two of these three 'store' structures and the value of interval select one of these four basic structures. The structure of Figure 3.8 is repeated for four intervals. Next section analyzes more how the reading of stored determinants becomes. State changes when the determinants for all elements of one row are computed. A,B,C states are used only for the first three rows and only other states are used for rest rows of image. As mentioned, each five_row structure used to perform 3x3 comparisons. Therefore, for each structure the value of first row from 3 is stored and supplied to next module.

3.5.2 Comparison Unit Module

This module performs the non-maximal suppression and finds candidate interest points as previously described. It is divided into two different comparison sub-units. The first sub-unit finds the maximum value in 3x3 regions of two intermediate intervals. One FSM which is based on initial values of row, column, produces all nine values of row, column to complete its function for two intervals. A second sub-unit,

using the row, column of detected maximum value from previous sub-unit, is enabled if maximum value is greater than a predetermined threshold, and produces the addresses for all nine values of neighboring pixels both for detected interval and for neighboring intervals. A general overview of this module is represented by Figure 3.9 and describes the main inputs and outputs needed to communicate with the store determinant module.

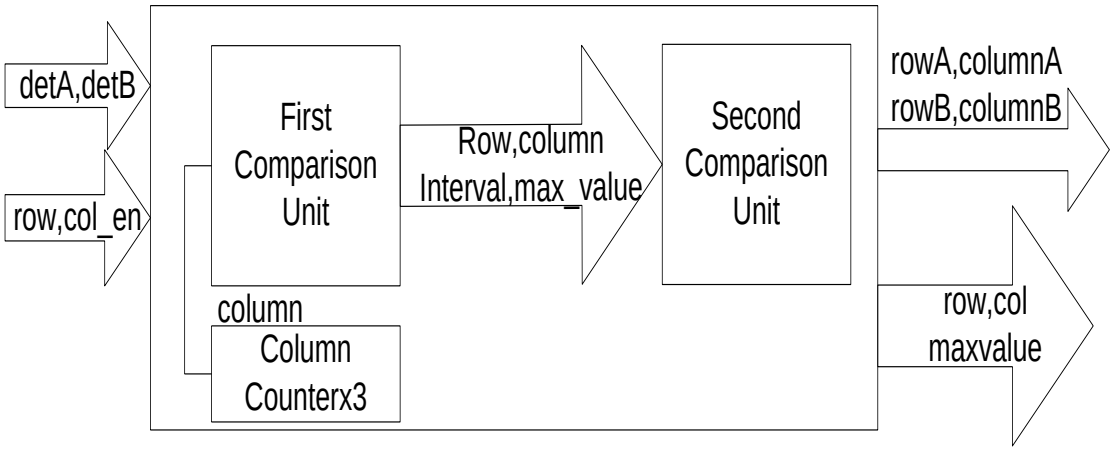


Figure 3.9: Comparison Module overview

Determinant response map is divided into 3x3 regions. Row is given from previous module and it already comes every three rows. Column counter is enabled when a 3x3 region is processed and interest point is detected and localized. Next module makes the localization and the next 3x3 region is provided to comparison module until all regions of five_row structure are completely processed.

3.5.3 Interest Point Localization Module

As mentioned previously, the scale takes discrete values due to the nature of implemented box filters. The space is also discrete as pixels are in discrete positions in the image. The difference between neighboring discrete values of scale enforces the use of the interpolation method described in chapter 2.

This method needs the creation of two matrices which have as elements derivatives approximated by finite differences of neighboring pixels not only in space but also in scale (neighboring intervals). After the matrix creation using again the store determinant module to read the determinants, we have one symmetric 3x3 matrix and one 3x1 matrix. It is performed a matrix inversion of the 3x3 matrix and then a matrix multiplication of the inverted matrix with the 3x1 matrix. The result is a 3x1 matrix containing factors which they are used to adjust both pixel location and scale if the values of these factors are not greater than 0.5. Figure 3.10 provides an overview based on the aforementioned and components are explained further down.

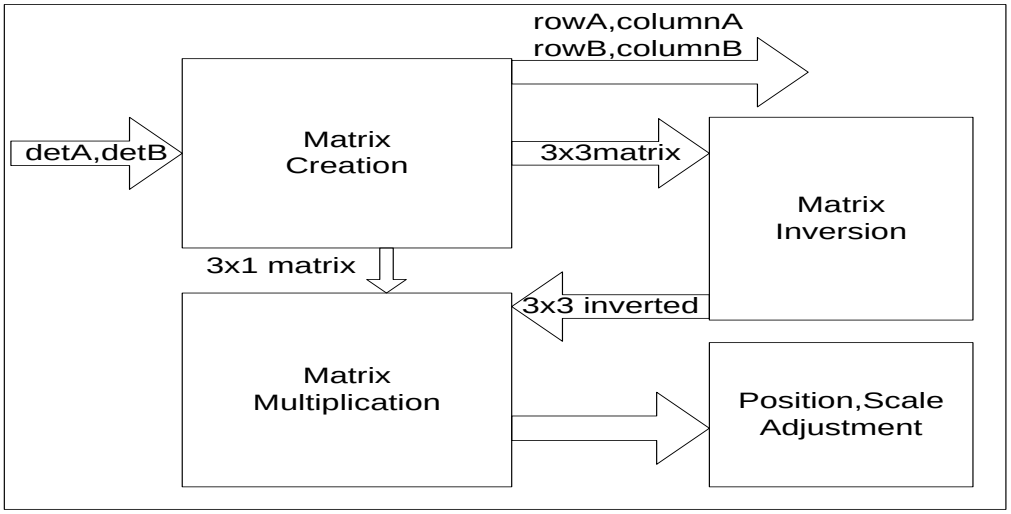


Figure 3.10: Interest Point Localization overview

Matrix creation reads determinants of neighboring pixels. It implements additions/subtractions, simple multiplication by two and division by four with left and right shifts respectively. FSM produces the neighboring pixels needed to create the elements of the two matrices.

Matrix inversion has many difficulties to be implemented successfully in hardware. Especially in our case where all matrix values belong to space [0, 1) the determinant of the 3x3 matrix is a very small number. This results to an inverted determinant which is a large number and can't be represented easily by fixed point arithmetic. Therefore, matrix inversion works with floating point numbers of single precision. Firstly, matrix elements are converted from fixed point to single precision floating point numbers. Using floating point adders, multipliers and divide units, we have to take advantage of pipeline provided by these units and limit the number of these. In addition floating point units require more cycles to produce the results but it doesn't affect the whole processing time as it becomes parallel with determinant computation. Finally, FSM controls the results from floating point units and determines also their inputs in order to perform correctly matrix inversion.

Position and scale adjustment uses the three elements produced by matrix multiplication, the initial position and the scale of the detected interest point are changed if the absolute values of these are not greater than 0.5. Firstly, these floating point values are converted to a fixed point representation. Each octave has a sampling interval (step) as explained in section 2.2.2, this step is multiplied with two from the converted results of matrix multiplication and are used to adjust the pixel position. Box filter size and scale are determined by octave and interval as shown in equation 2.7 and 2.8. Based on these equations comes the next equation which presents how the scale adjustment becomes.

$$\text{Scale} = \frac{1.2}{9} * (3 * (2^{\text{octave}+1} * (\text{interval} + \text{adjust} + 1) + 1)) \qquad (3.1)$$

In previous equation, the factor 1.2/9 is converted to a fixed point representation, 'adjust' is one element from previous matrix multiplication (relative to scale), multiplications are implemented and scale adjustment is completed.

Finally, the detected interest points which contain the pixel position and scale are stored in the devices's-on-chip memory. Once interest point detection part has finished, every interest point is supplied to next processing module which assigns an orientation to each point.

3.6 Detection using a dual core implementation

The large amount of available hardware resources in Virtex5 vsx240T device allows the two cores of feature detection module described in section 3.4 and 3.5 to be ported in the same FPGA. Some changes relative to the first and last row of each processing core are implemented.

The partition of the image as explained in section 3.3 facilitates the use of two cores without additional storage requirements. Rows used from first and second core belong to different parts of image. Thus, each core begins the processing stage from different initial rows and finishes also in different rows. Clearly, the initial rows for first core are the starting processing rows of the algorithm (applied filters must not exceed image dimension) and the final rows for second core are the ending rows of the algorithm.

The ending rows of first core and the initial rows of second core are determined based on the values needed by each core. We decide the last processing row of first core to be the 242 for each octave. However, we need neighboring rows of 242 and the ending rows will be different than 242. In addition, different steps for each octave create different ending rows. Same concept was followed to create the starting rows of second core. It is important to mention that interest point localization functions for every three rows. Thus, 242 is the first row of these three rows and comparison unit needs one extra row than these three rows to complete its process. Also different steps (sampling intervals) for each octave have as consequence different neighboring rows. Starting and ending rows for each processing core are described by Table 3.2.

octave	0	1	2
first_core_init_row	14	26	50
first_core_final_row	250	258	274
second_core_init_row	246	250	258
second_core_final_row	466	454	430

Table 3.2: Dual Core borders

Finally, last changes are needed for the fsm of store determinant module described in section 3.5.1. The first core completes its process when last row determinants are computed and interest point localization for the last three rows is performed (242 and after) then the second core begins its process from the rows of table 3.2 with a modified version of fsm described in Table 3.1. First processing row is needed only for comparison unit. In first core, the first processing row is the starting row of the three rows needed for localization and comparison. In second core, the first processing row is the previously neighboring row of these three rows needed by localization unit. These reasons lead to the modified version of previous FSM.

State	address_one	address_two	address_three	one_enable	two_enable	three_enable	ready_one	ready_two	ready_three
A	0	x	x	1	0	0	0	0	0
B	640	x	x	1	0	0	0	0	0
C	2*640	x	x	1	0	0	0	0	0
D	3*640	0	x	1	1	0	0	0	0
E	4*640	640	x	1	1	0	0	0	0
F	x	2*640	x	0	1	0	1	0	0
G	x	3*640	0	0	1	1	1	0	0
H	x	4*640	640	0	0	1	1	0	0
I	x	x	2*640	0	0	1	0	1	0
K	0	x	3*640	1	0	1	0	1	0
L	640	x	4*640	1	0	1	0	1	0
M	3*640	x	x	1	0	0	0	0	1
N	4*640	0	x	1	1	0	0	0	1
O	0	640	x	1	1	0	0	0	1

Table 3.3: Modified FSM for second core.

3.7 Orientation Assignment Module

This module implements the orientation assignment task for each interesting point as described in section 2.2.3. This section provides a more detailed description both in algorithm and in hardware implementation. The main purpose of this task is to achieve the rotation invariance of the detected features.

Scale invariance is guaranteed by using the scale of each interest point. Responses from Haar wavelets are computed for all pixels in a radius of 6σ (where σ is the scale) around interesting point with step equal to σ between neighboring pixels. In addition, haar responses are weighted with Gaussian values which are stored in a look-up table of size 7x7. It is also computed an orientation operation as described in section 2.2.3, for each pixel. The notion of Haar wavelets was introduced in Chapter

2. Wavelets are filters implemented using the Boxintegral function which described in section 3.4.2. The basic components of this module are depicted in next figure.

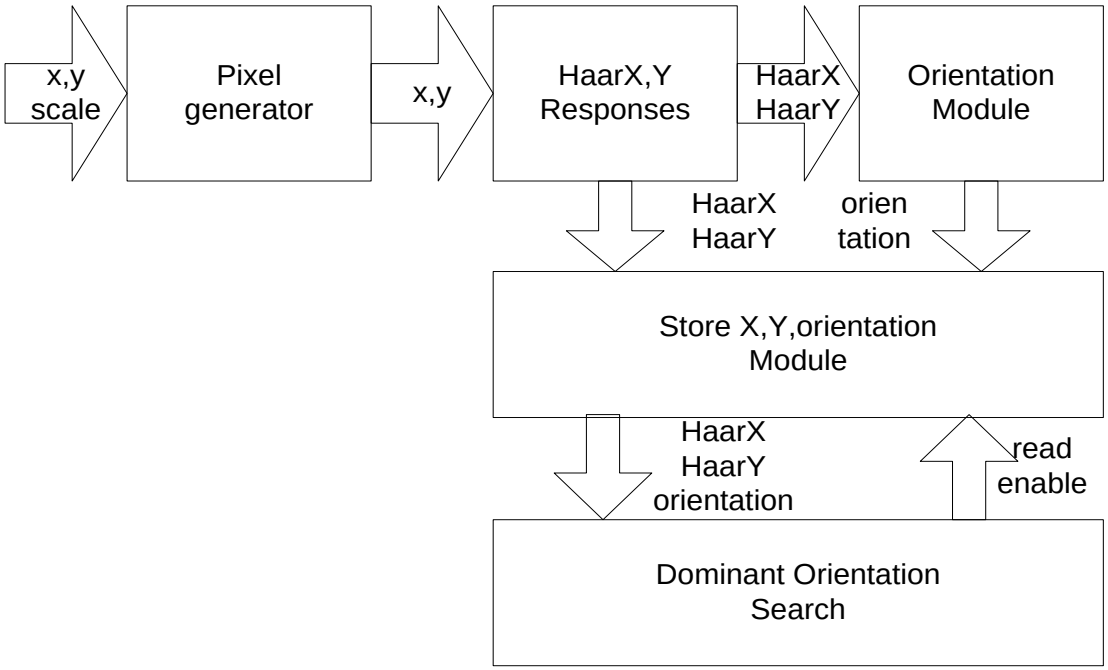


Figure 3.11: Orientation assignment

3.7.1 HaarX,Y Module

The Haar wavelets are shown in Figure 2.7. The size of these filters are 4σ and each haar response either in x or in y dimension requires the intensities of two rectangular areas (boxintegral) and one subtraction. Totally, 16 memory accesses required to compute Haar responses which are weighted finally with the same Gaussian value retrieved by a look-up table based on the pixel position. After analyzing Haar responses, we limit the memory accesses from 16 to 12 by storing some intermediate results. These weighted values are supplied to orientation module and they are also stored in a structure explained later. When a pixel is processed by this module, a pixel generator is enabled and haar responses are calculated for a new pixel.

3.7.2 Pixel Generator Module

We have to compute the orientation and haar responses for all pixels around interest point with step σ in radius of 6σ. This module is an FSM which begins from initial state the position (0,0) and takes all pixels in radius of 6 (for example (0,1),(0,-1) etc). These values are multiplied with the scale in order to take all pixels in a radius

of 6σ and added to the interesting point position. According to absolute values of pixels generated by FSM is calculated the position of the Gaussian value in a look-up table and used by HaarX,Y module. Totally 109 pixels are processed and are supplied to the next module.

3.7.3 Orientation Module

The orientation module finds the angle of a vector (X,Y) where X,Y are the Haar responses in x and in y-dimension respectively. The angle of the vector is calculated using the arctan function. The inputs of the arctan are only positive and the result is based on the quadrant that belongs the vector as shown in Table 3.4.

sign(X)	+	-	+	-
sign(Y)	+	+	-	-
angle	atan(Y/X)	pi-atan(-Y/X)	2*pi-atan(-Y/X)	pi+atan(Y/X)

Table 3.4: Angle of vector(X,Y)

The question is how the arctan is implemented in hardware as it is an expensive function in terms of cost and time. The Xilinx LogicCORE implements a generalized coordinate rotational digital computer (CORDIC) algorithm. ArcTan is selected among others functional configurations. A detailed information about the input and the output data representation is given in product specification of cordic LogicCore [38]. Latency is directly proportional to output width for every arctan calculation. Pipeline implementation of cordic core allow to increase throughput but it requires a buffer to store the signs of X,Y because result have to be adjusted according to Table 3.4. When a new vector is inserted to arctan function module, its sign is stored in a FIFO and when arctan is calculated, we read its sign from FIFO and return the angle of the vector. Implementation of arctan using LogicCORE is equipped with enable and done signals which used to write and read the signs in FIFO buffer respectively.

In addition, inputs of arctan implementation from Xilinx logicCORE belong to [-1, 1] space value. Gaussian weighted HaarX, HaarY responses have a different range which exceeds the previous values. Normalization of these responses is required. This process finds the biggest value between HaarX and HaarY and performs respective right shifts (division by powers of two). In order to limit the impact in accuracy, normalization is performed based on which range belongs the biggest value. For example if it belongs to (1,2] one right shift is performed, (2,4] two right shifts etc.

3.7.4 Store X,Y,Orientation Module

In software implementation all these 109 vector values (X,Y,orientation) are stored in one vector and these vector values are used from a loop which slides a pi/3 window around the feature point with a step of 0.2 in space [0,2π]. The above computation increases significantly the number of memory accesses. In the hardware implementation this vector is stored into seven different memories and seven processing units, run in parallel as will be explained in the next section.

It becomes clear from previous section that Haar responses are calculated before the respective angle. This leads to separate the storing procedure for Haar responses and for orientation. Two different write addresses are stored in registers. Write address is increased for haar response memory when new responses are computed from HaarX,Y module while write address for angle when new angle is produced by orientation module. Also, some logic is used to select which from the seven memories must be selected based on the value of write address. Finally, all 109 vectors and angles are stored in this structure. The last and most complex part of this task is the dominant orientation search. Figure 3.12 represents the seven parts of one vector with 109 values performing fourteen memory accesses and using seven dual-port rams per cycle instead of one.

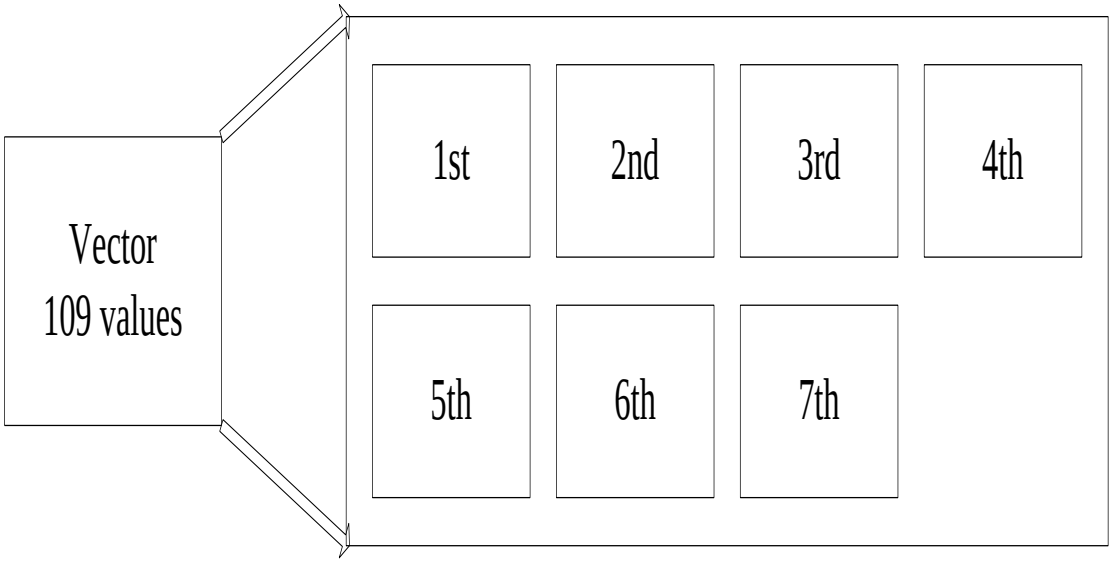


Figure 3.12: Vector partition

3.7.5 Dominant Orientation Module

Firstly, the space [0,2π] is divided in windows of size π/3 beginning from [0,π/3] and increasing by 0.2 until all space is covered. This is simply implemented by using a counter with a step of 0.2.

The one vector in software implementation is separated into seven and it allows a parallel processing. Seven units read the angles and check if they belong to current window, they perform the additions of weighted HaarX, HaarY responses and form a new vector. Also, it is performed a normalization to space [-1, 1] of these sums before arctan implementation computes the angle. If the vector length is longer than previous vectors lengths, this forms the new dominant orientation. This process is repeated for all windows in space $[0,2\pi]$. The vector which forms the dominant orientation is computed. Finally this vector is processed by orientation module and the angle of dominant orientation is calculated.

3.8 Fixed-point representation analysis

Although a floating-point representation can achieve great precision and large dynamic ranges, it is often prohibitive to use floating-point operators in an FPGA implementation because of the large amount of hardware resources required to implement the normalization operations. This becomes critical in hardware vision systems where operators are replicated numerous times and used in parallel to increase the speed of the system.

In vision algorithms, parameters and intermediate results rarely exploit the entire capacity of a floating-point representation. As a result, it is possible to study their precision and dynamic ranges to determine the optimal number of integer and fractional bits that need to be allocated to represent them in fixed point format. Larger bit-widths result in smaller quantization errors, however they require larger operators. In our case floating point representation for one module is used due to the fact that the dynamic range of these numbers is too large.

In general, finding the optimal bit widths for each variable is a complex multivariable optimization problem that requires knowledge of both the algorithm and the target hardware platform. Among others, the following conditions have to be taken into account:

- Some variable have a larger impact on the final results than others, and so the most critical should have larger bit-widths.
- Different input images may yield different optimal widths.
- An efficient implementation must take into account the relative size of the operations associated with each variable and how many times these operations are repeated throughout the circuit, to try to minimize the width of the operations that require most resources.
- In FPGAs, memory blocks and dedicated arithmetic circuitry, such as embedded multipliers and adders, are optimized for signals of certain sizes

(usually powers of 2), often the remaining resources cannot be used for other operations. To avoid waste of resources, signals should be kept in standard sizes whenever possible.

The rest of this section describes the inputs and the outputs of previously explained hardware modules from the point of fixed-point representation. A mixed fixed-point and floating point arithmetic was used.

Initially, floating point implementation normalizes pixel values in space [0,1) by dividing with 256. In fixed point implementation considering the whole pixel value (eight bit) as a fractional part, it is almost equal to divide with 255. Integral image pixels are represented by 32 bit values. The eight least significant bits form the fractional part and the rest twenty four the integer part.

During determinant calculation, a mixed fixed-point arithmetic is implemented. The floating point version of algorithm was used to determine the bit widths of intermediate results for test images. Intermediate results before normalization, after normalization, constant inverse factor and determinant fixed point representation are presented in Table 3.5.

# of bits	sign	integer part	fractional part
constant inverse	0	0	42
Dxx_bef	1	11	8
Dyy_bef	1	11	8
Dxy_bef	1	11	8
Dxx,Dyy,Dxy	1	0	31
determinant	1	0	31

Table 3.5: Determinant Module fixed-point representation

As mentioned previously, Dxx,Dyy,Dxy are normalized to space [0,1). This is done by dividing with a constant relative to the applied filter. Constants are pre-calculated and the range of these numbers requires 42 bits only fractional part as it is always less than one and positive. Therefore, all factors (Dxx,Dyy,Dxy) used in determinant calculation are less than one and the determinant is also less than one. A fractional part of 31 bits is been used and one bit for sign.

The next processing module is the non-maximal suppression. Determinant values are used to perform a series of comparisons. If these comparisons locate an interest point, interpolation module adjusts the location both in scale and in space. As described a matrix inversion is difficult in hardware especially when elements of matrix are less than one due to limited accuracy. Three multiplications of numbers belong to previous space may result a zero value. Fixed point implementation of this module leads to detected interest points which are neighboring with software detected interest points or some interest points were not detected because of required dynamic range of numbers in matrix inversion. Fixed-point to single floating point converters were used, floating point (adders, multipliers, dividers) units were limited to the

minimum required number in terms of hardware resources and float to fixed converters. Determinant values are converted to single precision floating-point numbers and factors of interest point localization from floating point to fixed point. Finally, next Table presents the pixel location and scale value bit widths of detected interest points.

# of bits	integer part	fractional part
x	10	0
y	10	0
scale	7	25

Table 3.6: Interest point representation

Pixel position has a decimal value after localization. However, both orientation and description assignment use a rounded value of pixel which is stored in this form. On the other hand, the scale value is used rounded by orientation assignment and as decimal value by description part.

Orientation is computed for each interest point. The process was described in previous chapters. Inputs and output of arctan function are represented in fixed-point arithmetic and the orientation is also in fixed-point. Input and output ranges and bit widths of arc tan module are shown in Table 3.7. Normalization unit as described in section 3.7.3 scale the inputs of arctan to range [-1, 1]. Inputs of this unit are the Gaussian weighted Haar responses and the summations from Dominant Orienatation module (section 3.7.5). Orientation is the final result of this task.

	Range	sign	integer part	fractional part
X	-1<=X<=1	1	1	30
Y	-1<=Y<=1	1	1	30
arctan(Y/X)	-pi<=phase<=pi	1	2	29
orientation	0<=orientation<=2pi	1	3	28

Table 3.7: Orientation assignment representation

	sign	integer part	fractional part
gaus_HaarX	1	5	27
gaus_HaarY	1	5	27
sumX	1	8	24
sumY	1	8	24
normX	1	1	30
normY	1	1	30

Table 3.8: Normalization Unit representation

Finally, the Gaussian weights are pre-computed and stored into look-up tables as described in previous section. These values are positive and less than one so 25 bits of fractional part are used to represent these numbers.

3.9 Image prefetching

In order to be able to support video at relatively high rates we pipelined the image loading with the image processing tasks. Therefore, we have two distinct memory banks and while we store the input image in one of them we process the previously stored image which has already been placed in the other bank. When the processing (which takes much longer than the image loading) is completed the two banks change roles. The image loading is explained through section 3.2 and 3.3.

Chapter 4

Results

This chapter presents a quantitative evaluation of the performance of the feature detector. Section 4.1 lists the hardware resources used in the system. Section 4.2 compares the computational speed of the system against the speed of software implementation. Finally, Section 4.3 compares the location, scale and orientation of the features detected in hardware with values from floating point software implementation.

4.1 Hardware utilization

Tables 4.1, 4.2 and 4.3 list the amount and kinds of hardware resources used in the system. Table 4.1 and 4.2 list the resources used in each of the modules described in Chapter 3(sections 3.2, 3.3, 3.4, 3.5, 3.6 and 3.7). The numbers in parenthesis indicate the percentage of the total resources of that kind available in Virtex5 vsx240T FPGA. As reference, this FPGA includes 149760 slice registers, LUTs,, 960 bonded IOBs, 516 Block RAM/FIFO, 32 BUFG/BUFGCTRLs and 1056 DSP48Es.

As seen in Table 4.3, the available resources in the Virtex5 vsx240T are not fully utilized except for the on-chip memory, utilizing around 90% available memory. This is due to the fact that the integral of an image with dimension 640x480 requires a large amount of memory. Integral image module is common for detection and orientation task.

Selected Device: Virtex5 vsx240T (speed grade -2)				
Module	**Slice Registers**	**Slice LUTs**	**Block RAM**	**DSP48Es**
Integral Image	365 (0%)	2892 (1%)	290 (56%)	0 (0%)
Grayscale Image	5 (0%)	60 (0%)	80 (15%)	0 (0%)
Retrieve Parameters	10 (0%)	94 (0%)	0 (0%)	0 (0%)
Box Integral Calculation	51 (0%)	281 (0%)	0 (0%)	0 (0%)
Det Calculation	1089 (0%)	818 (0%)	7 (1%)	13 (1%)
Store Determinants	0 (0%)	0 (0%)	24 (4%)	0 (0%)
Comparison Unit	522 (0%)	1150 (0%)	0 (0%)	0 (0%)
I_Point localization	4714 (3%)	4048 (2%)	0 (0%)	12 (1%)

Table 4.1: Resource utilization for Interest point detection sub-modules

Selected Device: Virtex5 vsx240T (speed grade -2)				
Module	**Slice Registers**	**Slice LUTs**	**Block RAM**	**DSP48Es**
Orientation	3208 (2%)	3349(2%)	0 (0%)	0 (0%)
Store X,Y,Orientation + Dominant Orientation	836 (1%)	5204 (3%)	21 (4%)	8 (1%)
HaarXY	141 (1%)	139(1%)	0 (0%)	0 (0%)
Pixel_Generator	23 (1%)	52 (1%)	0 (0%)	0(0%)
Used Multipliers	0 (0%)	0 (0%)	0 (0%)	18(1%)

Table 4.2: Resource utilization for Orientation assignment sub-modules

Selected Device: Virtex5 vsx240T (speed grade -2)					
Module	**Slice Registers**	**Slice LUTs**	**Slice LUT-FF pairs**	**Block RAM**	**DSP48Es**
DualCoreInterest point detection	13447(8%)	17257 (11%)	20310 (13%)	435 (84%)	50 (4%)
Orientation assignment	6832 (4%)	12496 (8%)	12394 (8%)	308 (59%)	18(1%)

Table 4.3: Resource utilization for Virtex5 vsx240T device

Selected Device: Virtex5 vfx200T (speed grade -2)					
Module	**Slice Registers**	**Slice LUTs**	**Slice LUT-FF pairs**	**Block RAM**	**DSP48Es**
DualCoreInterest point detection	13447(10%)	17257 (14%)	20310 (16%)	435 (95%)	50 (13%)
Orientation assignment	6832 (5%)	12496 (10%)	12394 (10%)	308 (67%)	18(4%)

Table 4.4: Resource utilization for Virtex5 vfx200T device

Finally, Table 4.4 presents the hardware utilization in virtex5 vfx200T device. This device is cheaper than vsx240T and it is obvious that our design reach the 95% of available BRAMs instead of 84% in vsx240T.

4.2 Performance

One of the most important measures in the evaluation of a hardware implementation is how it compares to a software implementation in terms of speed.

Table 4.5 presents the overall speedup and Table 4.6 shows the processing time for the hardware system and the floating-point software implementation for the case of six different images as far as interest point detection. Table 4.7 refers to orientation assignment time. The hardware processing time refers to a post place and

route implementation of the proposing scheme in the Virtex5 vsx240T device. The processing times for the software implementation were measured using Vtune (a commercial application for software performance analysis) running on a 2.4 GHz Pentium IV processor with a 4 GB of memory. The processing time of the hardware were computed for clock rates of 198.464 MHz for the dual core interest point detection and orientation tasks using the next formula. Clock rate remains the same for both devices.

$$Hardware\ time = (\#\ of\ cycles)\ x\ (\frac{1}{clock\ rate})$$

Input	Interest Points Speedup	Orient. Speedup	Overall Speedup
image_one	10,7	9,3	9,9
image_two	8,4	8,9	8,7
image_three	8,3	8,1	8,2
image_four	9,8	8	8,7
image_five	9,2	10,9	10,1
image_six	8,7	9,8	9,3

Table 4.5: Speedup

4.3 Accuracy

The fixed-point arithmetic used in the hardware implementation of the feature detector introduces errors in location, scale and orientation of the detected features. The main reason for these errors in the detected interest points is that some rounding errors occurred during determinant calculation. This was due to our slightly different accuracy used when implementing the three multiplications in equation 2.6. To provide a quantitive measure of these errors, the results obtained from Modelsim simulation of six test images, shown in Figure 4.1 and 4.2, were compared against the results obtained from processing the same images with the original floating-point software implementation.

Table 4.6 shows the numbers of features detected in each test image by the software (it also follows the modification of division by the factor of 256 for pixel intensity) and hardware implementation. Moreover, the total number of common detected points (position takes integer value) and the variance of scale for the common points (decimal value) are computed. Finally, the variance of orientation for the common interest points is also computed and orientation results presented in Table 4.7. After examining the results for numerous 640 x 480 images we have figured that in all cases the number of missing or additional points of interest reported by our

approach was always less than 1% of the overall reported images. At the same time the scale and orientation error introduced was always less that 0.01%.

Input	Hardware time (ms)	Software time (ms)	Points of interest (hw)	Points of interest (sw)	commons	scale_variance
image_one	7.551 ms	80.97ms	1634	1635	1609	0.0014744
image_two	7.555 ms	63.5 ms	1681	1677	1666	6.24121x10^-5
image_three	7.561 ms	62.47ms	1722	1713	1704	0.000524633
image_four	7.550 ms	73.71ms	1823	1822	1808	0.000640449
image_five	7.555 ms	69.47ms	1508	1502	1493	0.00038069
image_six	7.55 ms	65.4 ms	1704	1708	1693	0.000785097

Table 4.6: Interest point results

Input	Hardware time (ms)	Software time (ms)	Orientation Variance
image_one	9.902 ms	91.72 ms	0.00579719
image_two	10.255 ms	90.82 ms	0.0199012
image_three	10.494 ms	84.8 ms	0.01785
image_four	11.165 ms	89.4 ms	0.0169975
image_five	9.190 ms	100 ms	0.0296172
image_six	10.45 ms	102.5 ms	0.0245929

Table 4.7: Orientation Assignment results

These results are also compared to the initial software implementation (where pixel intensity is divided by 255). In this case, the missing points are 6% of overall detected points while scale and orientation errors are always less than 0.1%. It was difficult to predict the repercussions of these errors in performance of overall scheme because feature description is not yet implemented and we can't evaluate the effect in matching of features between different images. In chapter 5, we propose a slight different implementation which affects only one functional module and will improve our results.

4.4 Post place and Route verification

We performed a post-place and route simulation otherwise known as timing simulation on our design to verify that the functionality is correct after the place and

route. This process uses the post-place and route simulation model (a structural SIMPRIM-based VHDL file) and a standard delay format (SDF) file generated by NetGen. The SDF file contains true timing delay information of the design.

After getting the timing simulation model of our design we were able to run the same tetbenches we ran during the functional verification of the design and verify the correct results.

4.5 Tested Images

We tested our detector using images provides by Mikolajczyk [40] resized to 640x480 dimension and converted to a grayscale image. Second image set presents first image set under different viewing conditions such as rotation, light and zoom.

Figure 4.1: Image set one

Figure 4.2: Image set two

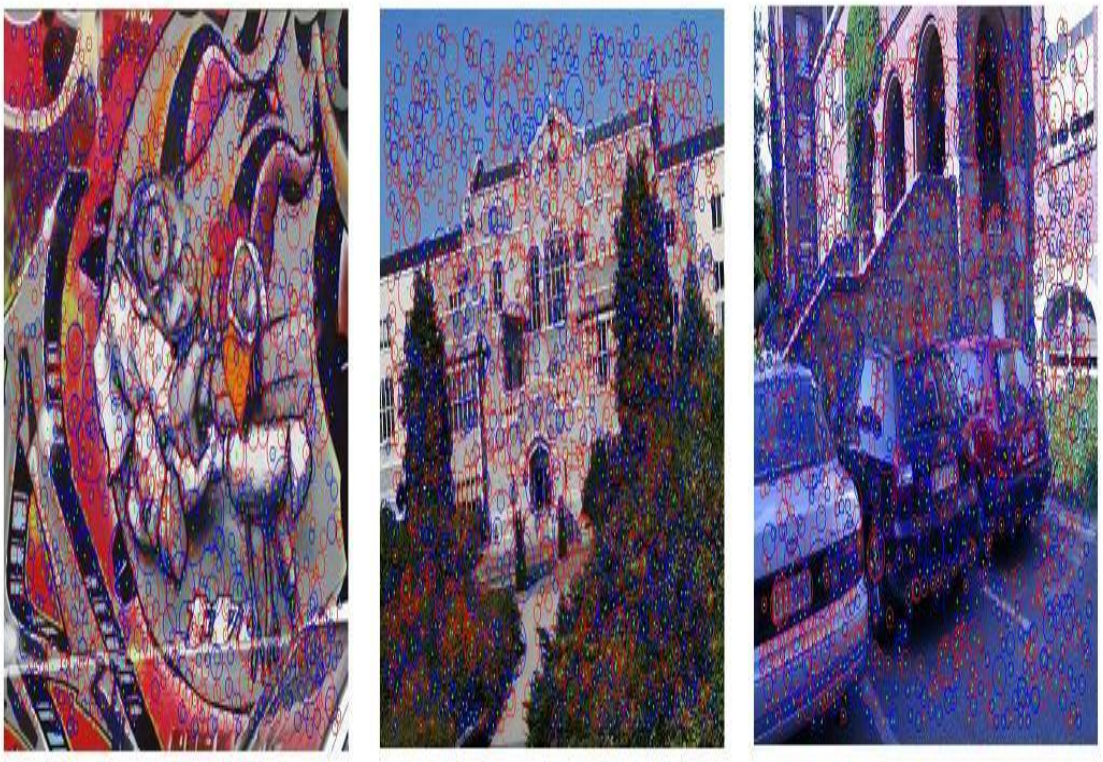


Figure 4.3: Software Interest points

Figure 4.4: Hardware Interest points

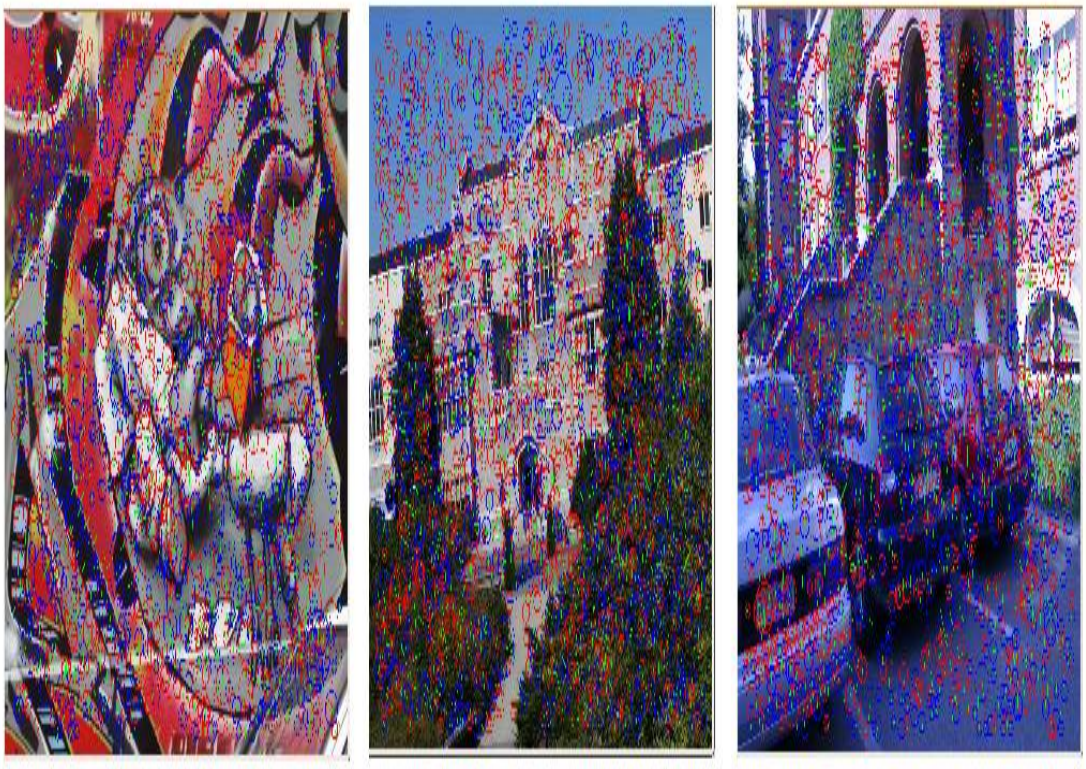


Figure 4.5: Software Interest points (oriented)

Figure 4.6: Hardware Interest points (oriented)

Chapter 5

Conclusions and Future Work

This thesis presents an FPGA-based implementation of SURF detector presented in [1]. The feature detector provides the location, scale and orientation of interest points in frames at a rate of 130 frames per second for detection part and at a rate of 95 frames per second for orientation assignment part.

5.1 Summary of Contributions

This is, to the best knowledge of the author, the first implementation of this scheme in an FPGA. Our innovative system can support processing of standard video (640 x 480 pixels) at up to 56 frames per second while it outperforms a state-of-the-art Intel CPU by at least 8 times. Moreover, the proposed system, which is clocked at 200MHz and consumes less than 20W, supports a frame rate which is less that 20% lower than that of a high-end GPU executing the same basic algorithm, the specified GPU consists of 128 floating point CPUs, clocked at 1.35GHz and consumes more than 200W.

The SURF detector produces features that are robust to image rotation and translation, as well as to significant changes in illumination and scale deformations. This performance, however, doesn't provide real-time processing. The fact many operations in hardware can be performed in parallel make it an ideal candidate for implementation.

Field-programmable gate arrays (FPGAs) offer a good hardware development platform because they can be easily reprogrammed to fit the requirements of a given application. This is particularly useful in the early stages of implementation of a new design since correction and additional features can be added in short periods of time.

The main challenge with implementing a complex algorithm in hardware is to balance the need for numerical precision with an efficient use of the available hardware resources. This implementation of the feature detector uses a combination of fixed-point and floating-point numerical representation limiting the number of floating-point units in terms of hardware resources. The results obtained from comparing the hardware implementation and the floating-point software model show that the location, scale and orientation achieve results with high-accuracy.

Comparison with other schemes	Performance in (fps)	Detection	Orientation	Description
CPU(2.4 GHz) original software	6.5 (640x480)	✓	✓	×
GPU-based SURF	85 (640x480, 500 features)	✓	✓	✓
Multi-core CPU SURF	33 (640x480)	✓	✓	✓
FPGA-based SIFT on exomars robot	16.5 (640x480)	✓	×	×
GPU-based SIFT	33 fps (1000 features, 1024x768)	✓	×	×
Our FPGA-based SURF	56 fps (640x480, 1500+ features)	✓	✓	×

Table 5.1 Performance comparison

Table 5.1 describes mainly the performance in terms of speed with other recently implemented detection systems and our software implementation. The number of detected features affects the processing time for orientation task directly proportional. Consequently, the number of detected interest points is reduced using an appropriate value of threshold and processing time is reduced only for orientation task as the detection is independent of this value because it is used by modules which are executed in parallel. More details are given in hardware design of section 3.5.

5.2 Future Work

There is enough room for improvements and refinements, both in hardware resources and in the processing speed. The following are a few ideas to guide any future work on this system.

- Accuracy in detected points as mentioned in section 4.3 can be improved. Determinant module (section 3.4) uses the results of boxintegral function which compute the intensity of any rectangular area. Dxx, Dyy, Dxy are intermediate results with eight bits fractional part (division by 256). We can concern these values as integer numbers without fractional part and perform division by 255 (multiplication with the inverse number of 255), increasing the bits of fractional part. We use only extra multipliers and multipliers with extra input bits. This module is also pipelined and doesn't affect the overall processing time. This modification will limit the errors in number of detected points.
- 'Store' Determinant module (section 3.5.1) utilizes a large amount of on-chip memory. As mentioned in section 3.5.1, determinant response map is divided into 3x3 regions and the neighboring pixel determinants are needed to perform interest point localization. This implementation stores all determinants for five

rows and interest point localization module divides these five rows into 3x3 regions. It becomes obvious that only five columns are needed to perform the localization of a 3x3 region. Thus, columns are divided into a similar way as five rows. This modification can reduce the used on-chip memory by a factor of 70%.

- Use of floating point units is expensive in terms of hardware resources. The two cores of interest point detection use two modules of interest point localization which consist of floating point units. It becomes obvious from simulations that the two cores can use the same interest point localization module as these processing units of two cores never function at the same time. Thus, reduction of hardware resources is achieved.
- Interest point detection in first octave is the most consuming part of detection for three octaves (5.8 ms from 7.55 ms for first test image). Moreover, applied box filters for the first octave require less previous and next rows for the current row determinant calculation. This conclusion in combination with the decreased on-chip memory of 'store' determinants module and an efficient use of interest point localization module can parallelize the detection only for the first octave in four parts. Thus, the detection time for first octave (5.8 ms) can be reduced by a factor around two.
- Efficient use of orientation module (section 3.7.3). Orientation module is a pipelined unit and it is used every 12 cycles when new HaarX, HaarY (section 3.7.3) are computed or every 8 cycles when new sumX, sumY (section 3.7.5) are computed. As orientation module is the most expensive part of orientation assignment, we have to take advantage of the pipeline. Image is already partitioned and we can have two orientation assignment cores using the same arctan unit. This leads to improvement of time performance by factor of two using only more DSP multipliers which can also be limited as these units aren't be used every cycle. Also, more than 65% of interest points belong to first octave. For this octave, it is possible to process the interest points from the four parts mentioned in previous bullet, parallel. Thus, almost 2/3 of the time required for orientation assignment can be reduced by a factor of 4.
- Description part of SURF.
- Combination with a classification system.
- Combination of detector with video interfaces to create a smart camera system.

References

[1] H. Bay, T. Tuytelaars, and L. Van Gool. Surf: Speeded up robust features. European Conference on Computer Vision, 2006.

[2] H. Bay, Andreas Ess, T. Tuytelaars, and L. Van Gool. SURF: Speeded up robust features, Computer Vision and Image Understanding (CVIU), June 2008.

[3] H. Moravec. Rover visual obstacle avoidance. In Proceedings of the Seventh International Joint Conference on Artificial Intelligence, August 1981.

[4] D. G. Lowe. Object recognition from local scale-invariant features. In Proceedings of the International Conference on Computer Vision ICCV, Corfu, Greece, 1999.

[5] D. G. Lowe. Distinctive image features from scale-invariant keypoints. International Journal of Computer Vision, 2004.

[6] E. H. Adelson, C. H. Anderson, J. R. Bergen, P. J. Burt, and J. M. Ogden. Pyramid method in image processing. RCA Engineer, 1984.

[7] J. L. Crowley and A. C. Parker. A representation for shape based on peaks and ridges in the difference of low-pass transform. IEEE Transactions on Pattern Analysis and Machine Intelligence, March 1984.

[8] L. Bretzner and T. Lindeberg. Feature tracking with automatic selection of spatial scales. Computer Vision and Image Understanding, September 1998.

[9] T. Lindeberg. Scale-space theory: A basic tool for analysing structures at different scales. Journal of Applied Statistics, 1994.

[10] T. Lindeberg. Detecting salient blob-like image structures and their scales with a scale-space primal sketch: A method for focus-of-attention. International Journal of Computer Vision, December 1993.

[11] T. Lindeberg. Feature detection with automatic scale selection. International Journal of Computer Vision, 1998.

[12] C. Harris and M. Stephens. A combined corner and edge detector. In *ProceedingsFourth Alvey Vision Conference*, Manchester, United Kingdom, 1988.

[13] K. Mikolajczyk and C. Schmid. Indexing based on scale invariant interest points. In IEEE International Conference on Computer Vision, 2001.

[14] A. Shokoufandeh, I. Marsic, and S. Dickinson. View-based object recognition using saliency maps. Image and Vision Computing, 1999.

[15] Paul Viola and Michael Jones. Rapid object detection using a boosted cascade of simple features. cvpr, 2001.

[16] http://web.mat.bham.ac.uk/C.Good/teach/2a/notes/2asec044.pdf

[17] J.J. Koenderink. The structure of images. Biological Cybernetics, 1984.

[18] T. Lindeberg. Scale-space for discrete signals. PAMI, 1990.

[19] http://www.nada.kth.se/~tony/cern-review/cern-html/

[20] A. Witkin. Scale-space _ltering, int. joint conf. Artif. Intell, 1983.

[21] W. J. MacLean. An evaluation of the suitability of FPGAs for embedded vision systems. In The First IEEE Workshop on Embedded Computer Vision, CVPR 2005, New York, June 2005.

[22] http://www.eecg.toronto.edu/~jayar/pubs/brown/survey.pdf
Architecture of FPGAs and CPLDs: A Tutorial Stephen Brown and Jonathan Rose.

[23] A. Darabiha, J. Rose, and W. J. MacLean. Recon¯gurable hardware implementation of a phase-correlation stereo algorithm. Machine Vision and Applications, 2006.

[24] D. K. Masrani and W. J. MacLean. A real-time large disparity range stereo system using FPGAs. In 7th Asian Conference on Computer Vision, volume 3852 of Lecture Notes in Computer Science, 2006.

[25] J. Schlessman, C.-Y. Chen, W. Wolf, B. Ozer, K. Fujino, and K. Itoh. Hardware/software co-design of an FPGA-based embedded tracking system. 2nd Workshop on Embedded Computer Vision, CVPR , 2006.

[26] M. Leeser, S. Miller, and H. Yu. Smart camera based on reconfigurable hardware enables diverse real-time applications. 12th Annual IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM'04), 2004

[27] M. Sen, I. Corretjer, F. Haim, S. Saha, S. S. Bhattacharyya, J. Schlessman, and W. Wolf. Computer vision on FPGAs: Design methodology and its application to gesture recognition. 1st Workshop on Embedded Computer Vision, CVPR 2005, 2005.

[28] D. Nguyen, D. Halupka, P. Aarabi, and A. Sheikholeslami. Real-time face detection and lip feature extraction using Field-programmable gate arrays. IEEE Transactions on Systems, Man and Cybernetics, August 2006.

[29] C. Tomasi and T. Kanade. Detection and tracking of point features. Technical Report CMU-CS-91-132, Carnegie Mellon University, April 1991.

[30] A. Benedetti and P. Perona. Real-time 2-d feature detection on a reconfigurable computer. In Conference on Computer Vision and Pattern Recognition, 1998.

[31] P. Fiore, D. Kottke, and D. Gampagna. E±cient feature tracking with application to camera motion estimation. In Conference Record of the 32nd Asilomar Conference on Signals, Systems & Computers, volume 2, November 1998.

[32] P. Giacon, S. Saggin, G. Tomasi, and M. Busti. Implementing DSP algorithms using Spartan-3 FPGAs. Xcell Journal, 2005.

[33] A. Bissacco and S. Ghiasi. Fast visual feature selection and tracking in a hybrid reconfigurable architecture. In 2nd Workshop on Applications of Computer Vision, ECCV 2006, Gratz, May 2006.

[34] S. Se, T. Barfoot, and P. Jasiobedzki. Visual motion estimation and terrain modeling for planetary rovers. In 8th International Symposium on Artificial Intelligence, Robotics and Automation in Space, Munich, 2005.

[35] C.-S. Bouganis, P. Y. K. Cheung, J. Ng, and A. A. Bharath. A steerable com-plex wavelet construction and its implementation on FPGA. In 14th International Conference on Field Programmable Logic and Applications, volume 3203 of Lecture Notes in Computer Science, January 2004.

[36] GPU-based Video Feature Tracking And Matching Sudipta N. Sinha, Jan-Michael Frahm, Marc Pollefeys, Yakup Genc. EDGE 2006, workshop on Edge Computing Using New Commodity Architectures, Chapel Hill, May 2006.

[37] GPU Accelerating Speeded-Up Robust Features Timothy B. Terriberry, Lindley M. French, and John Helmsen. Proceedings of 3DPVT'08 - the Fourth International Symposium on 3D Data Processing, Visualization and Transmission.

[38] http://www.xilinx.com/support/documentation/ip_documentation/cordic_ds249.pdf

[39] M. Brown and D.G. Lowe. Invariant features from interest point groups. British Machine Vision Conference, Cardiff, Wales, 2002.

[40] http://www.robots.ox.ac.uk/~vgg/research/affine/

[41] http://www.springerlink.com/content/y76446762147v68t/

Nan Zhang. Computing Optimised Parallel Speeded-Up Robust Features (P-SURF) on Multi-Core Processors

[42] Open source SURF feature extraction library, http://code.google.com/p/opensurf1/

[43] Evaluation of Interest Point Detectors, CORDELIA SCHMID, ROGER MOHR AND CHRISTIAN BAUCKHAGE, International Journal of Computer Vision, 2000 Kluwer Academic Publishers. Manufactured in The Netherlands.